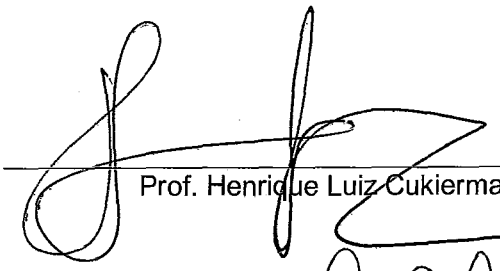
UM OLHAR SOCIOTÉCNICO SOBRE A ENGENHARIA DE SOFTWARE:
O CASO DO BNDES

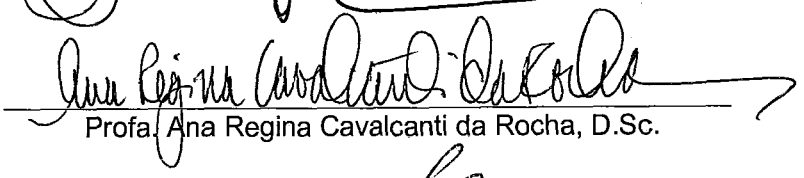
Cássio Adriano Nunes Teixeira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

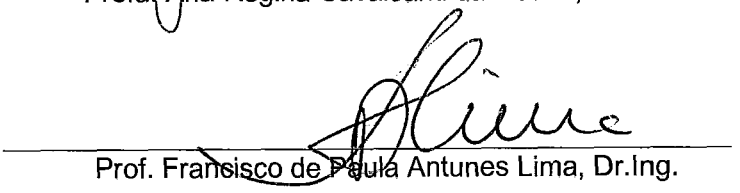
Aprovada por:



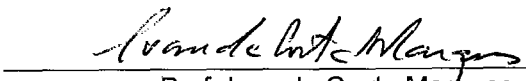
Prof. Henrique Luiz Cukierman, D.Sc.



Profa. Ana Regina Cavalcanti da Rocha, D.Sc.



Prof. Francisco de Paula Antunes Lima, Dr.Ing.



Prof. Ivan da Costa Marques, Ph.D.

RIO DE JANEIRO, RJ — BRASIL
SETEMBRO DE 2007

TEIXEIRA, CÁSSIO ADRIANO NUNES

Um olhar sociotécnico sobre a engenharia de software: o caso do BNDES [Rio de Janeiro] 2007

IX, 168 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Engenharia de software. 2. Redes Sociotécnicas. 3. História da Engenharia de Software.

I. COPPE/UFRJ II. Título (série)

DEDICATÓRIA:

Naquele final de 2003, em Formiga (MG), quando recebi o e-mail do Prof. Henrique comunicando minha aceitação no programa de mestrado, meu bom pai, Olemar Teixeira, efusivamente compartilhou de minha alegria. Em sua simplicidade, perguntou: "mas, exatamente, do que trata a engenharia de software?" Hoje, quando termino esta empreitada e teria algo mais para respondê-lo, quis o destino que eu não estivesse mais desfrutando aqui de sua companhia...

É a ti, meu querido pai, que dedico o resultado deste esforço. Por tudo que me foste ao longo da vida, com admiração, gratidão e orgulho, reconheço que há muito teu aqui também!

AGRADECIMENTOS

À Margarida Sá Freire e ao Sérgio Viveiros, os executivos do BNDES que viabilizaram meu ingresso neste esforço.

Ao Alexandre Pereira, ao Izidro Neto e ao José Marcos Gonçalves, também executivos com quem trabalhei, que, assim como o Sérgio e a Margarida, me apoiaram ao longo desta jornada que demorou mais do que o inicialmente previsto.

Aos colegas do BNDES entrevistados pela grande ajuda e boa vontade.

Ao Gilmar Oliveira e ao Paulo Barreto da GDOC/BNDES pela presteza na recuperação dos documentos arquivados.

Ao amigo César Gil, pela ajuda na revisão final.

Ao amigo José Ricardo e à "tia" Mônica por tantos finais de semana que foram companhias para "minhas duas pequenas" e, como não lembrar, por aqueles maravilhosos tampões de ouvido que contribuíram para noites melhores dormidas.

Ao amigo Marcelo Sávio, companheiro que conheci na COPPE, por tantas dicas valiosas e ajudas ao longo destes anos.

Ao amigo Prof. Antônio Sérgio, do DEP/UFMG, meu primeiro orientador acadêmico.

Ao amigo Dr. Eduardo Duarte pelo apoio preciso, sempre fraterno, em vários momentos difíceis.

À minha mãe, Ana Cecília, e irmãos, Daniel e André, pelas vibrações positivas.

Ao amigo Prof. Henrique Cukierman pelo enorme compromisso e seriedade com que tanto me ajudou, viabilizando a conclusão deste trabalho.

Finalmente, muito obrigado à querida esposa Marta, pela paciência, compreensão e apoio, e à pequenina Maria Cecília, filha que retribuía minhas ausências, por conta deste trabalho, com uma alegria incondicional, contagiante e revigorante.

O mundo não é, o mundo está sendo

Paulo Freire (1921 – 1997)

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM OLHAR SOCIOTÉCNICO SOBRE A ENGENHARIA DE SOFTWARE: O CASO DO BNDES

Cássio Adriano Nunes Teixeira

Setembro/2007

Orientador: Henrique Luiz Cukierman

Programa: Engenharia de Sistemas e Computação

É possível identificar um certo paradoxo na engenharia de software. Por um lado, seu tecnocentrismo produz um enquadramento de seu espaço de problemas e soluções; por outro lado, vários "fatores não-técnicos", como são referenciados, supostamente fora desse enquadramento tecnocêntrico, são cada vez mais reconhecidos como determinantes para o sucesso dos projetos de software. Cria-se uma dicotomia — "técnico x não-técnico", ou "dentro x fora" — em que questões reconhecidas como determinantes para os projetos acabam classificadas, *a priori*, como fora do alcance da engenharia de software, sendo relegadas a outras disciplinas. Este cenário dificulta o surgimento de problematizações — espaço de problemas e soluções — sob referenciais alternativos que poderiam, também, contribuir para o desenvolvimento e a prática da engenharia de software, principalmente no que tem transbordado de seu enquadramento usual. Esta dissertação abordará esse "paradoxo", ou abordará as chamadas questões "não-técnicas" relacionadas com a engenharia de software, através da explicitação do paradigma que a induz dicotômica e paradoxal, especialmente examinando sua fundação histórica e o caso do Banco Nacional de Desenvolvimento Econômico e Social (BNDES).

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A SOCIOTECHNICAL VIEW OF SOFTWARE ENGINEERING:
THE BNDES CASE

Cássio Adriano Nunes Teixeira

September /2007

Advisor: Henrique Luiz Cukierman

Department: Computer Science and System Engineering

There is a certain paradox in Software Engineering. Its technocentric bias produces one framing of its space of problems and solutions; meanwhile, several so-called "non-technical factors", supposedly outside this technocentric framing, are ever time more acknowledged as determinant for a software project success. A dichotomy — "technical x non-technical", or "inside x outside" — is created, and questions recognized as determinative to software projects end up being classified, a priori, as out of Software Engineering scope, so remaining relegated to other disciplines. Therefore, it is difficult to consider problems and solutions under alternative approaches that could also contribute to software engineering development and practice, mainly inspired by "questions" that overflow from its normal technocentric framing. This dissertation will approach that "paradox" by expliciting the paradigm that causes Software Engineering to be paradoxical and dichotomics, specially by examining its historical foundation and the Brazilian Development Bank (BNDES) case.

SUMÁRIO

Lista de Figuras

Capítulo 1
Introdução

Os engenheiros de software são cada vez mais confrontados com os "fatores não técnicos" que insistem em malograr tantos projetos de software, sejam eles projetos de desenvolvimento, manutenção e implantação de sistemas, sejam de implantação e melhoria de processos de software. Nas palavras de du Plooy:

> A alta taxa das chamadas "falhas" de sistemas de informação (sistemas abandonados antes de serem totalmente desenvolvidos, desenvolvidos mas nunca utilizados, imensos estouros de orçamentos e de cronogramas, p.ex.) tem sido preocupação desde os primeiros dias dos sistemas de informação baseados em computadores. É geralmente aceito que a maioria dessas falhas, senão todas, não pode ser atribuída à falta de ferramentas e de técnicas, mas à negligência dos [chamados] "fatores humanos" nas práticas dominantes de análise e projeto de sistemas.[i] (du PLOOY, 2003, p.43).

Nosso problema, o dos engenheiros de software, e aqui peço licença para usar a primeira pessoa e incluir-me no rol deles, parece ser buscar insumos em outras disciplinas para tratar os "fatores humanos", ou "não-técnicos," que nos atrapalham na aplicação dos preceitos da disciplina da engenharia de software. No entanto, algumas dúvidas nos pesam na consciência de praticantes: se os tais "fatores não-técnicos" tantas vezes determinam o sucesso ou fracasso dos projetos, por que são relegados a segundo plano na literatura da engenharia de software? Por que a própria engenharia de software não lida com os desafios relacionados aos "fatores não-técnicos"? Por que, mesmo se acreditarmos necessário prover o "contexto" adequado para os projetos, percebemos diante dos problemas práticos um distanciamento da engenharia de software teórica? Por que tantas vezes renegamos o valor de nossas próprias práticas em favor da validade inconteste dos princípios, modelos, padrões, teorias e ferramentas estabelecidos?

Sensibilizado (ainda com a licença de uso da primeira pessoa) por questões desse tipo, a motivação inicial para esta dissertação parecia ser lidar com determinantes "não-técnicos" para problemas práticos da engenharia de software. Ao longo de sua execução, no entanto, graças ao instrumental teórico-metodológico dos *Estudos CTS (Ciência-Tecnologia-Sociedade)*, introduzido mais adiante, começou a se fortalecer a percepção de que a demanda para tratar os "fatores não-técnicos" decorria da visão hegemônica modernista que subjaz à engenharia de software. Por, aparentemente, não existirem textos no Brasil que, focalizando a engenharia de software, discutam acerca dessa visão, parece oportuno fazê-lo, buscando os fundamentos históricos que nos levam a aceitar a divisão dos problemas em

"técnicos" e "não-técnicos" e a estabelecer *a priori* o que está "dentro" e o que está "fora" da engenharia de software.

Sem qualquer pretensão, diria mesmo sem o fôlego necessário, para propor alguma alternativa prática, a presente investigação apenas trilhou um caminho que pudesse explicar como entram em cena os chamados "fatores não-técnicos", delineando um enquadramento que surge a partir da submissão da engenharia de software, como de resto toda a ciência e tecnologia ocidental, a um *estilo de pensamento*[1] dominante. Para lográ-lo, esta dissertação acabou saindo dos moldes do que tradicionalmente é esperado: (1) a apresentação de uma revisão bibliográfica (o "quadro teórico"); (2) a delimitação de um problema, de conceitos e de variáveis; (3) o estabelecimento de proposições, a formulação de hipóteses e de objetivos específicos; (4) a escolha de uma metodologia para coleta de dados; (5) a apresentação dos dados obtidos e sua análise; e, finalmente, (6) a apresentação das conclusões, com uma avaliação crítica dos resultados.

Abriu-se mão aqui de um projeto de pesquisa que tentasse avançar sobre um determinado problema que permitisse, à guisa de conclusão, construir alguma "solução" relacionada aos tais "fatores não-técnicos". O que se buscou, ainda que parcial e incompletamente, foi, a partir da historicidade da engenharia de software, vislumbrar sua fundação moderna, ou, dito de outra forma, o *estilo de pensamento* que tutela seu espaço de problemas e de soluções. Assim torna-se possível compreender que *atribuir o desenvolvimento de software a uma questão de engenharia* não faz parte da "ordem natural das coisas", nem é fruto de uma "evidência natural", mas sim resulta de uma construção histórica.

Michel Callon (1999) define *problematização* não como a mera redução de um problema a um conjunto de questões. Mais do que isso, problematizar é também considerar as diversas entidades cujo modo de existência se dá a partir do problema estabelecido e propor alguma identidade e atuação para tais entidades. Neste sentido, a engenharia de software tradicionalmente formula seus problemas atribuindo ao "não-técnico" o papel de uma entidade que lhe é externa, ou seja, como uma entidade que não lhe pertence. Julgou-se que seria uma contribuição pertinente desta dissertação apontar essa problematização tradicional da engenharia de software, sugerindo uma alternativa que não a da submissão da engenharia de software ao pensamento moderno. Por fim, como todo mundo fala a partir de, ou enquadrado por,

---

[1] Ludwik Fleck define o *estilo de pensamento* como "*uma coerção determinada de pensamento e mais ainda: a totalidade da preparação e disponibilidade intelectual orientada a ver e atuar de uma forma e não de outra. A dependência de qualquer fato científico* [e artefato tecnológico] *ao estilo de pensamento*

seu *estilo de pensamento*, julgou-se que seria válida uma contribuição que facilitasse a ruptura com certos legados, posto que, apesar do grande desenvolvimento da engenharia de software e de sua consolidação como disciplina autônoma, os projetos de software ainda continuam sob o jugo de orçamentos que insistem em estourar, de prazos que não se deixam cumprir, de necessidades de usuários não atendidas, e de níveis de qualidade e confiabilidade dos produtos abaixo do requerido.

Renato Dagnino, sustentando uma bandeira alternativa ao pensamento, desenvolvimento e prática hegemônicos nas engenharias (mecânica, metalúrgica, produção, etc.), e na pesquisa e desenvolvimento (P&D), explicita que muitas vezes se vê confrontado com pedidos imediatos de soluções alternativas "concretas" para os problemas, reconhecendo porém que, no curto prazo, é preciso lidar com a verdade de que não as dispomos, tendo valor, *per si*, o esforço de desconstruir o que herdamos:

> De fato, se pedirmos a um engenheiro que projete uma tecnologia ele vai fazê-lo com as ferramentas que conhece. De acordo com o marco analítico-conceitual de que dispõe, que é aquele predominante no ambiente em que foi formado. O que cai fora desse marco — a tal tecnologia coerente com o estilo alternativo de desenvolvimento — não tem solução. E a tecnologia não tem como ser projetada. Se ele não sabe como introduzir na sua "planilha de cálculo" com a qual está acostumado a trabalhar os parâmetros técnicos e econômicos associados ao "custo" de condenar um trabalhador a trinta anos de "trabalho forçado" numa fábrica onde ele apenas "aperta botões", do desemprego, da degradação ambiental, da obsolescência planejada, do controle predatório da mão-de-obra, etc., ele não conseguirá atender a nossa demanda. O que sim sabemos, e temos dito é que até que tenhamos o marco analítico-conceitual capaz de dar resposta àquelas perguntas, temos que construir um modelo cognitivo alternativo ao hoje existente e, antes disso, *temos que desconstruir o que herdamos.* (DAGNINO; NOVAES, 2006, p. 85, grifo nosso).

Decorrente de sua própria história, a engenharia de software permanece sempre em busca da descoberta do grande avanço tecnológico que a resgatará de suas dificuldades, vivendo uma situação paradoxal, pois, por um lado, enquadra seu espaço de problemas e de soluções tutelada pelo *estilo de pensamento* que a domina e que contribuiu para seu estabelecimento como disciplina autônoma. Por outro lado, o que transborda deste enquadramento, o "fora" da engenharia de software, o "não-técnico", tem sido cada vez mais considerado de grande importância por praticantes, pesquisadores e pela própria literatura. Ou seja, paradoxalmente, são consideradas *a priori* "fora" da engenharia de software algumas questões que ela mesma ressalta como determinante no sucesso ou fracasso de seus projetos, e consideradas "dentro" apenas as abordagens tecnocêntricas que têm permitido seu avanço. Isso dificulta, sobremaneira, o surgimento de problematizações — espaço de problemas e de

---

é evidente." Não se trata simplesmente, portanto, de um tom particular dos conceitos ou uma forma peculiar de reuní-los. (FLECK, 1986, p.111).

soluções — sob referenciais alternativos que poderiam, também, contribuir para o desenvolvimento e a prática da engenharia de software. A dicotomia "técnico x não-técnico" serve para sustentar um discurso que mantém seu *status quo* tecnocêntrico, relegando o "não-técnico", via de regra o culpado pelos fracassos nos projetos, ao âmbito das questões a serem tratadas exclusivamente por outras disciplinas.

Como já comentado, originalmente a idéia central desta dissertação era a de trabalhar sobre "a relevância dos fatores não-técnicos" para o sucesso dos projetos de software. Contudo, ao utilizar a idéia das *redes sociotécnicas* como fio condutor, tornou-se problemática a interpretação de que *existem "fatores não-técnicos", supostamente exteriores à engenharia de software*. Com o avanço da pesquisa, acabou-se trocando uma pergunta do tipo "por que os 'fatores não-técnicos', mesmo aceitos como determinantes para o sucesso dos projetos de software, são relegados a segundo plano?" por outra do tipo "por que cremos, nós, os engenheiros de software, que existe algo 'não-técnico' a 'influenciar' a *difusão* dos preceitos 'técnicos' da engenharia de software em nossos projetos?". Se a primeira pergunta continua nos levando à busca de soluções tutelados pelo *estilo de pensamento* hegemônico na engenharia de software, a última pergunta viabiliza uma linha de investigação em engenharia de software com menores restrições apriorísticas, e, portanto, mais favorável à busca de sínteses brasileiras para os problemas da nossa realidade.

A questão da "influência dos aspectos não-técnicos" será abordada de forma diferente da rotineiramente encontrada na literatura da engenharia de software, concentrada majoritariamente em fatores arquetípicos cuja preexistência no contexto dos projetos de software é fator determinante para a obtenção do "sucesso" — por exemplo, nas reiteradas considerações sobre a necessidade de patrocínio, comprometimento, motivação e envolvimento, dentre outros —. Essa questão será aqui abordada buscando-se o porquê da classificação e divisão de fatores em "técnicos" e "não-técnicos". Por que tal divisão existe? Qual enquadramento resulta dessa divisão que estabelece, rigidamente, um "dentro" e um "fora" da engenharia de software, uma classificação do que é, ou não é, problema da engenharia de software, um julgamento sobre quais são, ou não são, soluções pertinentes à engenharia de software? Na prática dos engenheiros de software, como se comportam essas divisões rígidas entre "dentro" e "fora", "técnico" e "não-técnico"? Os problemas no dia-a-dia do engenheiro de software vêm com etiquetas identificando-os como "técnicos" ou "não-técnicos"? Esboços de respostas serão traçados a partir da explicitação dos vínculos da engenharia de software com a modernidade. Desde a escolha da metáfora da engenharia para o desenvolvimento de software, um direcionamento tecnocêntrico

tem delimitado o espaço de problemas e de soluções da engenharia de software. Dentre muitos efeitos, surge uma vigorosa busca por modelos e padrões de validade "universal" e um posicionamento assimétrico na contabilização de sucesso e fracasso. O sucesso sempre é creditado à eficiência "interna", "técnica", dos modelos e padrões, enquanto o fracasso é visto como decorrência de causas exteriores, "sociais" ou "não-técnicas".

Como produto da *tecnociência* do século XX, a engenharia de software é apresentada como um artefato a-histórico, cuja submissão à moderna ideologia do progresso dificulta sua avaliação por parte daqueles empenhados em seu desenvolvimento, evolução e prática. "Desnaturalizar" seu tecnocentrismo contribui para que haja uma abertura para busca e aceitação de soluções diferentes daquelas às quais estamos acostumados. Assim como também pode contribuir para aliviar o peso em nossa consciência de engenheiros de software, cuja formação tradicionalmente nos leva a aceitar uma nítida divisão entre teoria e prática. Quando percebemos a dificuldade de adequarmos à prática a teoria que assimilamos — válida e correta, por princípio e direito —, muitas vezes duvidamos de nossa própria habilitação. Quanto a isso, Michael Mahoney (2002, p.37) traz um certo alívio ao afirmar que a engenharia de software "*constitui um exemplo primordial de como a tecnociência moderna confunde as categorias tradicionais de teoria e prática*"[ii].

O esforço de tentar abordar os aspectos ou fatores "não-técnicos" se justifica porque eles constituem potenciais gargalos a dificultar o alcance do fim último da engenharia de software (PFLEEGER, 2001, p.5): produzir sistemas de software com maior qualidade possível obedecendo às restrições — custos, prazos, tecnologias, recursos — existentes nos projetos reais. Em uma sociedade cada vez mais dependente do bom funcionamento dos onipresentes sistemas de software, e em um contexto nacional de busca manifesta de inserção do Brasil no rol dos produtores relevantes de software no mercado internacional[2], poderia constar em pauta prioritária a discussão do tema desta dissertação, cujos objetivos específicos são:

---

[2] Em 2004 o governo federal, por intermédio do Ministério de Ciência e Tecnologia, estipulou a meta de aumentar as exportações de software de US$120 milhões para US$2 bilhões em três anos. Uma notícia no site do MCT, com muito otimismo, registrava: "*o esforço que vem sendo realizado em várias regiões do País mostra que esse desafio imposto pelo governo brasileiro, há dois anos, pode se transformar em realidade, se não no prazo estipulado, mas bem mais cedo do que os menos otimistas poderiam esperar*" (disponível em <http://ftp.mct.gov.br/Temas/info/Imprensa/Noticias_4/ Software_4.htm>, acesso em: 01 maio 2007). Em 25/07/2006, o jornal Valor Econômico, em reportagem intitulada "*Exportações patinam e ficam longe da meta do governo*", contrapunha os dados do governo e do mercado. O governo admitia que o máximo que se atingiu, em 2005, com exportações de software foram US$400 milhões, quando o mercado admitia apenas US$250 milhões. De qualquer maneira, um valor muito abaixo da meta estipulada, que se mostrou inexeqüível. Esta reportagem pode ser obtida em: <http://www.brq.com.br/english/brqnamidia-20060725-valoreconomico-00.htm>, acesso em: 01 maio 2007).

- Evidenciar na construção da engenharia de software sua fundamentação na racionalidade científica ocidental. Além de eivado dos pressupostos de ordem, controle, estabilidade, representação, racionalismo, separação entre sujeito e objeto, formalização e do método como diretriz para a verdade, o enquadramento da engenharia de software valoriza a "universalização" dos métodos, modelos, processos e padrões que produz.
- Apontar, através de um caso empírico, a situação paradoxal da engenharia de software que, por um lado, enquadra seu espaço de problemas e de soluções dentro da moldura do paradigma modernista, porém, por outro lado, na prática, vê-se confrontada cada vez mais com o que transborda deste enquadramento.
- Apresentar o enfoque sociotécnico dos *Estudos CTS* como instrumental metodológico útil à ruptura do enquadramento limitante da engenharia de software, visto que refuta a divisão apriorística entre o "técnico" e o "não-técnico", o "dentro" e o "fora". Ao invés de reconhecer um enquadramento preestabelecido, com limites fixos e rígidos definidos *a priori*, o enfoque sociotécnico reconhece que os próprios limites da engenharia de software se estabelecem no caso a caso, coerentemente com o tratamento empírico dos problemas e soluções envolvidos. Com este enfoque metodológico, privilegia-se uma forma de compreensão dos problemas e soluções que se pode chamar de "*situada*" (SUCHMAN, 1987).

Portanto, esta dissertação está norteada pelo campo de pesquisas conhecido como *Estudos CTS (Ciência-Tecnologia-Sociedade)*, internacionalmente denominado de *Science and Technology Studies* (STS), ou simplesmente *Science Studies* (SS). Trata-se de um campo interdisciplinar, para a qual concorrem igualmente a filosofia, a antropologia, a economia, a história e a sociologia da ciência e da tecnologia. As instituições mundiais de ensino e pesquisa de primeira linha já constituíram departamentos e programas autônomos para este campo de pesquisa. Sua localização nessas instituições revela a "hesitação" típica das áreas interdisciplinares quanto ao seu enquadramento no interior de fronteiras já consagradas (e, portanto, mais conservadoras) entre as diversas áreas de conhecimento[3].

Vamos encontrar os *Estudos CTS* ora colocados próximos às ciências humanas, ora próximos às ciências exatas, especialmente às engenharias. A título de exemplo,

[3] Este trecho e os dois parágrafos seguintes foram extraídos do *site* do grupo de pesquisas NECSO, cadastrado no CNPQ, disponível em <http://dgp.cnpq.br/buscaoperacional/>.

em Paris a área se concentra no *Centre de Sociologie de L'Innovation* (CSI), por sua vez abrigado no interior de uma das mais tradicionais escolas de engenharia da França, a *École des Mines*. Já do outro lado do Canal da Mancha, mais especificamente em Edimburgo, a área se situa dentro da Escola de Sociologia. Nos EUA, os *Estudos CTS* são desenvolvidos tanto no MIT quanto em *Stanford* vinculados à área de humanidades.

Independente do nome e da afiliação institucional, todas as universidades estrangeiras de renome trataram de fundar sua área de *Science and Technology Studies* porque compreenderam que não é mais possível conceber a tecnociência alcançando a fronteira do conhecimento, ou as áreas consideradas estratégicas para o desenvolvimento científico e tecnológico, se movida exclusivamente por uma mentalidade tecnicista. A tecnociência do século XXI se apresenta através de um conjunto tão complexamente relacionado de questões científicas, técnicas, éticas, sociais, políticas, econômicas e ecológicas que não se pode pensá-la em termos estritamente "técnicos". Não se tratam apenas de implicações e impactos advindos das atividades tecnocientíficas, como se elas estivessem "acima" ou fossem "anteriores" à cultura e à sociedade: em verdade, ciência e tecnologia são construídas no mesmo movimento que constrói a cultura, a sociedade e a própria natureza.

Examinar a engenharia de software à luz dos *Estudos CTS* possibilita questionar sua aura de "pureza técnica", questionar a suposta "naturalidade" do desenvolvimento de software ser simplesmente uma disciplina da "engenharia". O que se reconhece como engenharia de software é o efeito de uma extensa rede sociotécnica que socializa elementos diversos. Os próprios projetos de software, sejam eles projetos de desenvolvimento, manutenção ou de implantação de processos de software, poderão ser melhor compreendidos e sucedidos se encarados como esforços de estabilização de redes sociotécnicas e não como esforços de *difusão* de uma tecnologia — de um "conteúdo" — que requer apenas implantação em um determinado "contexto". Contexto e conteúdo estão sempre imbricados, cabendo ao engenheiro de software o esforço de arregimentação e socialização de coisas e pessoas em uma rede sociotécnica cujo efeito, conforme o caso, poderá ser visto como um software bem sucedido ou um processo de software implantado com sucesso.

Os *Estudos CTS* disponibilizam para o observador/pesquisador um instrumental que trata com equivalência analítica humanos e não-humanos. Os fatos e artefatos (científicos e tecnológicos) existem e se propagam em suas redes sociotécnicas, sempre com lastro na materialidade e na heterogeneidade de sua composição. Trata-se de um suporte metodológico que valoriza a explicitação das relações entre os diversos atores envolvidos nas redes sociotécnicas e que refuta as muitas divisões

inerentemente convencionais quando estabelecidas *a priori*, tais como contexto x conteúdo, sujeito x objeto, técnico x social.

No caso empírico investigado, o da informática no Banco Nacional de Desenvolvimento Econômico e Social (BNDES), buscou-se avaliar a prática de desenvolvimento de software ocorrida em três momentos: (i) de meados dos anos 1970 até meados dos anos 1980; (ii) de meados dos anos 1980 até o final dos anos 1990; e (iii) a partir dos anos 2000.

Adicionalmente ao instrumental metodológico dos *Estudos CTS*, foram utilizadas entrevistas semi-estruturadas (QUIVY; CAMPENHOUDT, 1998). As entrevistas foram realizadas entre os meses de junho e agosto de 2006, no próprio BNDES. O tema proposto aos entrevistados foi "*a informática do BNDES sob um ponto de vista das metodologias e processos de software*". Ao todo foram realizadas dezesseis entrevistas com duração de sessenta a noventa minutos (apenas uma não está registrada em áudio). Após a apresentação do tema e de uma pergunta de indução feita pelo entrevistador, era permitido ao entrevistado falar livremente. Novas perguntas eram feitas na medida do necessário, evitando-se interromper o entrevistado. Todos os dezesseis entrevistados eram funcionários do BNDES, sendo que quatro deles estão na iminência de aposentadoria. Três deles nunca trabalharam diretamente na área de informática do BNDES, e outros nove sempre trabalharam na área de informática desde que ingressaram no BNDES. Quatro ingressaram no BNDES na década de 1970, oito ingressaram na década de 1980 e os quatro restantes ingressaram na década de 1990. Soma-se, também, o uso de observação direta e participação ativa no terceiro momento considerado, pois o autor integrou, até o início de 2007, a equipe de um projeto que desde 2003 vem implantando um processo de software baseado no *Capability Maturity Model Integration* (CMMI) no BNDES.

As fontes documentais internas ao BNDES consistem basicamente de resoluções e instruções da alta administração, relatórios de consultorias, planos diretores de informática (PDI), planos de projetos, periódicos de circulação interna, relatórios de atividades e manuais de procedimentos e sistemas.

Além deste capítulo introdutório, a dissertação está estruturada em mais cinco capítulos. A seguir, no capítulo 2, buscou-se descrever o cenário inicial de produção de software nos anos 1950 e 1960, nos EUA, para dar conta de parte da construção histórica da engenharia de software. É abordada a influência militar em sua fundação, o que em nada surpreende dada a presença militar em todo desenvolvimento tecnocientífico do século XX. Quando surgiu como ator, o software encontrava-se em um cenário favorável ao entendimento segundo o qual desenvolver software é uma

questão de engenharia. A idéia da construção sociotécnica da engenharia de software será examinada a partir do uso do termo "engenharia" como uma metáfora, explorando os conceitos dos *Estudos CTS* e a história da engenharia de software.

No capítulo 3 será abordado o *estilo de pensamento* hegemônico na engenharia de software. Após um breve resumo sobre o apelo exercido pelo pensamento moderno na racionalidade científica ocidental, são explicitados diversos vínculos da engenharia de software com o modernismo, começando por sua busca de respaldo científico. A engenharia de software devota um significativo esforço em desenvolver métodos, modelos, processos e padrões cuja aplicação seja "universal". Prevalece a idéia de que o domínio "técnico" de tais métodos, modelos, processos viabiliza sua difusão e utilização e que, uma vez implantados, são capazes de materializar os benefícios que supostamente lhe são intrínsecos. Uma implantação bem sucedida de um método ou modelo "universal" (por exemplo, a *análise estruturada* ou o *CMMI*[4]) deveria seus bons resultados à presumida excelência técnica que detém, à suposta eficiência que lhe é intrínseca. Se porventura o projeto de implantação fracassar, ou os resultados esperados não se materializarem, as causas do insucesso são associadas a questões sociais, culturais, políticas, organizacionais, econômicas, nunca recaindo sobre o próprio modelo e sua presumida eficácia técnica. Produz-se assim uma explicação assimétrica: se o projeto de implantação dá certo, o sucesso é creditado à conta da eficiência técnica "interna" ao modelo, mas, se dá errado, o fracasso tem de ser explicado por causas que lhe são "externas"[5]. No encerramento do capítulo, será destacado que metáforas alternativas à engenharia poderiam ter conformado de modo diferente a prática do desenvolvimento de software que hoje conhecemos. Além disso, sua visão mecanicista subjacente dá cada vez menos conta da complexidade e dinamicidade do mundo que se apresenta ao engenheiro de software, pois também cada vez menos se consegue replicar as premissas demandadas por essa visão nas redes sociotécnicas dos projetos.

Saindo da discussão mais geral, no capítulo 4 será abordado um exemplo brasileiro, através de um relato sobre o processo de estruturação da informática do BNDES nos anos 1970. Após discorrer sobre aquele cenário — a estrutura organizacional, o quadro de profissionais, os equipamentos e sistemas da época, a

---

[4] Ambos são modelos para estruturar e permitir o controle do esforço de desenvolvimento de software. A *análise estruturada* foi muito popular nos anos 1980 e início dos anos 1990, quando começou a perder espaço para as abordagens de orientação por objetos. Já o CMMI — *Capability Maturity Model Integration* — é um agregado de modelos para o desenvolvimento e melhoria de processos, desenvolvido pelo *Software Engineering Institute* (SEI), Universidade *Carnegie Mellon*, sob o patrocínio do Departamento de Defesa norte-americano (DoD) (CHRISSIS et al., 2003).

[5] Sobre a assimetria na análise do sucesso e do fracasso do empreendimento científico e tecnológico, veja Bloor (1991).

prática de desenvolvimento de software que ocorreu —, explora-se a noção de rede sociotécnica como alternativa para o entendimento da ordem, padronização e produtividade que se reconhece ter existido. Ao término do capítulo 4, serão apontados paralelos entre o cenário visto no BNDES dos anos 1970 e, guardadas as proporções, o cenário inicial de produção da engenharia de software nos projetos militares norte-americanos, visto no capítulo 2. Destaca-se que em ambos os cenários pôde haver um enredamento, nas respectivas redes sociotécnicas, de diversas premissas necessárias à aplicação bem sucedida de uma engenharia de software imbuída de uma visão mecanicista de mundo.

No capítulo 5, ao se abordar dois outros momentos da informática do BNDES, aparentemente abandona-se a engenharia de software. O primeiro momento ocorreu no final dos anos 1980, com a tentativa de implantação de uma metodologia baseada na *análise estruturada*. O segundo, no início dos anos 2000, diz respeito à implantação de um processo de software baseado no CMMI. Serão apresentadas duas histórias para a reflexão sobre aquilo que transborda do enquadramento usual da engenharia de software. São dois exemplos empíricos onde modelos e abordagens, apesar de sua reivindicação de "universalidade", não conseguem, *per si*, apenas com sua suposta "pureza técnica", consolidar na prática situada do BNDES os benefícios que advogam. Assim como a teoria e a prática da engenharia de software parecem distantes (MAHONEY, 2002, p.37, PARNAS[6], 1998 apud SEELY, 2002, p.91), as duas histórias apresentadas parecem distantes da narrativa usual da engenharia de software. Deste modo, no capítulo 5 chama-se a atenção para a importância do estudo de nossas práticas locais, situadas, de engenharia de software para destacar empiricamente as complexas redes de relações, técnicas, sociais, políticas, organizacionais sempre presentes e contrapô-las à difusão dos pretensos modelos "universais", em busca de um confronto mais rico e apropriado às nossas realidades e problemas.

Por fim, no capítulo 6, sintetiza-se algumas conclusões.

Dentro do recorte parcial e incompleto que foi possível traçar, foram encontrados alguns elementos para a reflexão acerca da engenharia de software, uma disciplina da modernidade. Cabe reconhecer, no caso observado do BNDES, que a interpretação de "sucesso" ou "fracasso" não poderia ser explicada apenas pela informática. Muito menos pela explicação simplista, à qual esta dissertação vai contra, de que o sucesso ou fracasso deve ser reputado à adoção apropriada ou equivocada de um modelo "universal"; ou ainda, pela adequação ou inadequação do modelo escolhido ao

"contexto" de sua utilização. Um estudo mais rigoroso e completo do sucesso/fracasso dos casos apresentados não poderia focalizar apenas a informática, como foi feito, deixando de lado tantos outros nós e laços da rede sociotécnica.

Finalmente, cabe mais um destaque, desta vez acerca da unicidade usualmente conferida ao título *engenharia de software*, algo melhor compreendido caso se reconhecesse sua multiplicidade — conceito adequadamente tratado com a noção de rede sociotécnica —, como provoca Endres.

> Não coloque todo software junto e o trate como fenômeno único. Software para o ônibus espacial, um editor de texto ou um primeiro experimento JAVA de um estudante são coisas muito distintas. Fazer isso, seria o mesmo que tratar um manual de manutenção de um avião, um romance *best-seller* e pichações no subterrâneo como trabalhos de escrita semelhantes. Freqüentemente, pessoas que têm escrito um tipo de programa declaram-se expertos em todos os tipos. Você não aceita isso para nenhum outro tipo de engenharia ou trabalho literário. Tampouco isso tornaria o ensino da engenharia de software mais fácil.[iii] (ENDRES, 2002, p.81).

Barry Boehm (2006) questiona qual é a engenharia de software referida quando se fala "*a engenharia de software*": é a que trata de pequenos ou grandes sistemas de software? A que considera os softwares do tipo *commodities* ou personalizados? Aquela para software embarcado ou de uso direto? Aquela outra em situação de "terra virgem"[7] ou dirigida ao reuso/legado/COTS[8]? A praticada internamente na empresa, a terceirizada ou ambas? Aquela para software de uso casual ou para software de missão crítica? Nesta dissertação, a engenharia de software em foco é aquela praticada nas empresas cuja produção de software não é sua atividade fim, aquelas que desenvolvem software para apoiar seus processos de negócio. Mas, é bom que se diga, também não se pretende admitir generalizações além do caso em estudo, o do BNDES, valendo aqui o dito popular: "cada caso é um caso".

---

[6] PARNAS, D. L., 1998, "Successful Software Engineering Research ". In: Software Engineering Notes, v. 23, n. 3, p.64-68.

[7] "Terra virgem", ou "campo verde", lugar-comum não só no jargão de informática, mas em diversos ramos de negócio, que serve para figurar uma área, ou situação, ainda não cultivada, tratada, construída — *greenfield land* —, em contraste a uma outra situação — *brownfield land* — onde algo já foi cultivado, tratado, construído, podendo até conter os escombros do que ali teria existido.

[8] COTS — *Commercial Of The Shelf*; são os produtos de software "de prateleira".

Capítulo 2
A Engenharia no Desenvolvimento de Software

A engenharia de software é uma construção muito recente, pois o próprio software passou a ser reconhecido como artefato há aproximadamente meio século. Neste capítulo será abordado um pouco da história da engenharia de software, com foco nos EUA, país que capitaneou seu desenvolvimento. O conceito de software não surgiu imediatamente após o aparecimento dos primeiros computadores eletromecânicos e a válvulas, na década de 1940. Naqueles computadores, como ainda não havia o conceito de instruções armazenadas na memória, a programação era algo muito próximo do hardware, normalmente sendo realizada pelas mesmas pessoas que projetavam, construíam e operavam o computador. A programação consistia em ligar/interromper a conexão de fios ou ligar/desligar chaves em um painel do computador. Era um cenário em que não existia o entendimento da programação como hoje, como algo separado do hardware. Por isso, até mesmo o surgimento dos cartões perfurados para a programação de computadores representou um expressivo avanço na área.

A utilização do termo software, segundo Michael Mahoney (2002, p.43), surgiu somente em 1958 com John Wilder Tukey[9], que o entendia como antônimo de hardware, para significar os programas e procedimentos que habilitam a realização de uma tarefa por um computador. Já o termo "produto de software" esperou quase mais uma década, até 1966, para ingressar no léxico da computação. Em 1966 teve início a publicação de um catálogo, chamado *International Computers Programs Quartely*, onde desenvolvedores de software podiam anunciar seus produtos (CAMPBELL-KELLY, 2004, p.99). Até aquele instante, não era usual a venda de artefatos de software. Para os fabricantes de computadores, software era encarado como despesa de *marketing*, sendo necessários para a venda do hardware.

2.1 Produção bem sucedida em larga escala para uso militar

Um avanço brutal no desenvolvimento de software em particular, e nas tecnologias de computação como um todo, decorreu dos bilionários projetos militares de defesa, empreendidos pelo governo norte americano nos anos 1950 e 1960. Esses projetos

[9] TUKEY, J., 1958, "The Teaching of Concrete Mathematics.", *American Mathematical Monthly*, v.65, n.1, p.1-9.

criaram o mercado para os primeiros fornecedores de software, na época conhecidos como firmas de serviços de programação (*custom programming firms*) (CAMPBELL-KELLY, 2004, p.5). A própria engenharia de software, como um produto da tecnociência do século XX, faz parte, como diz Latour (2000, p.282), de "*uma máquina de guerra*". Na voz de Edwards,

> [...] A RAND Corporation, [...] onde cientistas [das ciências] naturais, sociais e matemáticos trabalhavam lado a lado para antecipar e preparar o futuro da guerra [fria], usaram técnicas de análise de sistemas, originada da pesquisa operacional no período de guerra, para investigar, tanto os problemas mundanos como compra de armamento, quanto o desconhecido reino da estratégia nuclear. Simultaneamente, essas técnicas foram beneficiadas, por, e beneficiaram a RAND, em, seu extensivo trabalho com computadores; entre os maiores do mundo, o centro de computação da RAND afetou significativamente a natureza e a direção do desenvolvimento de software nos anos 1950. Além de legitimar a análise de sistemas, os computadores ajudaram no avanço da abordagem da RAND para a estratégia, baseada em teoria e simulação.[iv] (EDWARDS, 1997, p.113).

Sob a ameaça de bombardeiros soviéticos equipados com artefatos nucleares, vários projetos para desenvolver os chamados sistemas de *comando e controle*, conhecidos como os "BIG-L", foram executados pelos EUA. Um precursor, e padrão, para vários outros foi o projeto 416L da Força Aérea que criou o sistema SAGE — *Semi-Automatic Ground Environment*, cujo protótipo começou a ser desenvolvido no MIT — *Massachusetts Institute of Technology* — em 1950 e cuja construção de fato começou em 1954. Mais alguns "BIG-L's" se seguiram, entre eles o projeto 412L, NORAD — *North Atlantic Air Defense*; 438L — *Air Force Intelligence Data Handling System*; o 474L, BMEWS — *Ballistic Early Warning System* (ibid., p.107). Com orçamentos bilionários, os projetos de defesa concretizaram empreendimentos de altíssima complexidade que, a reboque, trouxeram um sem número de inovações tecnológicas, tanto em hardware quanto em software, nas décadas de 1950 e 1960, que acabaram chegando à comunidade civil através de computadores mais poderosos e linguagens de programação de alto nível. Considerando apenas o SAGE, podem ser observados como seus méritos a consecução de inovações tecnológicas tão importantes para a computação como a memória de núcleo magnético (*magnetic core memory*); os terminais de vídeo; as canetas óticas; várias técnicas de computação gráfica (*graphic display techniques*); a primeira linguagem algébrica, efetiva, de computador; várias técnicas de simulação; a lógica de sincronização paralela (transmissão paralela de bits, ao invés de serial); técnicas para conversão analógico-digital, e vice-versa; a transmissão de dados digitais por linhas telefônicas; o multiprocessamento; a interligação de computadores em redes; além de idéias básicas para o desenvolvimento dos conceitos de compiladores e interpretadores (ibid., p.99-102).

A tônica dos "mega projetos BIG'L" era o desenvolvimento e integração de uma miríade de novos equipamentos eletrônicos. Os fornecedores envolvidos eram grandes empresas de engenharia elétrica e eletrônica, fabricantes de computadores, companhias aerospaciais e corporações governamentais, como a *RAND Corporation*. Via de regra, uma grande empresa integradora de sistemas vencia uma licitação, tornava-se a contratada principal (*prime contractor*) e subcontratava diversas outras empresas. No caso do SAGE, a contratada principal foi a Western Eletric, que subcontratou IBM, RCA, Bendix, General Eletric, Bell Labs e Burroughs para, dentre outras questões, desenvolver radares, computadores, comunicações e análise técnica (CAMPBELL-KELLY, 2004, p.37). Naquela época, era comum apenas a menor parte dos esforços e recursos ser contabilizada como programação dos sistemas:

> Uma estimativa comum para o [custo] total de desenvolvimento [do SAGE] é US$8 bilhões, incluindo a construção de vinte e seis centros de defesa, a compra de cinqüenta e seis computadores por US$30 milhões cada, a instalação de vinte e cinco mil linhas telefônicas, além dos outros equipamentos e serviços associados. *A programação* das oito versões do SAGE, feita pela SDC [como veremos, empresa originária de um desmembramento da RAND Corporation], *chegaria à cifra de US$150 milhões, em torno de dois por cento do custo total do sistema* [SAGE].[v] (BAUM, 1981, p.12-13, grifos nossos).

Este era o contexto no qual surgiu uma enorme demanda por programação — desenvolvimento de software como se diria atualmente — em larga escala, tutelada por uma forte cultura em gerenciamento de projetos das empresas de engenharia e construção.

As empresas contratadas para os serviços de programação acabaram adotando o mesmo modelo de operação das firmas de engenharia e construção para conseguirem se estabelecer no cenário de desenvolvimento de software em larga escala, demandados pelos projetos de defesa. Elas disputavam licitações e, quando venciam, firmavam contratos com base em preço fixo, às vezes com base em tempo e material[10]. A margem de lucro no mercado em que atuavam, diferente do que pode parecer, era muito baixa — entre 8% e 15%. Na maioria das vezes o custo e o prazo de execução eram estabelecidos no início da empreitada, logo, o gerenciamento efetivo de projetos e a alta precisão na estimativa de custos eram habilidades críticas para a sobrevivência dessas empresas. Elas optavam por operar em domínios

[10] Contratos por tempo e material (*time-and-material base*) apresentavam um menor risco para as firmas contratadas, porém a performance delas era crucial para continuarem conseguindo outros contratos. Nos contratos com base em tempo e material, o valor total não é definido no momento em que é firmado, podendo aumentar no decorrer da execução do projeto à medida que for necessário empregar mais esforços e/ou disponibilizar mais produtos, cujos valores unitários normalmente foram pactuados na contratação.

particulares, desenvolvendo um alto grau de especialização para atingirem a competência necessária ao seu sucesso. (CAMPBELL-KELLY, 2004, p.67).

Além das inovações em hardware e software, as técnicas para guiar o desenvolvimento de software que surgiram nos projetos de defesa também chegaram à comunidade civil. Um destaque é a disciplina de gerenciamento de projetos de desenvolvimento de software, ou de projetos de programação como eram chamados, cuja origem também remonta ao SAGE. Técnicas de gerenciamento já maduras na engenharia foram adaptadas em uma seqüência de estágios que partia do estabelecimento inicial dos conceitos e seguia até o artefato final produzido pela atividade de programação. Numa conferência seminal no *MIT Lincoln Laboratory* em 1956, John F. Jacobs, responsável por uma linha de programas no SAGE que envolvia entre 300 e 400 programadores, entre 1955 e 1956, apresentou, conforme ilustrado na Figura 1, a seguinte estruturação seqüencial para as tarefas de um projeto de programação:

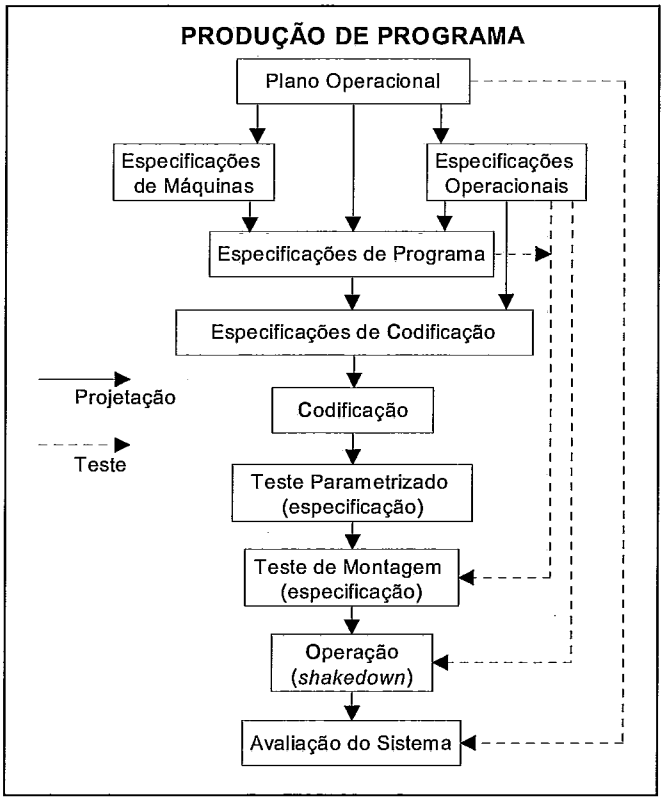


Figura 1 — Gerenciamento de Projetos no SAGE
(JACOBS[11], 1956 apud CAMPBELL-KELLY, 2004, p.68)

A especificação dos requisitos do sistema no estágio (i) de *especificações operacionais e de máquinas* era, em seguida, *congelada* para se poder avançar para o

estágio (ii) de *especificações de programa*. Nesse, os programadores seniores dividiam o programa em subprogramas que se intercomunicavam. Em seguida, no estágio (iii) de *especificação de código*, programadores especificavam a lógica de cada subprograma para a realização do estágio (iv) *de codificação*, onde os codificadores, considerados os programadores de menor qualificação, convertiam as especificações em código de máquina comentado, documentado com fluxogramas e pronto para a perfuração de cartões. Depois vinham os estágios de testes e início de utilização. Esse esquema hierárquico, guiado por especificações, foi largamente difundido na indústria de software no final dos anos 1950, por milhares de programadores que o vivenciaram no projeto SAGE (CAMPBELL-KELLY, 2004, p.67-69).

De fato, outro efeito dos projetos de defesa foi o de ostensivamente formar uma geração de profissionais, posteriormente absorvidos pelo mercado. Para a construção do SAGE, a *RAND Corporation* viu que não existiam programadores suficientes com a *expertise* necessária, ou seja, teriam que ser disponibilizados meios para sua formação em quantidade e competência suficientes. Em 1956, para cumprir essa tarefa, a RAND então criou a *System Development Corporation* (SDC) a partir do desmembramento de sua Divisão de Desenvolvimento de Sistemas. No início dos anos 1960, o mercado civil que passara a existir absorveu milhares desses programadores extremamente capacitados, com experiência no desenvolvimento de software militar de missão crítica, egressos do projeto SAGE (ibid., p.38-40). A SDC não foi apenas responsável por treinar programadores numa escala maior que qualquer outra instituição na época, mas foi ela que induziu, na indústria, a abordagem de desenvolvimento de software orientada a sistemas.

> "A RAND [Corporation] está muito ocupada para encarregar-se dos problemas da indústria. Ademais, seu estatuto o proibiria. Mas ela oferece sua promissora capacidade de pensamento. Homens treinados na Divisão de Desenvolvimento de Sistemas se tornarão uma fonte de suprimentos para diretores dos futuros projetos de automação..." Essas palavras apareceram na *Business Week* por volta de dezoito meses antes da SDC começar suas operações, profetizaram precisamente o papel que a companhia iria desempenhar ao treinar milhares de programadores, analistas de sistemas e peritos para treinamento em sistemas que em breve ocupariam a base da emergente industria de processamento de dados. Alguns veteranos da SDC questionados sobre esse assunto rapidamente recordam: "Nós treinamos os primeiros três mil programadores deste país". "Não existiam programadores de sistemas antes da SDC". "Qualquer companhia que visito, encontro dois ou três ex-alunos da SDC". "Nós treinamos a indústria!".[vi] (BAUM, 1981, p.47).

Um clássico exemplo do aproveitamento civil dos avanços gerados pelos projetos de defesa é o projeto SABRE da IBM. Tratava-se de um sistema de reservas

---

[11] JACOBS, J.F., 1956, *The SAGE Air Defense System.*

de passagens aéreas que a IBM desenvolveu para a *American Airlines* e que, na época, resultou em significativa vantagem competitiva para essa companhia aérea. Construído entre 1959 e 1965, o SABRE (não se trata de um acrônimo) foi apelidado de "filho do SAGE" porque, além de sua complexidade e de utilizar tecnologias de processamento em tempo real, originou-se de um projeto interno da IBM denominado SABER — *Semi-Automatic Business Environment Research*, que indiretamente referenciava o SAGE — *Semi-Automatic Ground Environment*. O SABRE começou a ser planejado em 1953, mais de dez anos antes de sua conclusão. Era um projeto economicamente inviável naquela ocasião, pois o desafio que representava e a quantidade de equipamentos que demandava eram compatíveis com o desenvolvimento de um centro de controle (*direction center*, ou centro de defesa) do SAGE, orçado em US$300 milhões (da época). Para termos uma idéia, os citados computadores NA/FSQ7, que a própria IBM fabricou para o SAGE em meados dos anos 1950 ao preço de US$30 milhões cada um, pesavam 250 toneladas, continham 49.000 válvulas e consumiam 3 megawatts de energia elétrica. No entanto, em 1959 o estágio da tecnologia já era tal que o custo de desenvolvimento estimado para o SABRE já havia caído para um décimo, US$30 milhões, o que viabilizou sua construção (CAMPBELL-KELLY, 2004, p.38-45).

2.2 Software, um novo ator em cena

Os primeiros computadores para uso civil já estavam disponíveis desde o início dos anos 1950. Contudo, como essas máquinas eram vendidas praticamente sem nenhum software, as empresas as adquiriam e eram confrontadas com a desafiante missão de torná-las operacionais. O modelo IBM-701, por exemplo, era acompanhado basicamente de um manual — Princípios de Operação — de 103 páginas, um *assembler* primitivo, um programa para carregar um cartão na memória e outro para limpar a memória. Curiosamente, naquela época os fabricantes dispunham de um quadro de profissionais dedicados à produção de software apenas ligeiramente maior do que as empresas usuárias de computador (ibid., p.29-30).

Em 1953, quando entrou no mercado de computadores, a IBM inaugurou seu *Technical Computing Bureau* e começou a treinar os futuros programadores das empresas usuárias seis meses antes das máquinas IBM-701 estarem prontas para entrega. Já no treinamento, alguns programadores desenvolviam aplicativos que favoreceriam o início mais rápido da utilização dos IBM-701, assim que eles fossem entregues às suas empresas. Portanto, rotinas extremamente básicas para a

operação daqueles primeiros computadores eram desenvolvidas em conjunto pelos programadores das empresas clientes e pela equipe do *Technical Computing Bureau.* Nos anos 1950, o custo de programação e operacionalização do computador era tão elevado que as empresas usuárias tinham dificuldade de justificar o custo total de computação[12]. Em decorrência, a IBM fomentou a criação de uma associação cooperativa das empresas usuárias, na qual também participava, para desenvolver programas de utilização comum. Em 1954 quando foi lançado o modelo IBM-704, seus usuários puderam se beneficiar, "sem custo algum", de toda a biblioteca já desenvolvida pela IBM e pelo grupo de usuários associados. Todos os fabricantes passaram a fomentar associações desse tipo. Mesmo assim, a maioria dos programas necessários ao controle e operacionalização das máquinas eram desenvolvidos nas próprias instalações que, por isso, demandavam um expressivo número de programadores altamente qualificados. Somente a partir de 1965 começou a haver uma mudança nesse cenário em que quase todo software de uma instalação de computador era desenvolvido exclusivamente para ela, seja por seu pessoal interno ou pelas empresas de serviços de programação que já existiam[13] (CAMPBELL-KELLY, 2004, p.31-33, 89).

Em meados dos anos 1960, o cenário da computação tornou-se bastante diferente, posto que havia uma pressão muito mais disseminada por desenvolvimento de software, com um número cada vez maior de instalações de computadores. Em 1955, existiam nos EUA apenas 240 computadores, já em 1960 esse número subiu para 4.400, alcançando, em 1965, a conta de 21.600. Nos anos 1970, a quantidade de computadores nos EUA já era de 41.500. Além disso, o poder de processamento e armazenagem de dados das máquinas também havia crescido vertiginosamente. Tomemos como exemplo o IBM-650 e o IBM-360/30, as máquinas mais populares da IBM nos anos 1955 e 1965, respectivamente. O IBM-360/30 tinha uma memória 66 vezes maior e uma velocidade de processamento 43 vezes maior que o IBM-650. Com isso, o custo por instrução de computador em 1965 era 40 vezes menor que em 1955. No entanto, por razões de produção e de *marketing*, que fogem do escopo desta dissertação, não era possível produzir computadores em 1965 custando um

[12] Por exemplo, a General Eletric adquiriu um UNIVAC em janeiro de 1954 e levou dois anos para tornar operacional um conjunto básico de aplicações de contabilidade, tendo, para isso, demitido uma equipe inteira de programadores no período (CAMPBELL-KELLY, 2004, p.30).

[13] Contribuiu também para essa mudança de cenário, o aparecimento da linha de computadores IBM/360, que trazia o conceito de família de computadores, com arquiteturas compatíveis. Isso favorecia a utilização do software de uma maneira menos exclusivista, permitindo maior reutilização, pois o software desenvolvido para uma máquina podia ser executado em outra com arquitetura compatível. Até o início dos anos 1960, a própria IBM tinha nada menos do que 7 tipos de máquinas com arquiteturas incompatíveis entre si. (CAMPBELL-KELLY, 2004, p.95).

quarenta avos do preço que tinham em 1955. Na verdade, o preço absoluto dos computadores estava duas vezes maior em 1965. O aumento da capacidade dos computadores pressionou pelo aumento do poder do software. Uma evidência indireta foi o crescimento da quantidade de software disponibilizado junto com o computador. Em 1955, o IBM-650 trazia aproximadamente 10.000 linhas de código. O IBM-1401, em 1960, vinha com 100.000 linhas de código. Já o IBM-360/30, em 1965, trazia 1.000.000 de linhas de código. Portanto, no caso da IBM, a maior fabricante, há cada 5 anos a quantidade de código fornecida com seus computadores crescia dez vezes. (CAMPBELL-KELLY, 2004, p.89-91).

A existência de máquinas muito mais poderosas, em maior número e com preços ainda mais elevados trazia um novo desafio à produção de software. Artefatos de software para atender às mais diversas finalidades passaram a ser muito mais demandados: (i) porque a taxa de utilização dos computadores — agora ainda mais potentes — tinha que ser maximizada, em função dos altos custos das instalações; e (ii) porque inúmeras outras áreas de atividade puderam ser abrangidas pela computação eletrônica, uma vez que o preço relativo da computação havia caído bastante. Os custos relativos entre hardware e software foram deslocados e o custo do software passou a ser mais significativo. Essa situação, ostensivamente, trouxe à tona os problemas relacionados aos projetos de programação.

Na década anterior, o valor contabilizado para a programação era praticamente marginal. Até mesmo quando os projetos gastavam vultosas quantias no desenvolvimento de software, quando vistos sob uma perspectiva de custo total, essa despesa "perdia a importância", como o citado exemplo do SAGE que embora tenha custado US$150 milhões à SDC para programá-lo, no custo total isso foi contabilizado como correspondente a apenas 2%[14]. Até os anos 1960, além dos programas mais básicos para a utilização dos computadores, os fabricantes disponibilizavam "sem custo" para os usuários até mesmo aplicativos para suportar atividades dos maiores

[14] É importante destacar que nos anos 1950 as categorias software e hardware, ou mesmo os limites entre software e hardware, não tinham a conotação atual. Por isso, a informação de que a programação do SAGE custou 2% de seu valor total pode demandar considerações mais profundas. Mesmo assim, é largamente reconhecido que o custo relativo do software era muitíssimo menor que o do hardware nos momentos iniciais da computação eletrônica, relação que se inverteu totalmente com o passar do tempo. Claude Baum (1981, p.8) diz que nos anos 1980 o software custava um pouco menos do que custava quando fizeram o SAGE, embora o custo do hardware houvesse despencado mais de 1000 vezes. Já Barry Boehm (2006) lembra que acertou a previsão que fizera em 1973, sobre a tendência de inversão dos custos de hardware/software em grandes organizações, antecipando que o software representaria 85% do custo total dos sistemas de computação em 1985, partindo de meros 16% em 1955. Parece intuitivo que o custo do hardware do SAGE, na maior parte exclusivamente desenvolvido para ele, carregando alto grau de inovação tecnológica, fosse sobremaneira superior ao custo de seu software, que, uma vez desenvolvido, foi implantado nos 56 computadores NA/FSQ7 dos centros de defesa. Ao custo de U$ 30 milhões cada um, só em computadores o projeto SAGE gastou US$1,68 bilhão (da época).

ramos de negócio, como o setor bancário, de seguro, industrial e de varejo. O custo do desenvolvimento desses aplicativos era considerado despesa de *marketing* pelos fabricantes, uma condição necessária para a venda de hardware, este sim, tido como seu verdadeiro negócio à época. (CAMPBELL-KELLY, 2004, p.91-98).

Em 1967, a IBM, a maior fabricante de computadores da época, começou a ser investigada pela Divisão Antitruste do Departamento de Justiça dos EUA por dominar 70% do mercado americano e por sua conduta comercial de não abrir o plano de preços dos diversos produtos que oferecia agregados — *bundled* — aos seus clientes. Argumentava-se que, a fim de vencer a concorrência, a IBM poderia oferecer software e outros serviços abaixo do custo, o que seria violação das leis antitruste. Em dezembro de 1968, a IBM anunciou sua intenção de separar — *unbundle* — seus preços e cobrar distintamente por serviços de engenharia, serviços de educação e treinamento, serviços de programação, e pacotes de software. A implementação da decisão de *unbundling* da IBM, em janeiro de 1970, é vista como um "ponto de inflexão" para a indústria de software. (ibid., p.109-110).

Daí em diante, o cenário de desenvolvimento de software tornou-se muito diferente do cenário inicial da computação eletrônica em que, por um lado, nos projetos civis, os poucos e limitados computadores não impunham demanda comparativamente expressiva por software, além de ser comum sua gratuidade. Por outro lado, nos projetos de defesa havia a cultura de engenharia, com projetos hierarquicamente estruturados em um modelo baseado em licitações, além da ampla possibilidade de formação de programadores qualificados em função dos bilionários orçamentos disponíveis a esses projetos. A partir de meados dos anos 1960, surgia a demanda por grande quantidade de software para utilização nas mais diversas atividades civis, com diferentes níveis de complexidade. Com o avanço tecnológico, tratavam-se de projetos com um custo total bem menor, embora com custo de software relativamente muito maior. E no custo do software, o determinante passou a ser a mão-de-obra.

Alertas enfáticos sobre a preocupante falta de programadores são tão antigos quanto a própria existência de computadores eletrônicos, e até hoje surgem com outras roupagens[15]. Por exemplo, em 1954 uma conferência na Universidade Wayne,

[15] Edwards (1997, p.247-249) traça uma relação entre a falta de programadores e o desenvolvimento das linguagens de programação de alto nível. Os primeiros programadores eram basicamente os matemáticos e engenheiros que projetavam e construíam os computadores. Para eles, prevalecia a estética matemática de concisão como norma de elegância, ainda mais sob a injunção dos escassos recursos das máquinas. Surge uma cultura de valorização de algoritmos pequenos, eficientes e altamente baseados na matemática. Para o projeto SAGE, a SDC precisou formar milhares de programadores, com destaque para o bom desempenho apresentado por mulheres e professore(a)s de música. Muitos não tinham o sólido embasamento matemático necessário à programação em linguagem de máquina e produção de algoritmos

Michigan, intitulada *The First Conference on Training Personnel for the Computing Machine Field*, abordava a questão (HOHN, 1955). Já em 2006, outra conferência, desta vez em Hydebaraad na Índia, alertava sobre a falta de meio milhão de profissionais de TI até 2010 (FAROOQ, 2006).

Não obstante, a partir de meados dos anos 1960, com o forte crescimento do mercado comercial de computadores e com a cultura da época em que expressiva quantidade de software era desenvolvido exclusivamente para cada instalação, de fato a demanda por programadores cresceu bastante rapidamente. A preocupação com a falta de profissionais, que até então sempre tinha um caráter mais imediato, passou a refletir uma preocupação com a própria viabilidade futura da indústria da computação. Segundo Ensmenger e Aspray (2002, p.142), uma pesquisa de 1967 da AFIPS[16], amplamente citada, estimava que embora existissem 100.000 programadores nos EUA, pelo menos mais 50.000 eram imediatamente necessários.

> A competição por programadores elevou os salários tão rapidamente que a programação tem provavelmente se tornado a mais alta remuneração de um posto na área tecnológica do país [EUA]. ... Mesmo assim, algumas companhias não conseguem encontrar programadores experientes a preço algum.[vii] (BYLINSKY[17], 1967 apud ENSMENGER; ASPRAY, 2002, p.142).

No final dos anos 1960 os cursos de computação já formavam profissionais em número significativo (CAMPBELL-KELLY, 2004, p.317), no entanto, a qualificação dos profissionais que estavam chegando ao mercado não era comparável àquela da geração anterior, na maioria das vezes educados em projetos militares de desenvolvimento de software de missão crítica. Ainda em Ensmenger e Aspray (2002, p.143), podemos encontrar um estudo do ACM SIGCPR (*Special Interest Group on Computer Personnel Research*) que alertava sobre a superabundância de um certo perfil de profissional de software, cujas habilidades e *performance* estavam aquém das necessidades do mercado. Ensmenger e Aspray traçam o problema do descasamento entre a formação universitária e as necessidades práticas dos negócios. Como a maioria dos cursos de computação surgia nos departamentos de engenharia, eles tendiam a ser mais direcionados para a máquina do que para a produção de software. O foco para o desenvolvimento do software, quando havia,

---

matematicamente elegantes. Linguagens de programação de alto nível, baseadas em manipulação de símbolos, facilmente apreendidas por não especialistas, eram necessárias para tornar essa nova força de trabalho efetiva, além de possibilitar às organizações não militares desenvolver seu próprio software sem incorrer nos elevados custos de contratação de *experts* no mundo dos computadores.

[16] AFIPS — *American Federation of Information Processing Societies*, estabelecida em 1961 pela ACM — *Association for Computing Machinery*, e pelos "*American Institute of Electrical Engineers*" e "*Institute of Radio Engineers*", que posteriormente se fundiram no IEEE — *Institute of Electrical and Electronic Engineers*. A AFIPS foi dissolvida em 1990.

[17] BYLINSKY, G., 1967, "Help Wanted: 50,000 Programmers", *Fortune*, mar.

constantemente recaía em aplicações científicas com bastante lastro na matemática. Após observar que "*os estudantes educados nesse ambiente tendem a absorver a tradicional distância acadêmica das aplicações práticas*", Ensmenger e Aspray citam Richard Hamming, na conferência *Turing Award* de 1968:

> A experiência delas [empresas empregadoras] é que os graduados em nossos programas parecem estar mais interessados em jogar, construir programas fantasiosos que não funcionam na realidade, escrever programas cheios de truques, etc., e são incapazes de disciplinar seus esforços próprios para que consigam fazer o que dizem que farão no tempo estabelecido e de uma maneira prática.[viii] (HAMMING[18], 1968 apud ENSMENGER; ASPRAY, 2002, p.144).

A partir dos anos 1960 ficaram muito claras as diferenças significativas entre os processos de produção de software e hardware. Começou a ser questionado, como inspiração para o desenvolvimento de software, o exemplo do processo de produção de hardware, com ênfase em revisão e verificação exaustivas das especificações antes do processo de produção. Os artefatos de software, aparentemente muito mais fáceis de alterar, permitem uma abordagem de tentativa e erro — programar e corrigir (*code and fix*). Além do que, produzir especificações precisas de artefatos de software era (e ainda é) muito mais complicado do que para hardware[19]. Também é importante mencionar que por conta da postura libertária dos anos 1960, questionadora do regime centralizado de autoridade, era comum que os programadores seguissem métodos próprios em detrimento dos métodos que suas empresas tentavam utilizar. Surgia a "cultura *hacker*" no meio acadêmico, cuja influência nas empresas se dava através da figura do "programador *cowboy*" que, com seus expedientes individuais e inexplicáveis, na última hora conseguia cumprir os prazos dos projetos. "Código espaguete" em profusão começou a ser gerado, trocando o aparente sucesso dos projetos alcançado na década anterior pelos altos custos com retrabalho e casos de fracasso. Parecia claro que eram necessários processos melhores de organização do trabalho e práticas mais disciplinadas. (BOEHM, 2006). A expressão *crise do software*, cunhada na famosa conferência da OTAN em 1968 (NAUR; RANDELL, 1969), sintetiza bem o que ocorria[20].

---

[18] HAMMING, R., 1987, "One Man's View of Computer Science", In: ACM Turing Award Lectures: The First Twenty Years, 1966-1985, p.207-218.

[19] Para se executar uma compra de US$5 milhões em hardware, provavelmente uma especificação de 30 páginas será suficiente. No entanto, se for para comprar U$5 milhões em software, com segurança, a especificação provavelmente atingirá 1.500 páginas. (ROYCE, 1970).

[20] Na conferência da OTAN, em 1968, foi cunhado o termo, desde então vastamente citado na literatura, (*gap* ou) *crise do software*. Com o rápido aumento do poder dos computadores e da complexidade dos problemas que podiam ser atacados, basicamente esse termo refletia a crença na dificuldade de se escrever, com correção, softwares inteligíveis e passíveis de validação precisa, em decorrência do crescimento de sua complexidade ser muito mais acelerado que o progresso da engenharia de software. Malgrado os grandes avanços, as demandas estariam além da capacidade, teoria, técnicas e métodos da

Então, com o custo relativo do software tornando-se mais expressivo e com a mão-de-obra sendo preponderante nesse custo, as atenções gerenciais foram direcionadas para o desempenho dos programadores. Surgia a percepção de que as organizações precisavam reduzir sua dependência das habilidades desses profissionais, sendo necessárias metodologias efetivas para o gerenciamento dos projetos de software e controle do processo de desenvolvimento.

> Em meados dos anos 1960, um perceptível deslocamento nos custos relativos de hardware e software ocorreu. Os custos decrescentes do hardware possibilitaram a utilização dos computadores em mais e maiores aplicações, que, por sua vez, requeriam softwares maiores e mais complexos. À medida que a escala dos projetos de software aumentava, eles se tornavam extremamente difíceis de supervisionar e controlar. Os problemas que agora afligiam os desenvolvedores de software eram mais gerenciais do que técnicos. Em uma apresentação na *Fall Eastern Joint Computer Conference* em 1965, J. Presper Eckert argumentou que a programação se tornaria algo gerenciável somente quando pudesse ser referida como "engenharia de software".[ix] (ENSMENGER; ASPRAY, 2002, p.154).

O argumento acima, em parte, é questionável, pois, afinal, supervisão e controle ocorreram nos projetos de defesa e não se tem notícia de projetos civis em tão grande escala. No entanto, pode-se julgar também que a argumentação parte da premissa que houve de fato supervisão e controle naqueles projetos e que, portanto, deviam ser replicados nos projetos civis em meados dos anos 1960 para garantir o mesmo sucesso[21]. Isso poderia ocorrer através da aplicação dos princípios da engenharia ao desenvolvimento de software.

Cabe ressaltar que, até aquela época, a conotação que se dava ao termo *programação* era bem maior que a atual. Michael Jackson, no prefácio de seu livro de 1975, julga absurda a divisão entre *projetação de sistemas* e *projetação de programas*, ou *programação*, afirmando que a palavra "programa" perdeu muito do seu significado quando os sistemas de processamento deixaram de ser apenas *batch* e tornaram-se sistemas de transações *online* (JACKSON, 1988). Hounshell (2002,

---

época, o que supostamente traria um sombrio cenário futuro para a indústria de software e para a própria sociedade, cada vez mais dependente de artefatos de software (NAUR; RANDELL, 1969, p.17, p.121).

[21] Nenhum dos autores pesquisados considera a possível relação entre o sucesso nos projetos de software dos grandes sistemas de defesa e a alta qualificação dos programadores, viabilizada por orçamentos bilionários. Milhares de programadores foram habilitados ao desenvolvimento de software de missão crítica para fins militares. Prevalece, na engenharia de software, a idéia, comum às engenharias, de buscar processos, métodos, técnicas e ferramentas que possibilitem a obtenção dos resultados desejados dependendo-se o mínimo possível da habilidade das pessoas que desempenham as tarefas (LEVESON, 1997). No clássico "*The Mythical Man-Month*", de 1975, Frederick Brooks já defendia a importância do foco nas pessoas para a obtenção de bom êxito nos projetos de software, trazendo a ilustração de que para um projeto com 200 profissionais, com seus 25 gerentes escolhidos entre os mais experientes e competentes programadores, é melhor, em termos de eficiência, custo e velocidade, demitir os outros 175 e realizar o projeto apenas com os 25 gerentes, pois bons projetos (*design*) decorrem de bons projetistas (BROOKS, 1995, p.30). Nessa mesma linha, como tantos outros, Pressman (2001) destaca as pessoas — o "fator humano" — como o mais importante para o sucesso dos projetos de software.

p.167) provoca: onde estão aqueles programadores? Quem eram os primeiros engenheiros de software? Nesta dissertação será considerada uma proximidade entre programação e programadores, vigente até os anos 1970, com a engenharia e os engenheiros de software, nos dias de hoje. É importante reconhecer, todavia, que não se pode utilizar direta ou linearmente o termo engenharia de software em referência àquele período anterior à sua existência, pois, uma vez estabelecida como uma *caixa-preta*[22], a engenharia de software passa, ela própria, a conformar a realidade do desenvolvimento de software. Em resumo, não podemos usar uma *caixa-preta* em referência a uma realidade anterior à sua existência, posto que a *caixa-preta*, quando passa a existir, ajuda na conformação dessa realidade (Latour, 2000, p.165).

2.3 A construção da engenharia de software

O estabelecimento das bases teóricas e práticas necessárias para a consolidação da engenharia de software como uma disciplina autônoma, como para qualquer outra disciplina (vide Latour (2000, cap.4)), é um esforço de construção, que aqui será brevemente abordado à luz dos *Estudos CTS (Ciência-Tecnologia-Sociedade)*.

Nas seções 2.1 e 2.2, podemos identificar as condições de produção da frase de John Eckert, quando sugere que o desenvolvimento de software devesse ser uma engenharia. Embora ele pudesse ter aventado outra metáfora, por exemplo, arte[23] ou arquitetura, é preciso reconhecer a aparente "naturalidade" do enquadramento do software como um artefato de engenharia, decorrente do cenário em que surgiu. O tom do relato de Herbert Benington, um dos envolvidos com a programação do SAGE, sobre a explicação para o sucesso atingido num projeto daquela complexidade, permite inferir o apelo que o termo engenharia de software teria junto à comunidade envolvida com a computação naquela época:

> É fácil para mim apontar um único fator que garantiu nosso relativo sucesso: todos nós éramos engenheiros e tínhamos sido treinados para organizar nossos esforços junto a linhas [de produção] de engenharia.[x] (BENINGTON, 1956).

O treinamento, a educação e a prática dos programadores diz muito sobre a forma de se encarar os artefatos de software e seu processo de produção. Em

---

[22] Caixa-preta é um fato plenamente aceito ou um objeto [artefato] não problemático. É um todo organizado que o construtor de fatos [e artefatos] quer propagar no tempo e no espaço e que dá a impressão de manter, por si só, o controle do comportamento de todos os seus componentes alistados, tendo ares de verdade pura (LATOUR, 2000, p.216).

[23] Donald E. Knuth usou essa metáfora já no título de sua série clássica de 3 volumes: "*The Art of Computer Programming*", publicada entre 1968 e 1973.

decorrência da prática bem sucedida observada até aquele momento, eivada de influência das grandes firmas de engenharia, da forte cultura de gerenciamento de projetos, e do surgimento dos primeiros cursos de computação em departamentos de engenharia, devia parecer que os artefatos de software "naturalmente" eram objetos do domínio da engenharia. Hohn (1955) revela o forte vínculo e uso inicial da computação, e, portanto, do software, nos domínios de cálculos de engenharia e científicos, sendo, nesse início, apenas marginal o seu uso para o processamento de dados nas empresas. Boehm (2006) destaca a forte orientação da engenharia do hardware presente manifestamente nas principais sociedades de profissionais de software, como a *Association for Computing* ***Machinery*** e a *IEEE* ***Computer*** *Society* (grifos do autor). Por tudo isso, parece "natural" ter-se buscado estabelecer o desenvolvimento de software como uma engenharia para garantir a volta do controle e do sucesso nos projetos de desenvolvimento de software obtidos nos projetos pioneiros (nos anos 1950 e início dos anos 1960).

Contudo, a afirmação de que "desenvolver software é engenharia", como qualquer outra afirmação, não é capaz, por si só, de estabelecer-se como fato, ou, ao contrário, ser relegada à categoria de ficção. Somente o uso posterior que os outros fizerem de uma afirmação a levará à condição de fato ou a fará desaparecer como uma ficção (LATOUR, 2000).

Para existir como fato, uma afirmação precisa superar as *controvérsias* em torno de sua validade, vencendo, afinal, toda argumentação que lhe for contrária. Se resistir até o final, a afirmação se tornará fato e assim conformará a realidade que lhe é subjacente. Durante longo período, era fato o Sol girar em torno da Terra. Hoje, a realidade que aceitamos é que a Terra gira em torno do Sol. Agora que a controvérsia cessou, consideramos que a realidade "sempre" foi esta, relegando à ficção o que se acreditou por vários séculos. É difícil não mencionar o caso de Plutão, cuja condição de planeta apenas recentemente deixou de ser fato. O ponto que buscamos chamar a atenção é que a verdade surge somente quando a controvérsia se encerra. Se para alguns de nós é estranho aceitar a verdade de Plutão não mais ser um planeta, embora aceitemos facilmente a verdade da Terra girar em torno do Sol, é porque a controvérsia sobre Plutão está morna, recente, e aquela acerca do sistema heliocêntrico, que teve conseqüências sobre a ordem mundial estabelecida muito mais expressivas, já está fria, devidamente encerrada. Somente a partir do instante em que termina a controvérsia, passamos a aceitar que a verdade conformada estivera lá o tempo todo. Quando extingue-se a controvérsia, e o fato ou artefato torna-se

estabilizado e real, foi produzida uma *caixa-preta*[24], algo que assume ares de verdade pura, que prescinde de qualquer autor para explicar seu funcionamento. Explica-se, ou funciona, por si só, servindo desta forma para persuadir outros acerca da verdade que encerra.

A criação de uma *caixa-preta*, seja ela uma afirmação que tornou-se fato, ou seja ela uma máquina ou artefato tecnológico bem sucedido, é decorrente da estabilização da rede sociotécnica que a engendrou. Para Latour (2000, p.230), um fato ou um artefato bem sucedido concentra em si o maior número possível de associações resistentes entre as mais diversas entidades, humanas e não-humanas, principalmente se tiver se transformado numa *caixa-preta*, algo com aparente moto próprio. Muitas vezes recebem adjetivos como "fatos inegáveis", "máquinas sofisticadas", "teorias eficazes", "provas irrefutáveis", mas são sempre resultados da estabilização de suas respectivas redes sociotécnicas.

O referencial metodológico dos *Estudos CTS* permite explorar, por exemplo, desde um processo de desenvolvimento de software particular até a construção de uma disciplina, como a engenharia de software. Na verdade, este instrumental trata da criação de fatos e artefatos, reconhecendo-a como um esforço coletivo, como um esforço de alistamento de aliados, humanos e não-humanos, relacionados em rede. A estabilização da rede vai ocorrendo à medida que as controvérsias existentes vão sendo vencidas, o que favorecerá a existência do fato ou do artefato que, por sua vez, concomitantemente, vai contribuindo também para a estabilização da (sua) rede.

Voltando à afirmação de John Eckert, vemos que seu *status*, até hoje, tem dependido de todo uso posterior que vem sofrendo. Podemos dizer que a afirmativa "*desenvolvimento de software é uma disciplina de engenharia*" vem caminhando na direção de tornar-se um fato. Ou, dizendo de outra forma, esta afirmação vem tornando-se uma *caixa-preta* e persuadindo, cada vez mais, acerca do que é, ou não é, desenvolver software.

Provavelmente, o mais influente uso posterior da afirmação de Eckert ocorreu três anos após sua palestra, em 1968, em uma conferência, organizada pelo comitê de ciência da OTAN — Organização do Tratado do Atlântico Norte —, para discutir os problemas relacionados com o desenvolvimento de software. Os organizadores da conferência da OTAN, segundo Peter Naur e Brian Randell (1969, p.13), escolheram como título "*'engenharia de software' para serem* ***provocativos***"[xi] (grifo nosso). Não estavam, ao que parece, primordialmente defendendo o desenvolvimento de software como ramo da engenharia, mas sim querendo alistar o máximo de aliados possíveis

[24] Caixa-preta: vide nota 22.

para os debates. A expressão "engenharia" naquele momento era uma metáfora que, para os organizadores, remetia à idéia de regulamentação e controle da profissão existente nas diversas modalidades de engenharia (PARNAS, 1997). Mas foi essa a metáfora que alinhou esforços e interesses em infundir a disciplina da engenharia no processo de desenvolvimento de software. Inúmeros atores passaram a adaptar idéias de outras engenharias ao desenvolvimento de software. Houve um grande entendimento do papel da abstração e da separação de conceitos, foram introduzidas as noções de modularidade e ciclo de vida de software, processos, medições, especificações abstratas e notações para registrá-las (LEVESON, 1997).

Um inegável desenvolvimento, normalmente reconhecido como "técnico", foi empreendido, desde os anos 1960, para que o desenvolvimento de software pudesse gozar, hoje, do *status* de disciplina de engenharia, inclusive com um corpo de conhecimento estabelecido (SWEBOK[25]). No entanto, de forma concomitante e inseparável desse desenvolvimento "técnico", precisamos reconhecer também o vasto empreendimento "social" que precisou ser realizado. Quando proferida em 1965, a frase de John Eckert era muito mais suscetível às pressões que porventura poderiam tê-la tornado mera ficção. Mas, desde então, alistaram-se na rede sociotécnica da engenharia de software inumeráveis eventos e acontecimentos, como, por exemplo, a conferência da OTAN, considerada por muitos um marco para a engenharia de software. Portanto, hoje em dia, essa rede, além de muito mais "técnica", necessariamente é também muito mais "social" — quanto mais percebermos um fato ou artefato como técnico, tanto mais social ele o será (LATOUR, 2000, p.103) —, uma vez que a comunidade envolvida tem que reconhecer e convencionar símbolos e elementos presentes na prática do desenvolvimento de software qualificáveis como "técnicos". Se cada desenvolvedor realizar sua prática de desenvolvimento isoladamente, negando o "social", não haveria parâmetros para julgá-la "técnica", não haveria convenções que possibilitariam *a priori* julgar a prática como puramente "técnica". Portanto, quanto mais "técnica" mais "social" será a engenharia de software, uma vez que sua rede socializa institutos, governos, universidades, publicações,

[25] SWEBOK — *Guide to Software Engineering Body of Knowledge* — é um projeto da *IEEE Computer Society* que busca compilar o corpo de conhecimento atual da engenharia de software. A gerência do projeto é feita pela Escola Superior de Tecnologia da Universidade de Quebec, sendo que diversas organizações mundiais estão envolvidas. Maiores informações disponíveis em www.swebok.org. Cabe observar que até mesmo nestes *handbooks*, é impossível verificar uma isenção de valores que permita tomá-los por "puramente técnicos". Como traz James Tomayko (2002, p.67), a motivação para o desenvolvimento de um *handbook* para a engenharia de software decorreu "*dos engenheiros de software de dez anos atrás terem visto o desenvolvimento de handbooks como um necessário sinal de* ***maturidade (afinal, o primeiro handbook de engenharia apareceu em 1853!)***" (grifo nosso).

regulamentações profissionais, associações, certificações, além de teorias, métodos, técnicas, ferramentas, computadores.

Atualmente aceito na comunidade acadêmica e prática, o termo engenharia de software deixou de ser apenas metáfora, uma afirmação ou suposição qualquer, e vem tornando-se um fato: "*em 20 anos [...], a engenharia de software evoluiu de uma idéia obscura praticada por um grupo relativamente pequeno de fanáticos para uma legítima disciplina de engenharia*"[xii] (PRESSMAN, 2001, p.xxv). Desenvolver software vem consolidando-se como uma engenharia graças a todos aqueles que foram alistados pela metáfora inicial e passaram a orientar sua prática de maneira coerente com a inspiração que dela advinha, contribuindo para a expansão e estabilização da rede sociotécnica da engenharia de software.

Nomear o desenvolvimento de software de "engenharia" foi um claro movimento, como veremos a seguir, de *tradução* bem sucedido, pois logrou êxito em alistar aliados — por exemplo os participantes da conferência da OTAN — e controlar seu comportamento — esses aliados, e muitos outros, passaram a agir, a serem vistos e a se verem como engenheiros —.

Latour (2000) e Callon (1999) propõem que criar e manter a rede sociotécnica é uma questão determinante para o sucesso do construtor de fatos/artefatos. Todavia, o construtor se depara com o problema da propagação de fatos e artefatos, no tempo e espaço, ser um processo cujo sucesso dependerá menos dele e muito mais das ações que outros realizarem quando tiverem em mãos aqueles fatos e artefatos. Sendo assim, como lidar com a incerteza sobre se os outros tomarão, ou não, o fato/artefato construído em suas mãos? Se não o tomarem, o suposto fato ou artefato existiu pontualmente apenas no mundo de seu construtor. Se o tomarem, o construtor não tem a menor garantia de que aquele fato/artefato não será desfigurado até perder totalmente sua identidade com o artefato original, expondo-se ao risco, inclusive, de não ser considerado seu legítimo construtor. Para superar esta incerteza é necessário realizar tarefas aparentemente tão contraditórias que soam inexeqüíveis (LATOUR, 2000, p.178):

- Alistar outros aliados para que participem da construção do fato/artefato, criando o interesse de novos aliados por ele, aumentando assim sua rede; e
- Controlar o comportamento de todos aliados para manter previsíveis suas ações.

Para dar conta desta contradição — manter o controle ao mesmo tempo que se busca mais aliados, o que dificulta o exercício do controle — é necessário o artifício das *traduções*. Tradução é um processo pelo qual construtores de fatos e artefatos,

em movimentos de ajustes mútuos e negociação de interesses, interpretam seus próprios interesses, e os das entidades que querem alistar, em busca de equivalências e coincidências, nem sempre existentes *ab initio*, que viabilizem a constituição de associações e a socialização de todos esses elementos para a construção e disseminação do fato ou artefato. Como já foi dito, no arcabouço metodológico dos *Estudos CTS* trata-se com equivalência analítica todas as entidades alistadas, não se fazendo distinção alguma entre humanos e não-humanos. Tampouco aceita-se a possibilidade de delimitações nítidas entre "fatores técnicos" e "fatores sociais". O fato ou artefato produzido é indissociável de seu contexto de produção. Traduções são fundamentais para manter estáveis as associações no decurso da resolução das controvérsias. Elas têm uma forte característica de incerteza, pois que, a *priori*, não se sabe se os elementos agirão como esperado ou mesmo se permanecerão alistados. Para John Law (1992), a resultante de um processo bem sucedido de tradução é a ordem observada como resultado de uma rede sociotécnica, às vezes *pontualizada*[26] em um dispositivo, em uma instituição, em uma organização, em um artefato, em uma disciplina, etc.

No centro do "redemoinho" está uma metáfora. No "redemoinho" se envolvem fatos, artefatos e seus autores. Observadores como A ou D não vêem seu centro, eles tendem a reconhecer que lá existe um fato, uma caixa-preta. Observadores como B e C têm a oportunidade (nem sempre aproveitada) de perceberem o centro como construção e não como "natureza", algo "pronto" e descoberto. (Vide nota 27).

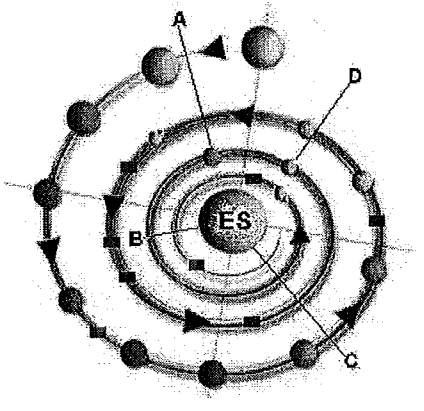


Figura 2 — Uma Metáfora Transformando-se em Fato

Podemos imaginar o processo de tradução, que vem urdindo a rede sociotécnica da engenharia de software, como um "redemoinho"[27] (Figura 2), que vai ganhando crescente amplitude ao envolver mais atores, artefatos e afirmações. À medida que aumenta suas dimensões, o "redemoinho" protege em seu centro aquilo que já foi apenas uma metáfora — "engenharia" —, que vai ficando cada vez mais

[26] O conceito de *pontualização* será detalhado mais adiante na página 102.

[27] A analogia com o redemoinho serve para ilustrar particularidades de uma realidade maior, melhor aproximada através da metáfora *rede* (sociotécnica). O "redemoinho" explicaria uma configuração particular de uma *rede* formada pelos fatos, artefatos, seus concebedores e usuários. A engenharia de software não é apenas o centro do "redemoinho", ela é o próprio redemoinho, melhor dizendo, é a própria

distante das novas afirmações, artefatos e autores capturados. Da periferia do "redemoinho", dificilmente se enxerga a metáfora em seu centro. Ao contrário, sob o ponto de vista da periferia, outros fatos, outros artefatos, outros autores já capturados ajudam a conformar a visão de que no centro há uma *caixa-preta*: *desenvolver software é engenharia* (TEIXEIRA; CUKIERMAN, 2005a).

É preciso reconhecer que a engenharia de software ainda não se configurou uma *caixa-preta*. Se fosse possível uma analogia, seu processo de tradução, de resolução de controvérsias, estaria mais à semelhança do caso de Plutão que do caso do sistema heliocêntrico, pois ainda existem controvérsias acerca do *status* de engenharia da atividade de desenvolvimento de software. Michael Mahoney (2004) diz que tanto engenheiros de software quanto historiadores têm se debruçado sobre a história da engenharia de software e vêem-na como uma disciplina ainda em processo de definição, ainda em dúvida sobre sua condição de disciplina de engenharia. Mahoney afirma ainda que após a conferência da OTAN, na busca de quase 40 anos por consolidá-la como uma disciplina de engenharia, têm aparecido diferenças fundamentais entre os artefatos de software e os demais artefatos das outras modalidades de engenharia, como máquinas, construções civis, processos químicos, circuitos elétricos, semi-condutores. Nessa linha, James Tomayko provoca a respeito dos engenheiros de software não serem lembrados no rol dos demais engenheiros:

> [A referência ao termo engenharia de software na conferência da OTAN] desencadeou uma verdadeira tempestade de freqüentes e amargos debates sobre a viabilidade do software como engenharia. A pouca conseqüência [prática] deste debate de mais de trinta anos é evidente numa chamada para as atividades da Semana Nacional de Engenharia do Carnegie Science Center de Pittsburgh. A nota dizia que 'engenheiros de *todas* as disciplinas — *química, elétrica, mecânica, civil*', participariam.[xiii] (TOMAYKO, 2002, p.65, grifo nosso).

A seguir, dois exemplos sobre fatos/artefatos constituídos e enredados na rede sociotécnica da engenharia de software, servirão para ilustrar os conceitos apresentados. A própria engenharia de software, como qualquer outro fato ou artefato, deve sua existência às traduções e ao encerramento das controvérsias. No primeiro exemplo, veremos que considerar o uso do comando *go to* uma prática ruim de programação virou um fato, uma *caixa-preta*. Já o segundo exemplo, sobre o inevitável sucesso e a "naturalidade" do paradigma de orientação por objetos, ainda está envolvido em controvérsias.

---

rede. Uma metáfora transformada em fato não apresenta um ponto central, ou de origem bem definida. Um fato surge de / em sua *rede sociotécnica*, sem um centro definido.

2.3.1 Usar *go to* é prejudicial

Hoje em dia é largamente aceito o fato de que usar o comando *go to*[28] não é boa prática de programação. Durante os anos 1950 e 1960 porém, programar era sinônimo de programar usando *go to*. Como essa era a realidade, diversos outros aspectos se estruturavam a partir dela, como, por exemplo, o uso disseminado de fluxogramas e uma hierarquia de trabalho que, rígida e arbitrariamente, distinguia entre programadores seniores e juniores, e entre programadores e codificadores. Campbell-Kelly (2004, p.6) observa que, para o mercado de aplicativos e pacotes que nascia, um dos primeiros e mais influente produtos de software foi o AutoFlow, lançado pela *Aplied Data Research* em 1965. O Autoflow produzia automaticamente os fluxogramas para a documentação dos programas. Portanto, era de se esperar a forte reação e controvérsia que surgiram, em 1968, com o artigo "*Go to Statement Considered Harmful*" de Edsger Wybe Dijkstra, uma afirmação que parecia ficção, algo sem objetividade. A controvérsia ainda existia em 1987, como denota o texto de Frank Rubin (1987) intitulado "*'GOTO considered harmful' considered harmful*", em que critica o argumento de Dijkstra e traz um contra-exemplo usando o comando *go to*, mais rápido, simples e fácil de entender do que o análogo sem *go to*.

> O mais notado item já publicado em *Communications* [*of the* ACM] é uma "Carta ao Editor" de Edsger W. Dijkstra chamada "*Go To Statement Considered Harmful*" que tenta argumentar sobre por que o constructo GOTO seria prejudicial. Embora o argumento fosse acadêmico e inconvincente, seu título parece ter tornado-se fixo na mente de todo gerente de programação e metodologista. Conseqüentemente, a noção de que GOTO é prejudicial é aceita quase universalmente, sem questão ou dúvida. Para muitas pessoas, "programação estruturada" e "programação sem GOTO" têm se tornado sinônimos. Isso tem causado um prejuízo incalculável para o campo da programação que perdeu uma eficaz ferramenta. É algo como os açougueiros banirem as facas porque algumas vezes se cortam. [...] Em resumo, a crença que GOTOs são prejudiciais parece ter tornado-se uma doutrina religiosa, não comprovado por evidência. Eu não sei se posso fazer algo que irá desalojar dogma tão profundamente entrincheirado. Pelo menos, posso tentar reabrir a discussão mostrando um claro exemplo onde GOTOs reduzem significativamente a complexidade do programa. [...] Toda minha experiência me compele a concluir que é hora de deixar o dogma da programação sem GOTO. Ela falhou em provar seu mérito (RUBIN, 1987)[xiv].

O *insight* de Dijkstra não veio de uma "genial inspiração", não partiu de uma definição teórica, tampouco é algo que se possa dizer estritamente "técnico". Ao contrário, surgiu para resolver problemas práticos em que estava envolvido e aproveitava experiências de outros, sendo um claro exemplo de construção sociotécnica. Em suas próprias palavras:

> Pelos padrões atuais nós não sabíamos [, naquela época,] o que estávamos fazendo; nem sonhávamos em dar qualquer garantia que nossa

[28] *Go to* é um constructo muito comum em linguagens de programação mais antigas, que permite o desvio incondicional do fluxo de execução de um programa.

> implementação [de um compilador ALGOL 60] pudesse estar correta porque sabíamos muito bem que nos faltava o *conhecimento teórico* que seria necessário para isso (DIJKSTRA, 2002, p.343, grifo nosso)[xv].
>
> Por vários anos tenho me familiarizado com a observação de que a qualidade dos programadores é uma função decrescente da densidade de comandos *go to* nos programas que produzem. [...] Sou conhecedor de quem influenciou minhas idéias? É óbvio que fui influenciado por Peter Lundin e Christopher Strachey. [...] Comentários sobre o indesejável uso do *go to* está longe de ser novo [...], mas não estou sendo capaz de resgatar sua origem, presumivelmente, devem ter sido feitos por C. A. R. Hoare (DIJKSTRA, 1968)[xvi].

De tanto ser reutilizada, apropriada e transformada, inserindo-se em outras cadeias de fatos e artefatos — como a programação estruturada[29] —, a afirmação original de Dijkstra conformou-se como fato. Hoje, várias linguagens de programação sequer implementam *go to* e até mesmo as linguagens mais antigas (FORTRAN, COBOL e BASIC), em versões posteriores, passaram a trazer construtos para a programação estruturada. Podemos crer que pelo menos a geração mais recente de programadores provavelmente não teve contato com o comando *go to*.

Quando apresenta seu trabalho, um construtor age como um *porta-voz* de todas as entidades que congregou em torno do fato/artefato que quer tornar uma *caixa-preta*. O *porta-voz* é alguém que fala em nome de um outro que não fala. Esse outro pode ser um humano ou um não-humano. Um líder sindical, por exemplo, é porta-voz dos trabalhadores sindicalizados, mas, da mesma forma, um programador é porta-voz de um programa do qual ressalta as qualidades (LATOUR, 2000, p.119-129). Para afirmar que o uso indiscriminado do comando *go to* era ruim, Dijkstra amealhou diversas entidades que se sobrepunham para sustentar sua afirmação. Dentre elas, pode-se apontar os argumentos acerca de erros e dificuldades causadas pelo *go to*, as linguagens de programação eficientes sem *go to*, as afirmações de outros autores, o rigor matemático, a programação estruturada, os desenvolvedores de software. Essas entidades, cessada a controvérsia, em uníssono corroboraram a afirmação de Dijkstra, que tornou-se o respeitável *porta-voz* de todas elas.

Passou a ficar difícil dissociar Dijkstra, o *porta-voz*, daquilo que afirmou sobre aqueles que representava, e, por isso mesmo, foi-lhe atribuído o mérito de vencer as *provas de força*[30], de mostrar com clareza o que havia entre sua afirmação e o que a sustentava. Latour (2000, p.285-289) caracteriza os *julgamentos de atribuição de responsabilidade* como constituindo aquilo que chama de *mecanismo secundário* de da formação das redes. Através desse mecanismo se explica porque é atribuída a

---

[29] DIJKSTRA, E. W., 1970, *Notes on Structured Programming*. Technical Report, Technological University Eindhoven-70-WSK-03, Netherlands.

[30] Podemos entender prova de força como o questionamento da validade de uma afirmação frente ao que ela representa, a explicitação daquilo que sustenta a afirmação (LATOUR, 2000, p.129).

responsabilidade pela construção de fatos e artefatos apenas a uns poucos, embora *tal construção decorra de um coletivo de entidades alistadas*, o que constitui o *mecanismo primário* de construção das redes, segundo Latour. Sobre isto, o exemplo de Dijkstra é esclarecedor, pois, como ele mesmo conta, seu amigo Niklaus Wirth, que não aparece na discussão sobre o *go to*, desempenhou um papel fundamental.

> Em 1968, "*Communications of the ACM*" publicou um texto meu sob o título: "*The goto statement considered harmful*", que nos últimos anos seria o mais freqüentemente referenciado, lamentavelmente, porém, muitas vezes por autores que não viram mais do que seu título apenas. Este texto tornou-se a pedra angular de minha fama porque virou um modelo: nós poderíamos ver todo tipo de artigo sob o título "X considerado prejudicial" para quase todo X, inclusive um intitulado "Dijkstra considerado prejudicial". Mas, de fato, o que aconteceu? Eu tinha submetido um artigo com o título "*A case against the goto statement*", que, a fim de acelerar sua publicação, o editor o tinha trocado para uma "Carta ao Editor", e nesse processo ele inventou um novo título para o artigo! O editor era Niklaus Wirth.[xvii] (DIJKSTRA, 2002, p.346).

Ser objetivo é ser o *porta-voz* que não é traído pelos seus representados, que se comportam de forma coerente àquela descrita pelo *porta-voz*. Ser subjetivo é deixar de falar em nome de pessoas ou coisas, para representar apenas a si mesmo (LATOUR, 2000, p.129-130). Durante alguns séculos, eram objetivos os que defendiam que o Sol girava em torno da Terra. O tempo acumulou provas de força que fizeram essa afirmação e seus *porta-vozes* sucumbirem. No exemplo do *go to*, vencidas as provas de força, ficamos com a impressão que Dijkstra apenas representou fidedignamente a realidade preexistente que, mais cedo ou mais tarde, iria expressar-se por si própria. Dijkstra foi objetivo, não representou a si mesmo. Subjetivos ficaram, a partir de então, todos os envolvidos na rede anterior, cujo estilo de programação, numa avaliação retrospectiva, passou a representar um padrão de pior qualidade, uma vez que se baseava em *go to*. A estabilidade da rede sociotécnica do *go to* ruiu sob a injunção da nova rede.

> [...] A necessidade de fluxogramas era questão de fé. Nos anos 1950 e 1960, o uso de fluxogramas na computação criou sua própria literatura e cultura. O Instituto de Padrões Americano produziu convenções, matrizes estêncil e outros auxílios para desenhar existiam nas papelarias, e os professores de programação exigiam que seus alunos usassem fluxogramas. Na segunda metade dos anos 1960, softwares de apoio foram desenvolvidos para produzi-los automaticamente, como o Autoflow. No início dos anos 1970, *a moda da "programação estruturada" varreu 20 anos de história de fluxogramas em cerca de 3 anos*. De modo similar, a hierarquia de trabalho foi alterada e a distinção arbitrária entre programadores seniores e juniores foi perdida. De fato, o termo "codificador" caiu em desuso.[xviii] (CAMPBELL-KELLY, 2004, p.69-70, grifo nosso).

É importante atentarmos para a imbricação das questões técnicas e sociais no processo de construção das redes, por isso mesmo denominadas sociotécnicas. É impossível qualificar estritamente como "técnica" a discussão acerca do *go to* e o encerramento da controvérsia. Por exemplo, um trabalho que normalmente é

apontado como a "origem técnica", matemática, do "teorema da programação estruturada" é o de Corrado Böhm e Giuseppe Jacopini[31], de 1966. No entanto, este trabalho não é considerado tão significante para a popularização da programação estruturada e o conseqüente desuso do comando *go to*. Sobretudo porque, diferente do texto de Dijkstra, a prova de Böhm e Jacopini não aborda de maneira prática a questão da programação estruturada. De fato, segundo o próprio Dijkstra (1968), a construção que propõem tornaria o programa mais obscuro, o que conflita diretamente com a defesa da programação estruturada, a de supostamente tornar os programas mais claros e simples de serem lidos e entendidos.

Na construção de fatos e artefatos, ao longo das controvérsias inevitavelmente ocorrem *traduções* e deslocamentos de interesse e objetivos que levam a resultados, na maioria das vezes, não previstos no início. Por exemplo, embora consigamos caracterizar Dijkstra como *porta voz* do movimento contrário ao uso do *go to*, partidário da programação estruturada, é preciso reconhecer que sua opinião não prevaleceu na íntegra, com a total abolição do *go to*. Por exemplo, Donald Knuth em 1974 não concordava com a abolição irrestrita do *go to*. No artigo "*Structured Programming with Goto Statements*"[32] ele defendia, através de exemplos, que em determinadas situações utilizar *go to* tornava o código mais eficiente além de garantir maior clareza. Embora hoje exista um consenso em torno da propriedade dos conceitos de programação estruturada, o comando *go to* foi substituído por "*go to's*" mais controlados nas linguagens de programação mais recentes para permitirem, na maioria das vezes, a implementação de pontos de saída múltiplos de trechos de programa. Essa idéia foi se estruturando ao longo da controvérsia, contradizendo a regra que Dijkstra defendia de ponto único de entrada e de saída das rotinas. O que prevaleceu foi o ponto único de entrada e a aceitação de múltiplos pontos de saída, inclusive porque isso torna os programas mais claros. Hoje em dia, são comuns, nas linguagens, mecanismos para o tratamentos de exceção, além dos comandos do tipo "*break*" e "*return*" — *go to*'s travestidos — para permitir múltiplos pontos de saída de um determinado trecho de código.

O conhecimento necessariamente apresenta um forte lastro material, além de ser um produto social, e não algo gerado exclusivamente a partir da operação de um método científico privilegiado, baseado em alguma teoria explicativa ou em um gênio criador (LAW, 1992). O caso do *go to* exemplifica-o perfeitamente, pois não surgiu a

[31] BÖHM, C.; JACOPINI, G., 1966, "Flow Diagrams, Turing Machines and Languages with only Two Formation Rules", In: Communications of the ACM, v. 9, n. 5, p.366-371.
[32] KNUTH, D., 1974, "Structured Programming with Goto Statements", In: Computing Surveys, v. 6, n. 4, p.261-301.

partir de uma derivação teórica, nem originou-se de uma inspiração genial de Dijkstra, mas, ao contrário, foi construído por uma rede sociotécnica que cuidou de materializar argumentos em artigos, publicações, linguagens de programação, compiladores, métodos e programas. Além disso, é preciso reconhecer a contingência e a imprevisibilidade sempre presentes nos processos de construção dos fatos e artefatos (processo de *tradução*, de construção de redes sociotécnicas). No exemplo do *go to*, não podemos desconsiderar o peso que a troca do título do artigo de Djikstra — uma questão retórica, do campo da linguagem, e não "técnica" — teve na dinâmica de solução da controvérsia relacionada.

2.3.2 Nós pensamos "naturalmente" orientado a objetos (OO)

A partir dos anos 1990 o paradigma de *orientação a objetos*[33] (OO) começou a ser bastante utilizado e, paulatinamente, vem substituindo as abordagens clássicas de desenvolvimento de software (PRESSMAN, 2001, p.541). Das "revoluções" que capturaram mentes e corações de desenvolvedores, nenhuma outra tem sido mais vigorosa e sedutora que a OO (HATTON, 1998). Vários autores exaltaram suas qualidades afirmando se tratar de um paradigma unificador, que traria um salto qualitativo no desenvolvimento de software, porque a mesma representação utilizada na fase de análise poderia ser utilizada nas fases de projetação (*design*) e codificação (MARTIN, 1993, p.3, RUMBAUGH, 1994, p.193). Defendia-se o inevitável sucesso da abordagem OO porque ela estaria baseada em princípios com os quais entendemos o mundo (BOOCH, 1991, p.137-141): "[a abordagem OO] *está baseada em conceitos que aprendemos no jardim de infância: objetos e atributos, classes e membros, o todo e suas partes*" (COAD; YOURDON, 1990, p.1).

Vimos que se o *go to* entrou em outras cadeias de fatos e artefatos, como a programação estruturada, o mesmo se dá com o paradigma OO. Por exemplo, como insumos na cadeia OO, estão os tão importantes conceitos de *ocultamento de informações* (*information hiding*) e tipos abstratos de dados, do trabalho de David Lorge Parnas[34]. Já com a própria abordagem OO servindo de insumo, vemos surgirem as idéias de *arquitetura de software*[35] e, posteriormente, dos padrões de

[33] Orientação a Objetos é uma abordagem de desenvolvimento de software que organiza tanto o problema quanto sua solução como uma coleção de objetos discretos. Estrutura de dados e comportamento estão incluídos na representação [dos objetos] (PFLEEGER, 2001, p.257).

[34] PARNAS, D.L., 1972, "On the Criteria to Be Used in Decomposing Systems into Modules". In Communications of the ACM, v. 15, n. 12, p.1053-1058.

[35] GARLAN, D., SHAW, M., 1996, *Software Architectures: Perspectives on an Emerging Discipline*. Upper Saddle River.

projeto (*design patterns*)[36]. Adicionalmente, numa tentativa de incluir a OO em outras cadeias de fatos e artefatos, Brad Cox (1990) aparece em um movimento que tentava colocá-la como integrante de uma "*bala de prata*", supostamente capaz de debelar a *crise de software*[37] através de um movimento bastante amplo que demandaria a remodelagem dos pressupostos da produção de software[38].

> A bala de prata é uma mudança cultural ao invés de tecnológica. É um deslocamento de paradigma — *uma revolução industrial de software* baseada em componentes reutilizáveis e intercambiáveis que irão *alterar* o *universo do software tão seguramente quanto o fez a revolução industrial* com a manufatura. [...] Para compreender efetivamente o significado de orientação por objetos é preciso se dar conta que ela é um fim, não um meio — um objetivo ao invés de tecnologias para atingi-lo. [...] Orientação por objetos significa abandonar a visão centrada no processo do universo do software [...] em favor de uma centrada no produto.[xix] (COX, 1990, grifos nossos).

Cox faz referência ao termo utilizado no clássico artigo de Frederick Brooks (1986) "*No Silver Bullet*", que defende a impossibilidade do surgimento de alguma solução tecnológica — uma *bala de prata* — capaz de gerar um ganho de uma ordem de magnitude em produtividade, confiabilidade e simplicidade no desenvolvimento de software, pois o desenvolvimento tecnológico atua apenas nas tarefas que Brooks denomina *acidentais*, não nas *essenciais*, do desenvolvimento de software. Seriam tarefas essenciais aquelas relacionadas com a apreensão e modelagem da complexa estrutura conceitual que compõe a entidade abstrata software. Tarefas acidentais dizem respeito à representação dessa entidade abstrata em linguagens de programação e ao mapeamento em linguagens de máquina, com limitações de velocidade e espaço de armazenamento. No próprio artigo "*No Silver Bullet*", Brooks desqualifica a OO como aquilo que chamou de bala de prata:

> O ganho de uma ordem de magnitude pode resultar da programação orientada a objetos somente se toda a desnecessária argumentação subjacente à especificação de tipos que permanece hoje em nossas linguagens de programação for responsável sozinha por nove décimos do trabalho envolvido na projetação de um programa. O que duvido.[xx] (BROOKS, 1986).

Quando revisitou o assunto, uma década depois, no artigo "*No Silver Bullet Refired*" (in BROOKS, 1995, p.207-226), Brooks comentou:

> Um excelente artigo de Brad Cox de 1990, "There Is a Silver Bullet", argumenta eloqüentemente sobre a abordagem de componentes reutilizáveis, intercambiáveis como um ataque à essência conceitual do

---

[36] GAMMA, E., et al., 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley.

[37] Crise de *software*: vide nota 20.

[38] A proposição de Cox, que inclusive faz alusão às tecelagens na América colonial, reflete a busca de impor uma organização de trabalho já bem estabelecida na indústria, não sendo fundamentalmente diferente em nada dos princípios da administração científica de Taylor (ENSMENGER; ASPRAY, 2002, p.151)

> problema. Eu concordo entusiasticamente. Cox porém entende mal [meu artigo] "No Silver Bullet" [...], ele o lê como declarando que as dificuldades com [o desenvolvimento de] software surgem "de alguma deficiência em como programadores constróem software hoje." Meu argumento era que as dificuldades essenciais são inerentes à complexidade conceitual das funções do software a serem projetadas em qualquer tempo, com qualquer método.[xxi] (BROOKS, 1995, p.210).

Diferente do que ocorreu com a afirmativa sobre o *go to*, as afirmações de que o paradigma OO está fadado ao sucesso porque se baseia em princípios com os quais entendemos o mundo, ou que o paradigma OO viabilizará uma revolução industrial do software, pelo menos por enquanto ainda não podem ser consideradas fatos, como denotam o título do artigo de Hatton (1998), "*Does OO Sync with How We Think?*", e o relatório *Extreme Chaos*[39], do *Standish Group International* Inc., de 2001, que revela que quando o software é desenvolvido a partir do zero, nos EUA, apenas 28% seguem o paradigma OO, enquanto 72% usam métodos e linguagens tradicionais.

Se remontarmos ao surgimento da abordagem OO, chegaremos ao ano de 1962 quando a linguagem SIMULA, destinada a simulação por eventos discretos, estava sendo projetada. Parece que seus autores, assim como Dijkstra no caso do *go to*, não partiram de derivações teóricas para moldar o conceito de objetos na linguagem SIMULA. Tampouco tiveram por motivação o "nosso modo de pensar e entender o mundo", ou vislumbraram alguma "revolução industrial do software" ao desenvolvê-la. Ao contrário, Ole-Johan Dahl e Kristen Nygaard, utilizando idéias já existentes, perceberam ser muito útil, para o problema em que estavam envolvidos, criar um conceito — o de objeto — que associasse estrutura de dados e operações.

Curiosamente, a história da linguagem SIMULA traz algo que ilustra bem a dificuldade de separar técnico e social na construção de fatos e artefatos. Bem cedo no processo de construção da linguagem SIMULA, os autores decidiram que ela deveria ser baseada em alguma linguagem bem conhecida na época. Eles escolheram a linguagem ALGOL 60 e justificaram sua escolha por três razões principais (DAHL, 2002, p.80, grifo nosso):

- estrutura de blocos;
- boa segurança para programação; e
- *patriotismo europeu.*

Essa declaração, na qual se explicita a opção por valores nacionalistas, confronta-se com a idéia dos critérios "técnicos" de decisão, ou, dizendo de outra forma, com a existência de um determinismo tecnológico guiando a evolução dos artefatos. Tendemos a crer que o paradigma OO é o que é em função apenas de escolhas

"técnicas" otimizadas. Contudo, se não fosse em decorrência do "patriotismo europeu", a linguagem base utilizada na criação de SIMULA poderia ser outra e o paradigma OO poderia ser diferente do que é hoje.

Mesmo ainda sendo controverso que o paradigma OO é mais aderente ao nosso modo de pensar e que está fadado ao sucesso, ele está estabelecido, e, portanto, diversos atores, indistintamente "técnicos" e "não-técnicos", humanos e não-humanos, estão envolvidos nas traduções que o tem conformado até o momento. Dentre muitos outros, podemos citar o *patriotismo europeu*, a linguagem ALGOL, SIMULA e diversas outras linguagens, métodos de projetação OO, nosso modo de pensar, ferramentas CASE[40] — *Computer Aided System Engineering*, produtividade, programadores, problemas de simulação, conceitos que aprendemos no jardim de infância, institutos e organizações diversas.

Como qualquer outro fato ou artefato, a OO não é uma realidade exclusivamente técnica ou social. É, indissociavelmente, ambas. Quanto mais um engenheiro de software puder afirmar que realizou uma especificação "técnica" de um sistema utilizando a abordagem OO, tanto mais amplo terá sido o esforço social para estabelecê-la, envolvendo o surgimento de grupos, comitês e instituições diversas para a negociação e o acordo sobre o que é "tecnicamente" a OO. Como exemplo de socialização de inúmeras entidades para seu estabelecimento, tomemos apenas a UML — *Unified Modeling Language* —, uma importante linguagem na rede do paradigma OO, estabelecida como padrão para a modelagem de sistemas OO em novembro de 1997. Grady Booch (1998, p.xxi) enumera nada menos que 13 entidades, como HP, IBM, Oracle, MCI, Unisys, no núcleo principal de definição da UML. Sem contar dezenas de colaboradores que influenciaram na definição da UML, como os conhecidos Bertrand Meyer, Edward Yourdon, Eric Gamma, Martin Fowler e Peter Coad. Além disso, não deixa de reconhecer que o resultado foi melhor devido ao imenso *feedback internacional* que obtiveram.

Diferente do que foi feito para o *go to*, não se pode resgatar tão facilmente um porta-voz para o paradigma OO. Ademais, ainda existem controvérsias acerca dos benefícios, vantagens e a "naturalidade" da abordagem OO. Ela não virou, ainda, uma *caixa-preta* que, além de refletir todos os benefícios e vantagens supostamente envolvidos, mantém, por si só, o controle do comportamento de todos os elementos alistados em sua rede sociotécnica. Hatton (1998) defende que quando descobrirmos

---

[39] *Extreme Chaos*, 2001, *Standish Group International, Inc.* Disponível em: <http://www.standishgroup.com/sampleresearch/index.php>. Acesso em: 30 nov. 2004.

[40] As ferramentas CASE cobrem uma vasta gama de diferentes tipos de programas de computador usados para suportar o processo de desenvolvimento e manutenção de softwares (SOMMERVILLE, 2004, p.12).

que OO não é tão apropriado quanto se acredita, teremos que amargar altíssimos custos de manutenção, pois, segundo ele, este custo é ainda mais expressivo na abordagem OO. Além disso, diz que o expressivo ganho de produtividade esperado não se pronunciou ainda. A "forma natural de pensar" ainda não é a escolha principal no desenvolvimento de software a partir do zero, pois apenas 28% dos projetos seguem o paradigma OO, segundo o citado relatório *Extreme Chaos*. Ao listar os fatores mais importantes para o sucesso de um projeto de software, este relatório sequer menciona o paradigma OO. E, por último, a tal revolução industrial do software não deu, até hoje, o menor sinal de atuação. Latour (2000) esclarece que enquanto perduram as controvérsias, mais e mais aliados são arregimentados na busca de estabilização da rede sociotécnica. Por esta razão é que observamos sempre novas linguagens, novos conceitos, novos métodos, novos atores na rede de fatos e artefatos da OO, ou, o que daria no mesmo, na própria engenharia de software em sua trajetória de consolidação como disciplina.

2.4 Enfim, uma metáfora fruto de uma máquina de guerra

É no embalo de um vigoroso discurso militar, de praxe explicitamente moderna, que surgem computadores e engenharia de software. No caso da engenharia de software, corrobora essa afirmação o fato do termo "engenharia de software" ter ganhado força a partir da conferência de 1968 na Organização — militar — do Tratado do Atlântico Norte (OTAN). Outra evidência está nos anos 1980, quando o Departamento de Defesa dos EUA (DoD) patrocinou o desenvolvimento da linguagem ADA, alardeando que o objetivo era "*substituir o idiossincrático ethos 'artístico' que desde muito tempo tem governado o desenvolvimento de software por uma abordagem mais eficiente, efetiva em custo, baseada em um pensamento de engenharia.*"[xxii] (MORRISON[41], 1989 apud ENSMENGER; ASPRAY, 2002, p.156). Outro patrocínio militar ostensivo para a engenharia de software, também nos anos 1980 pelo DoD, foi a constituição do SEI — *Software Engineering Institute* — na Universidade *Carnegie Mellon* (CHRISSIS et al., 2003).

Vimos que o termo engenharia aplicado ao desenvolvimento de software inicialmente não passava de uma metáfora. Porém, temos que às metáforas não cabe apenas o sentido lingüístico, uma vez que a ele se justapõem componentes de materialidade, experimentação e produção. É esta característica que torna as metáforas entidades políticas fortes na articulação dos discursos. Como um grande

modo de representação, as metáforas ajudam na organização de teorias de todos os tipos, deixando de ser algo meramente descritivo para cumprir um papel prescritivo. Quase sempre, nossas representações metafóricas das situações veiculam indicações de respostas e atitudes apropriadas. As metáforas geram um enquadramento que destaca alguns pontos da realidade ao passo que oculta outros. Algumas metáforas entrincheiram-se tanto nos discursos, que guiam e dirigem vários outros sistemas de descrição (EDWARDS, 1997, p.30, 155-158).

É nesse sentido que, na seção 2.3, viu-se a engenharia como metáfora aplicada à construção da engenharia de software. Uma metáfora que escapa da pura retórica para enredar também toda a materialidade dos fatos e artefatos desenvolvidos e cooptados em seu processo histórico. Processo articulado mutuamente com um discurso que, conforme entendido por Edwards (ibid., p.34), trata-se de "*um modo de conhecer, um arcabouço de pressuposições e acordos sobre como a realidade deve ser interpretada e expressa*". Decorrente da praxe modernista do discurso militar donde surge, já é possível, neste ponto, uma primeira compreensão da modernidade na engenharia de software, assunto explorado mais detidamente no próximo capítulo.

[41] MORRISON, D., "Software Crisis", In: Defense, v. 21, n. 2.

Capítulo 3
Engenharia de Software Moderna

Neste capítulo, serão explorados, brevemente, a modernidade e os vínculos que com ela mantém a engenharia de software. O pensamento moderno, com sua visão de mundo mecanicista e tecnocêntrica, é um estilo de pensamento dominante que tutela não só a engenharia de software mas, também, toda produção científica e tecnológica ocidental.

3.1 A modernidade

Embora pareça faltar modéstia ao título desta seção — sugestivo de um tratado sobre a modernidade —, aqui serão trazidas apenas algumas breves idéias introdutórias com o intuito de sensibilização acerca do poder tutelar da modernidade sobre o desenvolvimento científico e tecnológico. Com isso, busca-se sensibilizar também para a "naturalidade" da engenharia de software ter se constituído como uma disciplina moderna.

O processo de modernização foi um poderoso processo de mudança que começou na Europa no século XVI e transformou todos os aspectos da vida naquele continente, como a saída do campo, o abandono dos meios de produção feudais e artesanais, a descrença em milagres, a diminuição do poder religioso e da tradição. Em seu lugar, iniciou-se a construção de um mundo organizado nas cidades, um mundo que incessantemente iria buscar a ciência e a tecnologia, a democracia, a liberdade e o progresso, vinculados ao emergente modo de produção capitalista. No rol de feitos que resultaram do processo de modernização constam, por exemplo, as grandes navegações, a invenção das máquinas e a revolução industrial, uma perspectiva de "auto-referência" da arte ("a arte pela arte") e a sublevação religiosa. (DAHLBOM; MATHIASSEN, 1993, p.9).

No século XVIII, é possível identificar claramente o *projeto da modernidade*. Esse projeto dizia respeito ao esforço intelectual dos pensadores iluministas "*para desenvolver a ciência objetiva, a moralidade e a lei universais e a arte autônoma nos termos da própria lógica interna destas*". Passou a existir a idéia da acumulação, registro e a conseqüente difusão do conhecimento gerado por muitos. Passou-se a crer, também, na possibilidade de domínio científico da natureza para libertar a humanidade da escassez, da necessidade e da arbitrariedade das injunções da

própria natureza. Formas racionais de pensamento e de organização social seriam o antídoto para as irracionalidades do mito, da religião, da superstição, do poder arbitrário. Somente um projeto assim poderia revelar as qualidades "*universais, eternas* e *imutáveis*" do mundo e da humanidade. Abundavam doutrinas de igualdade, liberdade, fé na inteligência humana e razão universal, "*uma boa lei deve ser boa para todos [...], exatamente como uma proposição verdadeira é verdadeira para todos.*" Uma visão bastante otimista sobre o progresso, a compreensão do mundo e do eu, o controle das forças naturais, a justiça das instituições e, até mesmo, a felicidade da humanidade. Otimismo que sucumbiria frente aos campos de concentração, às guerras e à possibilidade de aniquilação nuclear do século XX e, como não lembrar, ao ameaçador aquecimento global, efeito colateral do "progresso moderno", neste princípio de século XXI. (HARVEY, 2006, p.23).

Para dar conta do imenso desafio do projeto da modernidade, foram desenvolvidas, na Europa, as ciências naturais, que elaboram uma explícita distinção entre o mundo e as suas representações. Sob a constatação de que nossa percepção de senso comum nem sempre coincide com o próprio mundo — por exemplo, parece que o Sol gira em torno da Terra, o que não ocorre; parece que os objetos têm cores, mas isso é só um efeito da luz que refletem —, a distinção entre o mundo e suas representações foi criada em resposta à necessidade de purificação das idéias e conceitos de quaisquer contaminações subjetivas, para que, "de fato", pudessem representar o mundo "real". Para o moderno, agir sobre as representações seria a condição para uma atuação racional, científica, evitando-se os enganos da percepção ingênua (do senso comum) do mundo. Nas representações, deveriam ser descartadas as impurezas subjetivas que não existiriam no mundo "real", dado serem fruto de percepções enganosas. (DAHLBOM; MATHIASSEN, 1993, p.10).

Ninguém promoveu mais a idéia do conhecimento como uma representação na mente de um mundo fora, exterior, do que Descartes. Desde o século XVII, ele identificava o pensamento com uma manipulação racional de símbolos por meio de regras. Aqui, um pequeno parêntese ajuda a puxar um dos fios da meada que atam a engenharia de software ao modernismo. É que muitos vêem o próprio surgimento do computador — donde não se dissocia a engenharia de software — como a coroação dessa longa tradição de pensamento, muito fortalecida por Descartes, como denota o rótulo de "*cérebros* eletrônicos" atribuído aos primeiros computadores eletrônicos quando surgiram (DAHLBOM; MATHIASSEN, 1993, p.7). É possível entender o vigor da metáfora da mente apartada do corpo como um computador, pois, quando passam a existir, os computadores tornam-se suportes ostensivos dessa metáfora.

> [O COMPUTADOR como metáfora ...] retorna à metáfora Cartesiana da mente como uma máquina matemática [uma máquina de cálculo, de computação], mas com uma estrutura concreta [os próprios computadores eletrônicos] que enriquece imensamente o conceito Cartesiano[xxiii] (EDWARDS, 1997, p.162).

A pressuposição de que o entendimento, o conhecimento e a experiência humana podem ser, de alguma forma, extraídos de seus contextos para serem codificados em uma máquina foi muito robustecida na modernidade. Mas, trata-se de um legado do *pensamento tecnológico*, promovido desde a lógica aristotélica, passando pelo racionalismo cartesiano e o princípio da causalidade de Leibiniz, estando presente no projeto enciclopedista do iluminismo e na busca do positivismo lógico por proposições verificáveis e isentas de valores. (COYNE, 1995, p.80).

Finalizando o parêntese e retornando à questão da separação explícita entre o mundo e suas representações, segundo Descartes, teríamos que garantir que nossas idéias fossem claras e exatas antes de dependermos delas para produzir um retrato acurado do mundo real. Isso só poderia ser alcançado através da aplicação de um método racional que viabilizasse a explicitação das regras, ou leis, do objeto sob análise em uma representação, esta última formalizada através de uma linguagem apropriada. As idéias de representação e formalização habitam o âmago da disciplina de ciência da computação, com presença muito forte na própria engenharia de software, que herda a postura modernista de representações "libertas" de impurezas subjetivas, assumindo assim um viés tecnocênctrico (DAHLBOM; MATHIASSEN, 1993, p.12).

A mudança radical ocorrida com o projeto da modernidade, tornou-se referida como a Revolução Científica do século XVII. Passou a existir uma filosofia que difundiu vigorosamente a visão de mundo mecanicista, uma visão que pressupõe existir ordem *a priori* subjacente ao mundo que apenas aparentemente seria desordenado. O objetivo científico passou a ser descobrir os *mecanismos* — as leis — da natureza para, obedecendo-os, poder controlá-la. O mecanicismo compartilha a certeza da matemática, à qual está vinculado. As ciências, por isso dito exatas, também devem buscar essa certeza em suas representações do mundo. A nova ciência das máquinas foi transformada em uma ciência mecanicista, mais universal, expressa através da linguagem analítica do cálculo e da álgebra. Os *Princípios de Newton*[42], como o grande expoente deste movimento, tornaram-se a pedra de toque para todas as ciências. (MAHONEY, 2002, p.47).

---

[42] No ano de 1687, em Londres, Isaac Newton publicou os "*Philosophiae Naturalis Principia Mathematica*", uma obra de três volumes considerada por muitos uma das mais influentes obras cientificas já publicada. Trata da mecânica clássica e da gravitação universal.

Alguns pontos marcam bem o modernismo, dentre eles as crenças: (i) no progresso linear, teleológico; (ii) em verdades intrínsecas à ordem das coisas; (iii) em "metanarrativas" — interpretações teóricas pretensamente universais, capazes de determinar, *a priori*, o fenômeno que representam —; (iv) na apreensão metafísica e positivista do mundo — cujas regras/leis fundamentais preexistentes são passíveis de explicitação de forma racional, não de uma forma empírica, ou, muito menos, revelada —. A modernidade envolve uma ruptura implacável com todas as condições históricas que a precedem e é bem caracterizada por seu interminável e inerente processo de fragmentação, simplificação e análise de seus objetos em busca da descoberta dos elementos "*eternos e imutáveis*". O modernismo sempre está comprometido com a descoberta do "*caráter essencial do acidental*". (HARVEY, 2006, p.19-22).

O projeto da modernidade foi muito proficiente em descrever, prever e conformar diversos segmentos da realidade. Por isso, tornou-se onipresente. Como ilustração, apenas para insinuar o reconhecimento da proficuidade do modernismo, segue o exemplo da mecânica newtoniana, com suas três leis básicas e a (lei da) gravitação universal que podem ser tomadas como as ilustres representantes dos postulados modernos desde o século XVII (DAHLBOM; MATHIASSEN, 1993, p.14). Usando o cálculo diferencial e integral como linguagem, as representações do mundo que Isaac Newton desenvolveu permitiram vasta descrição, compreensão, previsão, ordenamento e controle dos fenômenos do movimento dos corpos, desde minúsculos seixos até planetas inteiros. Lorde Keynes, o economista, que biografou Newton, explicita uma noção metafísica na qual crê-se que uma realidade externa, essencial, fora apreendida racionalmente:

> Este homem solitário [Isaac Newton] desenvolveu poderes excepcionais de introspecção contínua e concentrada. [...] O poder de manter continuamente na sua mente um *problema puramente mental*, ..., durante horas, dias e semanas, até que ele lhe entregasse o seu segredo.[43]

Trata-se de um exemplo que serve para denotar a presença, no discurso moderno, do esforço em se "descobrir" a ordem essencial, desconhecida mas preexistente e permanente no mundo "real", para poder controlá-lo. O suposto caos existente decorre da incompreensão, posto que, para o moderno, o mundo "lá fora" é ordenado, estável e imutável, ou, pelo menos, com mutabilidade passível de se conhecer. O inevitável progresso levará à descoberta do "caráter essencial do acidental", bastando para isso ter à disposição método e linguagem adequados e o tempo para que surjam as "descobertas" e "descrições" da ordem ainda não

[43] TIPLER, P.A., 1990, *Física*, Vol. 1a, 2ª ed., Rio de Janeiro, Editora Guanabara, p.102, grifo nosso.

apreendida, que poderá ser registrada em teorias e modelos "universais" que viabilizarão o exercício de controle sobre o mundo.

O discurso moderno exerce um apelo muito poderoso e tornou-se dominante na racionalidade científica ocidental, que passou a valorizar o método como diretriz para a verdade. Esse conceito entranhou-se profundamente na matemática e nas ciências naturais, que, por sua vez, passaram a influenciar no estabelecimento daquilo que deve contar, ou não, como conhecimento no ocidente. Até políticas e práticas nas sociedades industriais têm sido profundamente influenciadas pelo conceito cartesiano de método e demais postulados modernos (HIRSCHHEIM; HEINZ; LYYTINEN, 1995, p.21). A modernidade tornou-se um poderoso modo de vida social e organizacional a partir do século XVII, determinando nosso modo de desenvolvimento de conhecimento e tecnologia, com manifesta intenção de habilitar um controle mais eficiente sobre processos cada vez mais complexos, como produção, transportes, comunicação, etc. Trata-se de uma clara busca de se controlar a natureza e a vida social, razão pela qual a modernidade pode ser descrita também como a *revolução do controle* (HANSETH; BRAA, 2001, p.45).

3.2 Vínculos da engenharia de software com a modernidade

O brutal desenvolvimento das ciências naturais, consubstanciado por grandes feitos técnicos e tecnológicos — desde a descoberta da América até a última viagem espacial — conferiram à modernidade a reputação de via segura para o progresso. Como efeito colateral, a ciência tornou-se um modo de prática social tão especializado que deixamos de reconhecê-la como uma prática (COYNE, 1995, p.39), colocando-a num pedestal pré-discursivo como fiel depositária da verdade. Ao invés de ser vista (e criticada) como uma construção humana, passou a ser algo a que se deve prestar culto, respaldando o que pode ou não contar como conhecimento verdadeiro no ocidente.

> Graças ao poder das forças sociais reificadoras que estavam por trás do sucesso do positivismo e do neopositivismo, uma visão extremamente unilateral do desenvolvimento "autônomo" da ciência e da tecnologia se tornou, não o "senso comum", mas o mistificador *lugar-comum* de nossa época. Seus defensores vão desde filósofos ganhadores do prêmio Nobel, como Bertrand Russel, até sábios midiáticos dedicados à divulgação de vôos lunares religiosamente acompanhados; desde os escritores de ficção científica até os bem recompensados propagandistas do complexo militar-industrial. A aceitação acrítica dessa visão foi particularmente favorecida pelo pós-guerra, período marcado pelo consenso e por sua irmã gêmea, a ideologia do "fim da ideologia". Tornou-se moda falar sobre a "ascensão da *sociedade tecnológica*, um *tipo totalmente novo de sociedade humana*, na qual a *ciência e a tecnologia ditam* as formas dominantes de pensamento e moldam cada vez mais quase todos os aspectos de nossa vida cotidiana". (MÉSZÁROS, 2004, p.261-262, grifos do autor).

Bruce Seely (2002, p.85), historiador da *Michigan Technological University*, diz que existe a crença de que um campo terá mais valor se for realmente aceito como engenharia, porque assim seria dotado de um certo *pedigree* científico, supostamente compartilhando da verdade detida pela ciência. Por isso, e por tudo mais visto até aqui, a jovem disciplina da engenharia de software só poderia estar impregnada pela moderna racionalidade científica ocidental. Para desfrutar da inelutável garantia do progresso, bastaria à engenharia de software escorar-se no círculo vicioso de afirmação e negação do modernismo (COYNE, 1995, p.13) com a aplicação reflexiva do conhecimento que produz, até dominar e controlar, um dia, as forças que dificultam e tornam complexa a atividade de desenvolver e manter sistemas de software[44].

Um exemplo sugestivo deste "inevitável progresso", poderia ser o desenvolvimento das linguagens de programação de alto nível, assim que houve hardware com suficiente potência para torná-las operacionais no final dos anos 1950. Com as linguagens de alto nível, parecia que o "caráter essencial do acidental" pôde ser descoberto e controlado, de sorte que os esforços com a programação tornaram-se menos custosos, resolvendo alguns dos problemas que prejudicavam a produtividade dos programadores. Como dizia John Backus, responsável pela especificação da linguagem FORTRAN em 1954 e sua primeira implementação em 1957, na IBM (ENSMENGER; ASPRAY, 2002, p.154):

> Programação nos anos 1950 era magia negra, uma matéria privada e secreta, ... cada problema requeria um início único a partir de uma posição qualquer, e o sucesso dependia primariamente da técnica e da inventividade do programador.[xxiv]

Sob uma visão modernista, diferente da idéia de construção sociotécnica (seção 2.3), é como se desenvolver software pudesse ser naturalmente engenharia, algo genialmente "descoberto" por J. Presper Eckert e proferido em sua já citada (pág. 23) apresentação na *Fall Eastern Joint Computer Conference* em 1965.

[44] Em 1956, o matemático John McCarthy, concebedor da linguagem funcional LISP, pleiteava patrocínio junto a Fundação Rokefeller para a realização da primeira, das hoje famosas, conferências no Dartmouth College sobre inteligência artificial. Ele deixava claro sua pressuposição de progresso inevitável ao afirmar: "*o estudo parte da base conjectural de que todo aspecto do aprendizado ou qualquer outra característica da inteligência pode em princípio ser tão precisamente descrito que uma máquina poderá ser construída para simulá-lo.*" (DAHLBOM; MATHIASSEN, 1993, p.3).

3.2.1 A busca do *pedigree* científico

Nos últimos 50 anos, não só a engenharia de software, mas a própria ciência da computação, deixou de ser uma disciplina cujo número de participantes se podia contar nos dedos das mãos para se tornar uma extensa rede de participantes e pesquisadores na academia e na indústria. Grande parte deste crescimento deriva-se da busca por uma chancela científica. (MAHONEY, 2002, p.27).

A busca de cientificidade pela engenharia de software, além de ser uma tentativa de compartilhar a "verdade detida pela ciência ocidental" e se estabelecer como uma disciplina autônoma, decorre também das próprias circunstâncias de surgimento dos computadores e de seus programas no meio científico, bem como da finalidade a que se destinavam. Howard Aiken, físico da universidade de *Harvard*, e responsável pelo projeto de um dos primeiros computadores[45], portanto um profundo conhecedor da tecnologia da computação da época, declarou equivocadamente, em 1956:

> Se algum dia ocorrer da lógica básica de uma máquina projetada para solucionar numericamente equações diferenciais coincidir com a lógica de uma máquina cujo propósito é gerar faturas para uma loja de departamentos, eu iria julgar isto a mais surpreendente coincidência que já tenha encontrado[xxv] (DAHLBOM; MATHIASSEN, 1993, p.3).

Naquela época, parecia natural enxergar a computação e sua programação como claros domínios da matemática e ciência, pois os computadores eram extensivamente usados apenas para cálculos de engenharia e de pesquisas científicas. Somente de forma muito restrita eram usados para o processamento de dados em atividades de negócios. Por exemplo, o problema da escassez de profissionais de computação chegou a ser visto como um problema estrito da própria comunidade da matemática.

> Que aquela conferência [*The First Conference on Training Personnel for the Computing Machine Field*] foi oportuna nem precisa ser enfatizado, pois a falta de candidatos competentes para posições no campo da computação tem se tornado crescentemente crítica à medida que cresce a demanda por computadores. Não tão amplamente reconhecido é o fato de que esse problema é de *importância para toda a comunidade da matemática.*[xxvi] (HOHN, 1955, grifo nosso).

A exploração da linhagem histórica de algoritmos importantes para a computação nos leva, com freqüência, a algum matemático originalmente envolvido com a concepção de uma solução para um problema similar ao do algoritmo (HASHAGEN; KEIL-SLAWIK; NORBERG, 2002, p.3). De fato, o conceito de software

como um objeto matemático e como produto de engenharia teve livre curso nos primeiros 50 anos da história do desenvolvimento de software (LEVESON, 1997). Todas as engenharias, e não seria diferente com a engenharia de software, tomam a matemática como sua linguagem natural (TOMAYKO, 2002, p.66) em busca do lastro científico que as garanta valor, respeito e reconhecimento como disciplina autônoma.

A base científica para o software, hoje chamada de "fundamentos da teoria da computação" ou "teoria da ciência da computação", foi estruturada praticamente entre os anos 1955 e 1975 por pesquisadores e praticantes de vários campos, como o dos autômatos, das gramáticas formais e da complexidade computacional, todos erguidos sobre sólida base matemática. Por volta de meados dos anos 1970, o campo da "teoria da ciência da computação", por servir de recurso para várias outras ciências, começou, ele próprio, a ser reconhecido como um ramo da matemática. (MAHONEY, 2002, p.29).

É bem verdade que alguns tipos de software têm se estabelecido sobre firmes bases teóricas matemáticas com sucesso, como os compiladores e os gerenciadores de bancos de dados relacionais, e espera-se ainda que outros mais alcancem o mesmo *status*, como, por exemplo, os sistemas operacionais (HAIGH, 2002, p.62). A própria engenharia de software dispõe de exemplos com forte lastro na matemática, como a técnica *cleanroom*[46] desenvolvida por Harlan Mills, na IBM. Sua base científica, sem novidade alguma, é matemática, e foi estendida para um argumento aplicável à programação estruturada[47]. É possível identificar uma linha de princípios científicos matemáticos desde a teoria da computabilidade de Turing nos anos 1930, passando pela prova formal das três únicas estruturas de programação necessárias — seqüência, seleção, iteração — de Böhm e Jacopini (vide nota 31) nos anos 1960, até chegar à idéia de Mills sobre se derivar a prova da correção dos programas a partir da prova de cada um de seus módulos, considerados funções matemáticas. (TOMAYKO, 2002, p.72).

Ao longo desses anos, os tipos de conhecimentos teóricos usados na busca de cientificidade para o software e para a engenharia de software têm se transformado, mas nunca se desvincularam da matemática. Hoare, Charles Anthony Richard Hoare, importante nome na ciência da computação[48], em uma aula inaugural no

---

[45] Aiken dirigiu a construção do ASCC — *Automatic Sequence Controled Calculator* — uma máquina eletromecânica especializada para a realização de cálculos numéricos científicos, inaugurada em 1944.

[46] A técnica *cleanroom* explicita o uso do controle estatístico de qualidade, baseando-se na verificação de especificações através de provas matemáticas de correção (PRESSMAN, 2001, p.703).

[47] LIGER, R.C.; MILLS, H.D.; WITT, B.I., 1979, *Structured Programming: Theory and Practice.* Reading, Mass.

[48] Dentre outras contribuições, como as provas formais de correção de programas, Hoare concebeu, no início dos anos 1960, o mais rápido algoritmo de ordenação de uso geral conhecido — o *quicksort*.

departamento de computação da Universidade de *Oxford*, em 1985, declarou (MAHONEY, 2002, p.36):

> Nossos princípios podem ser sumariados em quatro tópicos.
> (1) *Computadores são máquinas matemáticas.* Todo aspecto de seu comportamento pode ser definido com precisão matemática, e cada detalhe pode ser deduzido desta definição com certeza matemática decorrente das leis da lógica pura.
> (2) *Programas de computadores são expressões matemáticas.* Eles descrevem com precisão sem precedentes e no menor intervalo de tempo detalhes do comportamento, pretendido ou não, do computador onde estão sendo executados.
> (3) *Uma linguagem de programação é uma teoria.* Ela inclui conceitos, notações, definições, axiomas e teoremas, os quais ajudam um programador a desenvolver um programa que cumpra suas especificações, e a provar que isto ocorre.
> (4) *Programação é uma atividade matemática.* Como outros ramos da matemática aplicada e da engenharia, a prática bem sucedida da programação requer a aplicação meticulosa e determinada de tradicionais métodos matemáticos de entendimento, cálculo e prova.
> Esses são *princípios filosóficos e morais gerais*, e eu os tenho por auto-evidentes, pois todas as evidências reais são contrárias a eles. Nada é realmente como eu descrevi, nem computadores, nem programas, nem linguagens de programação, nem mesmo os programadores[xxvii].

No fundo, esconde-se por detrás da declaração de Hoare, se não a crença, pelo menos o desejo do progresso teleológico, herdado da racionalidade científica moderna, que um dia possa garantir esse estado à computação e à programação, mesmo reconhecendo que na prática a ligação com a matemática não é evidente.

O lastro do software na matemática foi ficando cada vez menos aparente à medida que os computadores alcançaram praticamente todas as áreas da atividade humana. No final dos anos 1940 e início dos anos 1950, os computadores eram, de fato, instrumentos matemáticos, o que justificava que os programadores devessem ter treinamento formal em matemática. Já a partir de meados dos anos 1950, com o início da aplicação dos computadores no mundo dos negócios, o treinamento de programadores focado na lógica formal e na análise numérica tornou-se cada vez menos requerido. Ao contrário, começou-se a perceber, já no início dos anos 1960, que a habilidade de programação tem pouca relação com o treinamento formal em matemática. (ENSMENGER; ASPRAY, 2002, p.143).

> Como uma ferramenta de projetação, a semântica matemática ainda está muito longe do objetivo de correção das anomalias que resultam em erros nas linguagens de programação reais. Se computadores e programas são "objetos inerentemente matemáticos" a matemática de computadores e programas de coisas que dizem respeito à prática real tem até o momento provado-se ilusória.[xxviii] (MAHONEY, 2002, p.36).

Com seu lastro na matemática enfraquecido na prática, o que a diferencia das demais engenharias, a engenharia de software, apenas por ostentar o título de engenharia, já garantiria seu *pedigree* científico devido a idéia, ainda aceita por muitos, de que a engenharia é uma aplicação da ciência. A força dessa idéia — de

subordinação da engenharia à ciência — em grande parte surgiu da influência do relatório de Vannevar Bush para o presidente Roosevelt, em 1945, intitulado "*Science The Endless Frontier*"[49]. Esse documento determinou a política científica norte-americana no período pós guerra, exatamente o período em que surgiu a computação eletrônica. Bush conseguiu tornar axiomática a proposição de que a tecnologia emerge da ciência básica. O racionalismo científico suporta essa visão de que a tecnologia é ciência aplicada (COYNE, 1995, p.27). Certamente, contribuíram para a tomada de força dessa idéia os bilionários projetos militares — como o SAGE, por exemplo — que viabilizaram, em curto espaço de tempo, muitas inovações científicas e miríades de artefatos técnicos e tecnológicos associados. Isso fortaleceu a idéia de que um campo de engenharia detém o valor intrínseco herdado da ciência que lhe dá suporte. O respeito dos círculos acadêmicos aumenta com o aumento da base científica de uma disciplina (SEELY, 2002, p.84-85).[50]

Como a ciência da computação carrega todas as marcas de uma autêntica disciplina científica, há quem a veja como a ciência que ainda proverá os "princípios fundamentais" de base para a engenharia de software, tal qual a física o faz para a engenharia civil. Nesta linha, se a ciência da computação é um estudo sobre a manipulação simbólica, o comportamento algorítmico constitui um fenômeno sob investigação científica. Como outras engenharias, a engenharia de software também seria uma disciplina de projetação (*design*) perfeitamente dotada de um componente científico. (COYNE, 1995, p.207-208). Figuras importantes para a ciência da computação defendiam essa idéia já no início dos anos 1960. Por exemplo, John McCarthy tanto acreditava na possibilidade do suporte matemático da "teoria da computação" à engenharia de software — ou à programação, termo utilizado na época — que afirmou: "[no futuro] *ninguém pagará por um programa de computador até que esteja provado* [matematicamente] *que ele atende suas especificações.*" (MAHONEY, 2002, p.35). Para McCarthy era razoável:

> [...] esperar que o relacionamento entre a computação e a lógica matemática será tão frutífera no próximo século como foi o relacionamento entre a análise [, o cálculo] e a física no século passado. O desenvolvimento deste relacionamento demanda uma preocupação tanto com as aplicações como com a elegância matemática.[xxix] (McCARTHY[51], 1963 apud MAHONEY, 2002, p.35).

---

[49] BUSH, V., 1945, *Science: The Endless Frontier: A Report to the President on a Program for Postwar Scientific Research*. Washington, DC: U.S. Government Printing Office.

[50] O argumento aqui não deve soar como a defesa das outras engenharias serem mais "científicas", ou mais "modernas", do que a engenharia de software. Sob o ponto de vista dos Estudos CTS, não corroborando a idéia da engenharia ser ciência aplicada, a engenharia de software é tão engenharia quanto as demais. A questão em foco é que a engenharia de software pôde valer-se do termo "engenharia" como metáfora, beneficiando-se da estabilidade preexistente de outras disciplinas de engenharia.

[51] McCARTHY, J., 1963, "A Basis for a Mathematical Theory of Computation". In: BRAFFORD, P.; HIRSCHBERG, D. (eds), *Computer Programming and Formal Systems*, Amsterdam, p.33-69.

Na literatura da engenharia de software resgata-se com facilidade a busca de respaldo científico. Por exemplo, além da correlação aparentemente natural entre os domínios da programação e da matemática nos primeiros momentos da história da computação, a engenharia de software busca o alistamento da ciência e seus métodos para alcançar "objetivamente", com a guarida da teoria dos sistemas, soluções válidas para os problemas que enfrenta. Ao seguir a teoria dos sistemas, a engenharia de software explicita sua busca modernista pela razão instrumental e o desejo de controlar pessoas e resultados. Até os anos 1960, o tema predominante do movimento relacionado aos métodos de projetação (*design-methods*) foi a identificação e desenvolvimento de métodos, modelos e teorias, de modo que esse movimento buscou colocar a projetação em um pedestal científico. (COYNE, 1995, p.11, 209, 219). Não só os da engenharia de software, mas, "*desde o final da Segunda Grande Guerra, os educadores das disciplinas de engenharia têm seguido uma abordagem analítica e científica* [...]".[xxx] (SEELY, 2002, p.86-87).

O pouco tempo que o campo teve para avançar advogaria em prol da idéia da engenharia de software ser aplicação de alguma ciência ainda não totalmente estabelecida, o que serviria de explicação apriorística para seu "estágio incompleto de engenharia".

> Podemos mencionar brevemente algumas vezes onde a ciência é auto-conscientemente aplicada à engenharia de software. Alguns têm argumentado que a ciência subjacente ainda não está completa o suficiente para verdadeiramente suportar a engenharia de software da mesma forma que ocorreu com as engenharias mais antigas. Neste ponto concordamos, mas o conhecimento de engenharia é ganho primeiro na prática, só então mais tarde por explicações científicas para as práticas bem sucedidas. O fato de que não existem explicações científicas para toda engenharia de software se encaixa bem para um campo nascente. Existem, todavia, exemplos onde a ciência tem surgido como suporte à engenharia de software. Esses exemplos são todos vindos da matemática, que tem se tornado a linguagem da engenharia de software, bem como sua fundação.[xxxi] (TOMAYKO, 2002, p.71-72).

3.2.2 A modernidade na engenharia de software

A metáfora da engenharia aplicada ao desenvolvimento de software, *per si*, não consegue responder pelo *status* tecnocêntrico da engenharia de software. Este *status* foi conformado pelo estilo de pensamento modernista que tutela em grande parte o desenvolvimento científico e tecnológico de todas as engenharias, fazendo-as buscar a eliminação das "contaminações subjetivas" ou "não-técnicas" de suas representações. Como metáfora, mas entendendo de forma bem distinta, a

engenharia daria chance a uma construção diferente para a engenharia de software, como interpreta Endres.

> Engenharia não pode ser vista como o contraste, o oposto, de arte. Quanto mais arte houver na projetação, melhor. Quanto mais audaciosos os projetos (*design*), maior o sucesso (ou a falha). [...] Pode ser que os engenheiros de língua inglesa derivem a designação de sua profissão da palavra mecanismo (por exemplo, as máquinas de guerra) [bem o caso tratado no capítulo 2, com os BIG'L]. Em contrapartida, nós alemães preferimos (ainda) usar a palavra francesa "ingénieur" como importada há mais de 300 anos atrás. Ela é estreitamente relacionada com a palavra para *ingenious*, em francês (*ingénieus*) para a qual Petit Larousse dá a seguinte definição: cheio de espírito, inventividade e habilidade.[xxxii] (ENDRES, 2002, p.78).

No entanto, uma metáfora é algo que reside no interstício entre o ser e o não ser (COYNE, 1995, p.298). Primariamente, uma metáfora é uma ferramenta discursiva que serve para manter viva alguma relação. Sua eficácia dependerá de sua adoção e uso em um discurso (ibid., p.245). Logo, ocorre que, no discurso estabelecido, a metáfora da engenharia aplicada ao desenvolvimento de software vem conformando uma disciplina da modernidade, portanto com uma visão de mundo mecanicista e tecnocêntrica.

Computadores, programas e modelos de projeto (*design*) de sistemas são artefatos e, portanto, qualquer ciência a seu respeito deveria ser uma "ciência do artificial"[52]. Todavia, de todas as abordagens teóricas, a que mais tem contribuído para moldar o projeto e a pesquisa de sistemas de informações, portanto, também a engenharia de software, é o hegemônico positivismo. Os positivistas vêem seu trabalho como uma continuação da longa tradição, de mais de 300 anos, do Iluminismo europeu através da modernidade, na qual a razão remove costumes e superstições e aumenta nossa posse da "Verdade" e, consequentemente, nosso poder em controlar o mundo e seus destinos. Embora não se trate de uma ciência clássica, o projeto e a pesquisa sobre os sistemas de informações e a engenharia de software são tradicionalmente construídos em torno da metafísica realista da ciência clássica, algo portanto que busca um conhecimento racional — não empírico e tampouco revelado — da essência última das coisas, uma realidade externa, separada do observador. (ARNOLD, 2003, p.227).

A maioria de nós, se pensarmos na engenharia de software e na literatura comum a seu respeito, poderia concordar que a afirmação de David Harvey, a seguir, continuaria fazendo sentido se a considerássemos no âmbito da engenharia de software e até mesmo se substituíssemos o termo "modernismo" por "engenharia de software":

---

[52] Termo definido por: SIMON, Herbert, 1969, *The Science of the Artificial*. Cambridge, Mass.

> Geralmente percebido como positivista, tecnocêntrico e racionalista, o modernismo [a engenharia de software] tem sido identificado com a crença no progresso linear, nas verdades absolutas, no planejamento racional de ordens sociais ideais, e com a padronização do conhecimento e da produção (HARVEY, 2006, p.19).

Elementos da visão moderna de mundo são facilmente resgatados na literatura da engenharia de software. Os chamados métodos estruturados (ou *hard methods*) de projeto de sistemas, por exemplo, se auto-identificam como objetivos e científicos (ao invés de humanistas) e já trazem cravado em sua denominação o valor devotado ao método (ARNOLD, 2003, p. 228). Existe uma forte crença no poder do projetista deter todo o controle do processo de projetação porque utiliza um método, que, supostamente, conforma e dá legitimidade científica às atividades. Baseando-se na teoria dos sistemas, os métodos arvoram para si competências prescritivas, propondo procedimentos que, uma vez seguidos, seriam capazes de produzir o resultado particular esperado. Subjaz a visão de que a projetação é a conversão de uma situação em uma outra mais desejável, uma postura na qual se crê que objetivos e necessidades podem ser identificados e caracterizados, emergindo do ato de projetação e guiando todo o processo (COYNE, 1995, p.210, 238-243). Trata-se de uma abordagem que vê o projeto (*design*) bem sucedido em termos instrumentais de efetividade e eficiência, baseando-se em premissas segundo as quais, por exemplo, sistemas de informações são sistemas fechados e que existem por si mesmos dentro dos limites organizacionais. E ainda, que as organizações são seqüências de processos de negócio e fluxos de dados, povoadas por indivíduos inerentemente racionais. (ARNOLD, 2003, p. 228).

Vejamos a *modelagem de informação*[53], ferramenta chave com presença muito forte nos ciclos de desenvolvimento de sistemas. Ela denota "*uma posição realista ingênua*", ao pressupor que é bastante a existência de um método adequado para que o mundo "real", "objetivo", o "lá fora", possa ser descoberto. É clara a pressuposição subjacente de existir um mundo ordenado *a priori*, bastando ao engenheiro de software descobrir / capturar os requisitos preexistentes, formalizar uma especificação — o modelo ou representação do problema — e desenvolver o sistema desejado a partir dela. A maioria das abordagens tende a considerar que é possível, de antemão, definir os requisitos e que eles se manterão estáveis ao longo do desenvolvimento, processo no qual deixa-se de considerar o mundo "real" em favor da atuação exclusiva sobre as representações. (HANSETH; MONTEIRO, 1998, p. 83;141).

> A maioria das (mas não necessariamente todas) técnicas de modelagem focam em funções, dados ou objetos como os mais básicos elementos. As pressuposições implícitas e/ou explícitas são que: (1) esses elementos

[53] Uma introdução à *modelagem da informação* pode ser obtida em COAD e YOURDON (1990).

> básicos existem no mundo (realismo); e (2) existe um conjunto de coisas passíveis de definição objetiva cuja própria definição independe da percepção do desenvolvedor (objetivismo). A primeira pressuposição implica que o trabalho do desenvolvedor seria o de "encontrar" aqueles objetos como se eles fossem um tesouro afundado em um navio carregado pelas ondas até o litoral esperando para ser apanhado pelo primeiro que passasse. A segunda pressuposição implica que quaisquer dois desenvolvedores deveriam produzir o mesmo modelo (pois eles iriam achar o mesmo tesouro) e se houver diferenças elas são solucionáveis. Se dois desenvolvedores vêem as coisas de modo diferente, a pressuposição (2) sugere que um não está vendo a aplicação tão claro quanto o outro, ou que um desenvolvedor simplesmente não é tão bom quanto o outro.[xxxiii] (HIRSCHHEIM; HEINZ; LYYTINEN, 1995, p.xi-xii).

Um exemplo dessa posição realista pode ser visto na reflexão de Tom DeMarco acerca de suas importantes idéias sobre a *análise estruturada*:

> É uma importante *verdade*: quando você está atacando a complexidade através de particionamento, quanto mais tênues as interfaces melhor o particionamento — se as interfaces ainda estão grosseiras, exageradas, volte e particione novamente, buscando os contornos *naturais do domínio* [xxxiv] (DeMARCO, 2002, p.526, grifo nosso).

A engenharia de software devota grande importância à busca por padrões e modelos, influenciada pela própria busca da ciência moderna por modelos, padrões, fatos e leis universais (HANSETH; MONTEIRO, 1998, p.133). Ser científico é ser universal, valer em qualquer lugar, facultar a qualquer um a repetição dos fenômenos observados, desde que respeitado o método. Assim, modelos e métodos universais adequados garantiriam, *per si*, o sucesso dos projetos. Podemos ver esse ponto de vista também nas palavras de DeMarco (2002, p.524, grifos nossos).

> Eles [seus leitores] estavam convencidos com [a validade de] o método porque ele dava uma confortável sensação de completute; ele aparecia para eles como *A Resposta para todos os problemas. Quando não conseguiam resolver seus problemas eles culpavam a si próprios* e tentavam com maior rigor ainda. Hoje acredito que meu livro de 1975[54] foi excessivamente persuasivo e que muitos em nossa indústria [de software] foram simplesmente seduzidos por ele. Isto em parte resultou de meu irrestrito entusiasmo *com um método que funcionou soberbamente* para mim (*num domínio limitado*) [...][xxxv].

O fato do projeto não estar de acordo com a teoria dos sistemas — teoria subjacente aos métodos estruturados — recai pejorativamente sobre o próprio projeto, supostamente eivado de defeitos. Quase nunca se abre a discussão sobre a possibilidade de inadequação da própria teoria dos sistemas ou de suas aplicações. (COYNE, 1995, p.225).

Outro postulado modernista, o imperativo da representação e formalização, pode ser visto explicitamente nas inúmeras linguagens, métodos e ferramentas, como, por exemplo, na UML — *Unified Modeling Language* — e seus inúmeros diagramas.

---

[54] DeMARCO, T., 1975, *Structured Analysis and System Specification*. Prentice Hall.

A perspectiva do controle, algo central no pensamento modernista, jaz na disciplina dos *processos de software*[55], herdeiros das idéias de burocracia, que considera possível prever, predeterminar e explicitar papéis, além de estabelecer regras que garantam o comportamento esperado e a coordenação central necessária. Normalmente parte-se do pressuposto, nos projetos de software, de que é possível saber antecipadamente o que deve ser feito, existindo pouca incerteza acerca das tarefas (DAHLBOM; MATHIASSEN, 1993, p.16).

A área dos *processos de software* surgiu nos anos 1980 como uma disciplina autônoma, inspirada no movimento da qualidade total, cuja premissa básica é a existência de uma correlação entre a qualidade do processo e a qualidade do software produzido (FUGGETTA, 2000). A origem da disciplina dos processos de software, no entanto, remonta aos problemas de gerenciamento e controle da força de trabalho que, no final dos anos 1960, passou a representar o maior fator de custo das instalações de computação. O desenvolvimento de novas metodologias para o gerenciamento de projetos e para o controle do processo de desenvolvimento de software foi uma tentativa de resposta ao problema de disciplinar e controlar a força de trabalho dos programadores, vista como recalcitrante. Com profunda inspiração modernista-taylorista, buscou-se fragmentar, rotinizar e mecanizar o trabalho, travestindo este esforço em uma imagem de progresso e busca de eficiência. (ENSMENGER; ASPRAY, 2002, p.151).

> Programadores, analistas de sistemas e outros profissionais da indústria de software estão vivenciando esforços para quebrar, simplificar, rotinizar e padronizar seu próprio trabalho de forma que ele, também, possa ser feito por máquinas ao invés de pessoas. ... Esforços elaborados estão ocorrendo para desenvolver meios de gradualmente eliminar os programadores, ou pelo menos reduzir seus níveis médios de habilidades, treinamento requerido, experiência, e coisas assim. ... A maioria daqueles que chamamos programadores, em breve, serão largamente relegados a papéis subsidiários e subordinados no processo de produção. ... Enquanto uns poucos deles vão sentar-se do lado dos executivos, aconselhando e provendo assessoria especializada, a maioria simplesmente executará o que outros os atribuírem.[xxxvi] (KRAFT[56], 1977 apud ENSMENGER; ASPRAY, 2002, p.152).

De forma explícita ou não, o taylorismo continua a influir na engenharia de software. Watts Humphrey, principal concebedor do *Capability Maturity Model* (CMM), o precursor do CMMI, no *Software Engineering Institute* (SEI), baseando-se em Peter Drucker, afirma que "*embora as tarefas manuais e intelectuais sejam significativamente diferentes, nós podemos medir, analisar e otimizar ambas e,*

[55] Um processo de software é um conjunto de atividades, e seus resultados associados, que produzem um produto de software. Quatro atividades são fundamentais: especificação do software, desenvolvimento do software, validação do software e evolução do software (SOMMERVILLE, 2004, p.8).

[56] KRAFT, P., 1977, *Programmers and Managers: The routinization of computer programming in the United States.*, New York.

portanto, igualmente bem aplicar os princípios de Taylor."[xxxvii] (HUMPHREY, 2000). Durante os anos 1970 e 1980, a maior parte dos esforços relacionados à engenharia de software buscou sedimentar a possibilidade de gerenciamento efetivo do processo de software, entendendo-o como um exercício hierárquico sob total controle do projetista e guiado por especificações estáveis. Com inspiração nos ambientes fabris e suas linhas de produção e montagem, houve intensa busca por ambientes de suporte à produção de software. Assim como as peças de montagem chegavam aos trabalhadores nas linhas de produção que os controlava, tarefas de desenvolvimento poderiam ser trazidas aos desenvolvedores através de ferramentas de software, o que aumentaria a possibilidade de controle do processo de software.

John McCarthy, considerando "*mecanicamente*" um advérbio aplicável quando algo seguisse procedimentos claros e não ambíguos, afirmava que "*qualquer processo intelectual que pudesse ser realizado mecanicamente poderia ser executado por um computador digital*"[xxxviii]. Uma afirmativa assim serve de profunda inspiração para uma engenharia de software cientificista em busca de soluções "mecânicas" para os problemas. Enfim, a linha de montagem, um dos ícones dos tempos modernos, continuamente seduz a engenharia de software que, ironicamente, convive com o *paradoxo do software*[57]: os programadores conseguiram automatizar o trabalho de todo mundo, menos o seu. (MAHONEY, 2004, p.10, 15).

Outra idéia modernista naturalizada na engenharia de software é a idéia de *difusão* de modelos, padrões, técnicas, tecnologias e teorias "universais". Como é uma questão que merece ser detalhada, será tratada na próxima seção.

3.3 Difusionismo

Utilizaremos aqui a explicação de Bruno Latour (2000, p.218ss.), quando descreve o que chama de *modelo de difusão* de fatos e artefatos, para discutirmos os mecanismos relacionados à dinâmica de implantação dos modelos "universais". Tomaremos como exemplo de modelo "universal" um modelo de melhoria de processo de software, tal como o CMMI, mas a discussão valeria, da mesma maneira, caso fosse considerado, por exemplo, a implantação de um pacote ERP — *Enterprise Resource Planning* —, ou o desenvolvimento interno e implantação de um software em uma organização qualquer.

[57] BLUM, B., 1985, "Understanding the software paradox". In: ACM SIGSOFT Software Engineering Notes, v. 10, n. 1, p. 43-46.

Estabelecendo o conceito de *determinismo técnico*, o modelo de difusão quer que acreditemos que fatos e artefatos, quando constituem *caixas-pretas*[58], podem se mover, progredir, difundir, até mesmo existir sem as pessoas. É como se o CMMI pudesse existir por si só, "algo extraído da natureza", sem uma *Carnegie Mellon*, um SEI — *Software Engineering Institute* — e um DoD — *Department of Defense*. Um outro exemplo: é como se a inadequação do uso do *go to* (seção 2.3.1), de um momento para outro tivesse sido "descoberta", um *em si* que dispensaria todo o esforço de Dijkstra e de tantos outros para estabelecê-la. O determinismo técnico considera que a transformação de proposições e protótipos em fatos e artefatos — em teorias e modelos "universais" com a suposta competência que encerram — dá-se por conta de suas qualidades intrínsecas, embutidas no momento de sua concepção por cientistas e engenheiros geniais. Dito de outra forma, para o determinismo técnico a trajetória de fatos e artefatos independe do comportamento das pessoas e entidades alistadas pelas redes sociotécnicas que os consubstanciam e ratificam. Por este ângulo, a engenharia de software tornou-se fato por conta da genialidade da afirmação original de Eckert — *programação deveria tornar-se "engenharia de software"* —, e não por conta de todos que vieram em seguida e escolheram obedecer à inspiração que dela advinha, construindo, assim, a engenharia de software que hoje conhecemos.

Sob o estilo de pensamento modernista, o engenheiro de software (pelo menos a esmagadora maioria, incluindo "todos" — do Brasil inclusive — dos países periféricos em termos de pesquisa e desenvolvimento em ciência da computação) passa a ser mero difusor de modelos "universais" estabelecidos longe dos locais de exercício de sua prática (TEIXEIRA; CUKIERMAN, 2007). Toda vez que o projeto de implantação de um modelo é bem sucedido, o difusionismo garante os méritos para o próprio modelo implantado. Nos casos em que a implantação do modelo fracassa, ou quando, uma vez implantado, não materializa os benefícios esperados, ou mesmo quando, na prática, o modelo é utilizado de uma maneira não totalmente alinhada ao que se previa, os difusionistas socorrem-se nos "fatores não-técnicos" como explicação. Para o difusionismo, o social é algo criado pela necessidade, e com o objetivo, de explicar o fracasso e a inconstância da difusão de modelos "universais". A sociedade, oferecendo diferentes níveis de resistência, é apenas um meio onde os modelos trilham seu caminho de difusão. O fracasso sempre será uma questão de resistência, passividade ou mesmo de cultura local, posto que ao modelo "universal" é sempre garantido o posto de verdade incontestável e comprovada eficácia.

[58] Caixa-preta: vide nota 22.

Porém, elaborar as relações entre o universal e o local, quando se implanta um modelo ou padrão qualquer, é matéria típica de projetação (HANSETH; MONTEIRO, 1998, p.135). Na implantação do CMMI, por exemplo, o desafio não é apenas implantar um modelo "universal", mas sim de projetar relações, papéis e habilidades que se esperam dos desenvolvedores e de todos os demais atores envolvidos. Dominar as prescrições do CMMI talvez seja a parte simples do esforço, porquanto a grande tarefa é construir localmente uma instância deste modelo. Padrões ou modelos utilizados são apenas mais um ator com o qual se negocia para a construção da rede sociotécnica demandada. A verdadeira dificuldade subjaz no transbordo das prescrições dos modelos "universais". Por conseguinte, a divisão apriorística entre o "dentro" e o "fora" só faz reforçar o valor intrínseco, "abstrato", "teórico", dos modelos e padrões "universais", em detrimento do rico valor da prática local de projetação do engenheiro de software ao implantá-los.

O difusionismo impõe uma incoerência à engenharia de software cuja retórica, ao mesmo tempo que os exclui de seu enquadramento, valoriza os "fatores não-técnicos" reputando-os determinantes no sucesso dos projetos. Para o modelo de difusão necessariamente temos "*ciência e técnica, de um lado, e sociedade, do outro*" (LATOUR, 2000, p.233), ou seja, estão do outro lado os grupos que potencialmente podem resistir e que, portanto, precisam participar, ser motivados e envolvidos, como ilustram as afirmativas:

> É necessário *envolver* o time de todos os modos ao longo do processo de mudança [decorrente da implantação de um processo de software], entendendo suas dúvidas e *envolvendo*-o no planejamento do novo processo. Tornando-os *stakeholders* no processo de mudança, é muito mais provável que *eles desejarão fazer* [o processo] *funcionar*.[xxxix] (SOMMERVILLE, 2004, p.680, grifo nosso).
>
> Gerentes de projetos têm que resolver problemas *técnicos* e *não técnicos* através das pessoas alocadas em suas equipes da maneira mais efetiva possível. Eles têm que *motivar* as pessoas [...]. Um gerenciamento pobre das pessoas é um dos mais significativos contribuintes para o fracasso do projeto.[xl] (ibid., p.592, grifo nosso).

Ora, se estão, por definição, em lados opostos como um pode envolver o outro?

3.4 Metáforas e enquadramento alternativos

A rede sociotécnica da engenharia de software vem sendo urdida há mais de 40 anos. É importante reconhecer este caráter de construção, reconhecer sua historicidade, para que possamos criticar a "naturalidade" de sua postura modernista. É possível "desnaturalizar" essa idéia da engenharia de software ser o que é hoje em

dia em decorrência da "ordem natural das coisas", pois, como qualquer outra, sua rede sociotécnica está sempre sujeita à desestabilização, o que equivale a dizer que a ordem que dela resulta é sempre precária. Se novos e antigos aliados enredados pela engenharia de software deixarem de contribuir para que sua rede sociotécnica permaneça estabilizada, poderiam materializar-se outras metáforas acerca do desenvolvimento de software ao passo que a própria engenharia de software poderia tornar-se uma ficção.

Causaria estranheza pensar no desenvolvimento de software como romance, arte ou poesia, ao invés de engenharia. Mas se o investimento em alistar aliados em torno dessas metáforas "alternativas" tivesse preponderado, hoje elas é que poderiam estar "naturalizadas" e o *status quo* do desenvolvimento de software poderia ser bem diferente. Tais metáforas "alternativas" de fato apareceram. Relembrando sua apresentação sobre a estruturação de projetos (esquematizada na Figura 1), na longínqua e seminal conferência no *MIT Labs* em 1956, John F. Jacobs revela: "*eu a chamei de '**O Romance da Programação**' e isso pareceu-me a melhor coisa que já fiz. Mais pessoas lembram-se de mim por aquela palestra do que por contribuições mais significativas que gerei para o programa [SAGE].*"[xli] (CAMPBELL-KELLY, 2004, p.68, grifo nosso). Um ano antes, em 1955, Franz Hohn, na primeira conferência sobre treinamento de Pessoal no campo da computação, falava em "***arte** da computação*" (HOHN, 1955, grifo nosso). Na segunda edição da histórica conferência da OTAN, em 1969, I. P. Sharp também propôs uma metáfora alternativa para o desenvolvimento de software:

> Creio que muito do que construímos como sendo teoria e prática de fato é *arquitetura* e engenharia. [...] Não creio, por exemplo, que o que Dijkstra faz é teoria — creio que, em tempo, provavelmente nos referiremos como "A Escola Dijkstra de *Arquitetura*". (BUXTON; RANDELL, 1970, p.12, grifo nosso).[xlii]

Àquela época a proposta de Sharp não teve muito eco, no entanto, hoje em dia, *arquitetura de software* também é um termo largamente reconhecido e utilizado. Somente por isso, por conta do robustecimento posterior que sofreu, não estranhamos, tanto quanto tendemos a fazer para o uso das expressões "arte" ou "romance", quando se diz que "arquitetura" poderia ter sido uma metáfora alternativa para o desenvolvimento de software. Na linha da arte ou romance, seriam fortalecidas metáforas acessórias como *interpretação*, *individualidade*, *fluxos*, *espírito*, *sujeito*, *persuasão*; em alternativa àquelas hoje fortalecidas na engenharia de software racionalista estabelecida, como *representação*, *sistemas*, *mecanismos*, *ordem*, *objeto*, *controle* (COYNE, 1995, p.246).

A história da engenharia de software vem sendo construída de forma a se encaixar na própria história da engenharia. Todavia, mesmo com a existência de um trabalho que busca relacionar as demais engenharias com sua nuance artística (TOMAYKO, 2002, p.74):

> Engenheiros de software, em uma mal guiada tentativa de obter legitimidade, negaram os aspectos artísticos de seu campo. No entanto, Fred Brooks encontrou um truísmo que indica que arte, ao invés de *handbooks* de engenharia, é claramente dominante [para o alcance dos objetivos da própria engenharia de software, pois bons projetos (*designs*) decorrem de bons projetistas, vide nota 21].[xliii] (TOMAYKO, 2002, p.74).

À primeira vista a idéia das metáforas alternativas pode parecer simplista demais, principalmente para aqueles que, como os engenheiros de software, têm, na esmagadora maioria das vezes, uma formação calcada na longa tradição do pensamento moderno. "É simples assim?", poderia ser perguntado, "basta mudar a metáfora que mudaríamos todo *status* estabelecido do desenvolvimento de software?" Certamente não é simples assim. Além do mais, a noção de metáfora aqui considerada é aquela na qual extrapola-se a mera conotação retórica justapondo-se elementos materiais de experimentação e de produção. Como visto, uma metáfora deixa ser mero recurso retórico para, no caso, materializar-se numa disciplina menos por conta de seu autor do que por conta de todos aqueles que vêm depois e a inserem em novas cadeias de fatos e artefatos que, corroborando a metáfora original, a transformam em realidade. Criticar a afirmação de Eckert em 1965 era muito mais fácil do que fazê-lo hoje. Segundo Latour (2000, cap. 4), à medida que a *caixa-preta* vai se fechando, ou seja, a resolução das controvérsias vai consolidando o fato ou o artefato, cada vez mais torna-se oneroso se interpor ao processo de consolidação desse fato ou artefato. Além de uma questão de custo, não se sabe *a priori* a possibilidade de se reabrir uma *caixa-preta*, algo também dependente de traduções. Quanto custou rebaixar Plutão da condição de planeta? Quanto custaria sobrepujar o sistema heliocêntrico pelo sistema geocêntrico em pleno século XXI? Seria possível? Quando custaria, hoje em dia, defender que desenvolver software não é engenharia em favor de outra metáfora?

Entretanto, longe vai o tempo em que experiências como os "BIG-L" conseguiam reproduzir um mundo moderno em suas redes sociotécnicas e, por isso, contribuíam para a naturalidade da visão do desenvolvimento de software ser uma engenharia. A comunidade acadêmica e prática da engenharia de software reconhece, cada vez mais, a relevância dos chamados "aspectos não-técnicos", ou as "questões sociais", no sucesso ou fracasso dos projetos (ARNOLD, 2003, p.228). Ou seja, o transbordo do enquadramento tecnocêntrico da engenharia de software, o seu "fora", pressiona,

crescentemente, reclamando solução. Não deverá ser surpresa se surgirem, também cada vez mais, metáforas alternativas para o desenvolvimento de software.

A retórica da importância dos "fatores não-técnicos", ou dos "fatores humanos" para os projetos de software tornou-se comum nos anos 1990 (BOEHM, 2006), mas, já em 1975, Frederick Brooks descrevia com propriedade a importância dos "aspectos humanos" nos projetos de software (BROOKS, 1995). Mesmo antes, logo no início dos anos 1960, podia-se identificar a prática de desenvolvimento de software em cenários radicalmente diferentes daqueles que retratavam uma prática centralizada e hierarquizada, como a que ocorreu nos "BIG-L", por exemplo:

> Apenas a IBM tinha projetos em tamanho e complexidade comparáveis [aos militares] nesse período. A maioria dos desenvolvedores de software do mercado estavam trabalhando em projetos menores, esforços mais autocontidos requerendo muito menos programadores. *Programadores nessas instalações trabalhavam em múltiplos projetos envolvendo uma ampla faixa de problemas de negócio. Eles freqüentemente participavam de diversos aspectos do desenvolvimento de sistemas, desde o levantamento de requisitos até o projeto do sistema e sua implementação* [...].[xliv] (ENSMENGER; ASPRAY, 2002, p.142, grifo nosso).

Em um tal cenário, como não podia existir tão claramente a divisão burocrática de papéis e funções, o analista/programador era um "faz-tudo". Com ele mesmo fazendo tudo, as especificações perdiam um pouco de sua força, muitas vezes sequer sendo feitas, pois dificilmente o profissional fazia uma especificação se ele mesmo a iria implementar. Perdia-se o sentido e a possibilidade do projeto ser controlado hierarquicamente por especificações congeladas, bem como o exercício de um controle centralizado, garantido pela divisão de tarefas e subordinação a um método bem definido. Em suma, as pressuposições modernistas, inspiradoras da engenharia de software, não estavam presentes: onde estavam, por exemplo, as especificações que representavam fidedignamente o mundo subjacente ordenado e estável, e o método garantidor da possibilidade de controle hierárquico do processo de desenvolvimento?

Mesmo sob cenários, na maioria das vezes, desfavoráveis à aplicação de uma visão de mundo mecanicista, a engenharia de software trilhou seu caminho modernista, o que faz até hoje. Bem coerente com o difusionismo, ao invés da engenharia de software questionar o modelo mecanicista que propunha para o desenvolvimento de software em cenários desfavoráveis, a resposta esperada, que não demoraria a surgir, seria atribuir aos "fatores não-técnicos" a culpa pelos malogros. Cabe destacar que um estilo de pensamento permite a formulação apenas de questões que consegue resolver. Por isso, para manter sua coerência interna como disciplina, a engenharia de software, como de resto as mais diversas disciplinas de ciência e tecnologia que conhecemos, "naturaliza" a existência de um "dentro",

"técnico" e "correto", e de um "fora", o "contexto não-técnico" que passa a responder pelos erros e fracassos nos projetos. Está "dentro" da engenharia de software, isto é, faz parte da natureza daquilo que deve ser seu campo de atuação, e, portanto, deve ser o objeto de seus modelos e padrões "universais", apenas o que se coadunar com as premissas da visão mecanicista, por exemplo, representação, formalização, ordem, estabilidade, controle, razão, método (quadro mais interno da Figura 3). Dizendo de outro modo, a engenharia de software estabelece, *a priori*, um enquadramento dos objetos e fenômenos que deve manipular. Tudo que produz incertezas, tal como o comportamento das pessoas, tido como incerto e pouco previsível, transborda deste enquadramento e deve ser considerado "fora", "não pertencente" à engenharia de software, aos seus modelos, padrões e ferramentas. É o que está enumerado fora do quadro mais interno da Figura 3. (TEIXEIRA, 2006).

Figura 3 — O enquadramento mecanicista / positivista da engenharia de software e seus transbordamentos "sociais, culturais, políticos"

Mesmo encorajando os praticantes a enfrentarem a questão, DeMarco e Lister, imbuídos dessa perspectiva dicotômica "dentro" x "fora", explicitam sua crença de que os problemas relacionados ao comportamento das pessoas não pertencem à engenharia de software:

> A sociologia de projetos e equipes *pode estar bem fora de seu campo de especialidade*, mas não além de sua capacidade. Qualquer que seja o nome que você dê aos problemas relacionados com as pessoas, eles muito provavelmente causarão mais dificuldades [...] do que as questões de projeto (design), implementação e metodologia que você terá que lidar. De fato, esta idéia é a tese básica de todo este livro.[xiv] (DeMARCO; LISTER, 1999, p.4, grifo nosso).

Sem menoscabar todo o desenvolvimento da engenharia de software, em verdade, ela logrou bom êxito apenas em determinadas questões, como delimitam Dahlbom e Mathiassen (1993, p.15):

> Quando tentamos controlar o mundo com programas de computador ou métodos para o desenvolvimento de sistemas, não podemos nos esquecer que a visão mecanicista tem por premissa que o mundo que tentamos entender e controlar é, por si próprio, um mundo ordenado, um sistema fundamentalmente imutável[xlvi].

Propondo uma visão adaptada do modelo de desenvolvimento cascata[59], Mahoney (2002, p.37-38) permite entrever que se alcançou uma grande evolução no desenvolvimento de software nos problemas bem tratados sob um enquadramento modernista. A engenharia de software foi muito bem sucedida em determinadas condições de onde surgiram soluções tecnológicas para as *tarefas acidentais* (BROOKS, 1986). Essas tarefas correspondem às etapas abaixo da linha pontilhada na Figura 4. Abaixo da linha pontilhada, os problemas relacionados podem "facilmente" ser formalizados matematicamente, podem ser traduzidos fidedignamente para as etapas mais abaixo.

> Hoje, relativamente poucos erros ocorrem nos estágios da metade inferior do esquema [Figura 4]. Isso não surpreende, uma vez que eles são os aspectos da computação melhores entendidos matematicamente, e este entendimento tem sido traduzido na prática em ferramentas como compiladores para linguagens de programação de alto-nível.[xlvii] (MAHONEY, 2002, p.38).

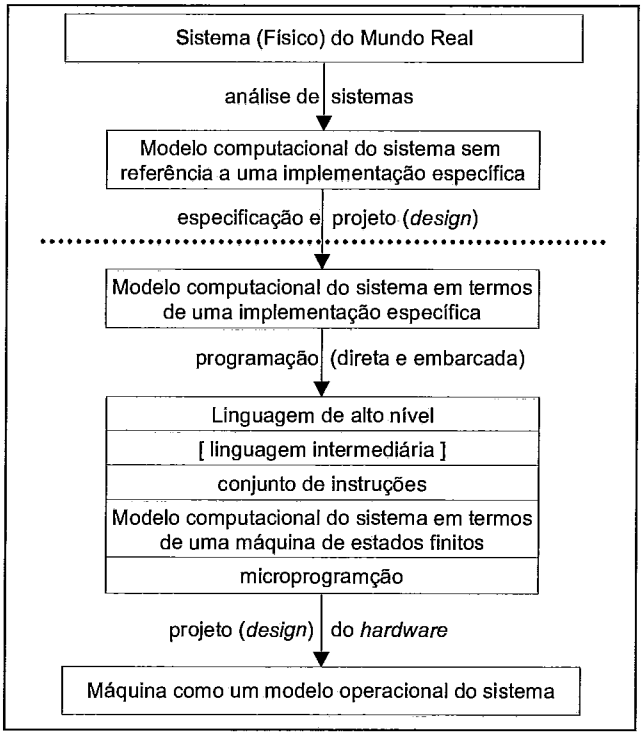


Figura 4 — Desenvolvimento de Software: Níveis de Abstração
(MAHONEY, 2002, p.37)

[59] O tradicional "*waterfall lifecycle*" foi formalizado por Winston Royce (1970). Campbell-Kelly (2004, p.69) descreve que este modelo tão popular a partir dos anos 1970 é sucessor do método seguido no SAGE (vide Figura 1), embora tenha sido influenciado por outras fontes também. O esquema básico consiste em fragmentar o esforço de desenvolvimento em seqüências temporais de tarefas, congelando as especificações entre elas e estruturando hierarquicamente a divisão de trabalho desde o analista até o codificador, passando pelo programador sênior e demais programadores.

Contudo, o objeto da engenharia de software é justamente o que antecede à produção de especificações e projetos, incluindo-os também. Ou seja, tudo que está acima da linha pontilhada, as *tarefas essenciais* (BROOKS, 1986), permitindo afirmar que o objeto de análise sai da esfera do computador e abrange o mundo. Ora, diferente do que ocorre abaixo da linha pontilhada da Figura 4, onde está o maior investimento acumulado, acima dela nem sempre (talvez quase nunca) verificamos a validade dos princípios de estabilidade, ordem, formalização, ou mesmo a possibilidade de enredar esses princípios na rede sociotécnica em questão.

> [Na parte superior do esquema (Figura 4)...] é onde a engenharia de software tem focalizado sua atenção desde os anos 1970. Mas é também onde a ciência relacionada deixa os computadores para abranger o mundo inteiro e interagir com as ciências (se é que elas existem) pertinentes aos sistemas a serem modelados computacionalmente. [...] Mas esta é a questão que engenheiros de software compartilham com cientistas que tem deixado o computador para ocupar-se de reinos que não são acessíveis nem por experimentos nem pela análise matemática.[xlviii] (MAHONEY, 2002, p.38).

O mundo atual, um mundo com processamento distribuído, com desenvolvimento de software também distribuído, com mudanças céleres nas atividades de negócio, com transformação constante nos mercados, com sistemas de software ubíquos, com elevada complexidade tecnológica, com elevada complexidade nas organizações, com interconectividade "universal", dentre tantas outras características, traz grandes dificuldades para uma engenharia de software modernista. Os problemas começam a aparecer porque tenta-se aplicar a visão mecanicista fora do enquadramento que a suporta. Isto é, tenta-se resolver os problemas acima da linha pontilhada da Figura 4 sem ter à disposição as premissas que melhor podem ser consideradas abaixo dela. Aparentemente, a engenharia de software busca ampliar seu enquadramento para contemplar também parte do "não-técnico", supostamente "fora" da engenharia de software, como sugerido na Figura 5 (TEIXEIRA, 2006), porém, continuando a admitir a possibilidade de fatoração dos problemas em questões "técnicas" de um lado e questões "humanas, sociais ou não-técnicas" de outro, preservando a idéia de que essas grandes divisões existem "naturalmente no mundo".

James Tomayko e Orit Hazzan (2004) denotam bem a tentativa de se aumentar o enquadramento da engenharia de software subjugados pela modernidade, pois escoram-se na suposta divisão nítida da realidade em "aspectos humanos", de um lado, e "aspectos de engenharia, ou técnicos", de outro, como bem denota o título de seu livro "*Human Aspects of Software Engineering*". Constatando que tem tornado-se comum a percepção da situação paradoxal da engenharia de software — com seu

"fora" cada vez mais reconhecido como determinante nos projetos — eles afirmam que atualmente é possível observar uma diminuição da ênfase em seu "aspecto de engenharia". O que poderia ser explicado parcialmente pela impossibilidade de desenvolvimento de um processo, orientado a engenharia, que respondesse a todos os problemas típicos dos projetos de software. Abordagens mais recentes entendem a engenharia de software como disciplina multifacetada, devotando mais atenção aos "aspectos humanos". (TOMAYKO; HAZZAN, 2004, p. 116).

Figura 5 — Enquadramento ampliado da engenharia de software, ainda dividindo: "técnico" x "não-técnico"

No entanto, a maioria das abordagens considera que projeto e desenvolvimento de software são apenas uma questão de técnica, quando muito, com conseqüências sociais. Isto conformou a engenharia de software como uma disciplina que crê lidar sobretudo com objetos de complexidade "técnica", demandando somente, e cada vez mais, ferramentas, métodos, modelos e princípios técnicos (HIRSCHHEIM; HEINZ; LYYTINEN, 1995, p.1). Alfonso Fuggetta corrobora esta percepção, quando cita o exemplo notoriamente bem sucedido das ferramentas de suporte à *gerência de configuração*[60] cujo sucesso advém da efetividade com que abordam tarefas extremamente maçantes e repetitivas, "*razoáveis de serem automatizadas*". E questiona: será que não estaríamos tentando modelar e automatizar coisas que intrinsecamente não podem ser modeladas e automatizadas no ciclo de vida de desenvolvimento de software? (FUGGETTA, 2000). Essa pergunta retrata a dúvida sobre a efetividade de se tentar resolver o "não-técnico" de uma maneira mecanicista.

[60] Gerência de configuração de software visa ao estabelecimento e manutenção da integridade de todos os produtos de software criados ao longo do processo de desenvolvimento. Basicamente tem dois objetos principais: controle de versões e gerenciamento das mudanças (CHRISSIS et al., 2003).

O enfoque sociotécnico permite uma outra via de ampliação do enquadramento da engenharia de software. Uma via contrária às divisões apriorísticas em "fatores" ou "aspectos", contrária à definição "teórica" do que está "dentro" e do que está "fora" da engenharia de software. Uma via que amplia o enquadramento ao desconstruir as tais divisões em "fatores", reconhecendo que o técnico está presente no social, tanto quando o social está presente no técnico — indissociável e sociotecnicamente —. A Figura 6 ilustra um enquadramento mais amplo porque não supõe divisões existentes *a priori*.

A alternativa sociotécnica reduz a distância entre "teoria" e "prática" porque, ao invés de entendê-las como categorias fixas *a priori*, busca reconhecê-las através dos limites empíricos e variáveis configurados caso a caso. O instrumental sociotécnico contrapõe o "desejo normativo" moderno — um desejo simplificador, fragmentador, que busca regras e leis, similaridades e universalidades — a um "desejo descritivo" — um que busca o detalhe, a particularização, a *descrição densa*[61], e, fundamentalmente, a "desnaturalização" de modelos através da explicitação de sua historicidade — (TEIXEIRA; CUKIERMAN, 2007).

Figura 6 — Enquadramento ampliado da engenharia de software, sem divisões entre "técnico" x "não-técnico"

[61] Termo proposto em: GEERTZ, C., 1978, *A interpretação das culturas, Rio de Janeiro*, Zahar Editores.

Capítulo 4
Um Cenário Moderno de Desenvolvimento de Software

Por conta de seus vínculos com a modernidade, é praxe na engenharia de software oferecer narrativas sobre uma determinada experiência de desenvolvimento de software bem sucedida destacando um "conteúdo técnico" — no caso, processos, métodos, técnicas, modelos, melhores práticas e padrões — que, dotado de supostas qualidades intrínsecas, foi implantado em um "contexto organizacional" apropriado. Ao final de uma história assim, conclui-se que os bons resultados alcançados são conseqüências das qualidades intrínsecas deste "conteúdo técnico" — a própria engenharia de software —, considerado preexistente ao "contexto organizacional" em que foi implantado. Trata-se do difusionismo, discutido na seção 3.3.

Outra abordagem seria evitar distinguir, *a priori*, contexto e conteúdo e reconfigurar a narrativa a partir da prática local e situada, a partir das associações entre as mais variadas entidades que culminam com a construção concomitante de contexto e conteúdo. Sobre as entidades, não importariam distinções acerca de sua possível natureza — técnica, social, humana, não-humana —, privilegiando-se a observação de como atuam na rede que constróem através de suas associações. A prática bem sucedida de software resulta da estabilização dessa rede, uma *rede sociotécnica*, conceito que dá conta da construção indistinta de contexto e conteúdo.

Neste capítulo, será exercitada esta segunda abordagem com um exemplo sobre o alto nível de estruturação na prática de desenvolvimento de software alcançado no BNDES — Banco Nacional de Desenvolvimento Econômico e Social — nos anos 1970, que não pode ser atribuído "friamente" à aplicação da engenharia de software, mesmo porque, àquela época, como visto no capítulo 2, ela apenas ensaiava seus primeiros passos como disciplina autônoma.

4.1 A informática do BNDES nos anos 1970

Não é possível investigar uma prática de desenvolvimento de software sem imbricá-la com seu contexto de produção. A informática no BNDES foi instituída já nos anos 1960. Um quadro sinóptico de seu estabelecimento principia pela criação, em 1966, do Serviço de Processamento de Dados (SPD) subordinado ao Diretor-

Superintendente. Naquela época, a operacionalização dos serviços de processamento de dados ocorria através da contratação de birôs de serviço. Em 1967 foi criada a Coordenação de Serviços Específicos (COSE) que, além de outros serviços — como assistência e previdência, documentação e divulgação, controle interno —, passou a responder pelo SPD. Em 1971, o BNDES já dispunha de um computador próprio para o processamento de dados e o SPD passou a compreender duas divisões — análise e programação; e operação e controle — dentro do Departamento Administrativo[62].

4.1.1 A estruturação

O BNDES (à época apenas BNDE) tornou-se uma empresa pública em 1971 e, "*com o propósito de aumentar sua flexibilidade e tornar-se ainda mais apto a fomentar o desenvolvimento econômico do país*", empreendeu então uma significativa reorganização administrativa, contratando para isso a consultoria *Booz, Allen and Hamilton International Inc. Management Consultants* (BOOZ ALLEN, 1972, p.2-3). É a partir dessa reorganização que se pode datar o início do uso ostensivo dos recursos de informática do BNDES, pois anteriormente seu uso era muito incipiente, como denota o fato de existir apenas um analista de sistemas nos quadros do BNDES até o final de 1972. Com a reorganização administrativa foi criada uma estrutura de informática e começaram a ser contratados analistas de sistemas através de concursos públicos. Segundo o analista de sistemas Melvyn Cohen: "*em 1974 e 1976 o banco começou a contratar profissionais de informática mesmo — analistas de sistemas,* [...] *programadores eram terceirizados contratados do SERPRO*" (Entrevista concedida ao autor em 20.06.2006).

Na reestruturação organizacional de 1972, foi criada a Área de Serviços Gerenciais (ASG) composta por quatro departamentos: financeiro, jurídico, pessoal e administrativo[63]. Uma das responsabilidades dessa área era "*oferecer a maior quantidade possível dos mais variados* ***serviços administrativos, inclusive, processamento de dados*** *[...].*" (BOOZ ALLEN, 1972, p.144, grifo nosso). Assim, até aquele momento, a informática do BNDES era uma divisão, denominada "processamento de dados", no departamento administrativo (Figura 7), algo, portanto, com *status* de serviço administrativo. Osmar Frota, analista de sistemas ainda no BNDES, descreve essa reestruturação.

---

[62] Resoluções Nºs 245/66; 283/67; 402/71 do Conselho de Administração do BNDES.
[63] Resolução de Diretoria Nº 425/72.

> Nos anos 1960/1970 existia uma estrutura clássica de administração de empresas de informática. A grande maioria das empresas prestadoras de serviço no Brasil tinha [além de] os departamentos administrativo e financeiro, os DEPROJ — departamento de projetos [de desenvolvimento e manutenção de sistemas], e os DEPROD — departamentos de produção. Especificamente no BNDES, pegamos o final dessa estrutura de desenvolvimento de projetos antiga, o modelo clássico: um departamento de projetos e um departamento de produção, [que funcionavam] em prédios diferentes. (Entrevista concedida ao autor em 31.07.2006).

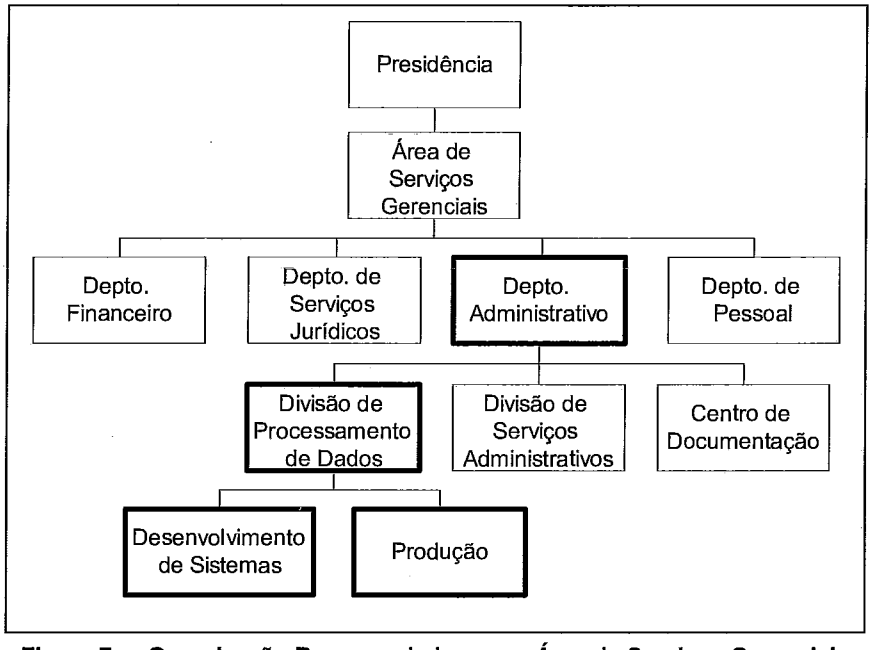


Figura 7 — Organização Recomendada para a Área de Serviços Gerenciais
Adaptado de Booz Allen (1972, anexo XXI)

Atreladas à estrutura organizacional estabelecida, foram sugeridas funções especializadas necessárias ao bom funcionamento da divisão de processamento de dados, como, por exemplo (BOOZ ALLEN, 1972, p.429):

- Supervisor de treinamento e planejamento interno — para garantir a capacitação adequada das equipes ao longo do tempo e para suportar, continuamente, o "planejamento futuro" da divisão de processamento de dados.
- Supervisor de desenvolvimento de sistemas — para supervisionar o trabalho dos líderes de projetos de sistemas.
- Analista de procedimentos manuais — para prover procedimentos de apoio à utilização dos sistemas. Um comentário acerca desta função será necessário na seção 4.1.4.
- Programador da produção — para viabilizar o escalonamento da produção.

Além dessas funções especializadas sugeridas, outras tantas eram requeridas na operação de uma instalação padrão de computador da época. Comparado a hoje

em dia, a operação de um computador era algo bastante complexo que envolvia diversas categorias de profissionais, como digitadores, preparadores de lotes de processamento, operadores, responsáveis pela manipulação dos cartões perfurados, controladores de bibliotecas, discos, fitas. Existia, também, uma nítida separação entre os papéis do analista de sistemas e do programador. Todas essas funções povoavam a estrutura da divisão de processamento de dados, cujo detalhamento está registrado na Figura 8.

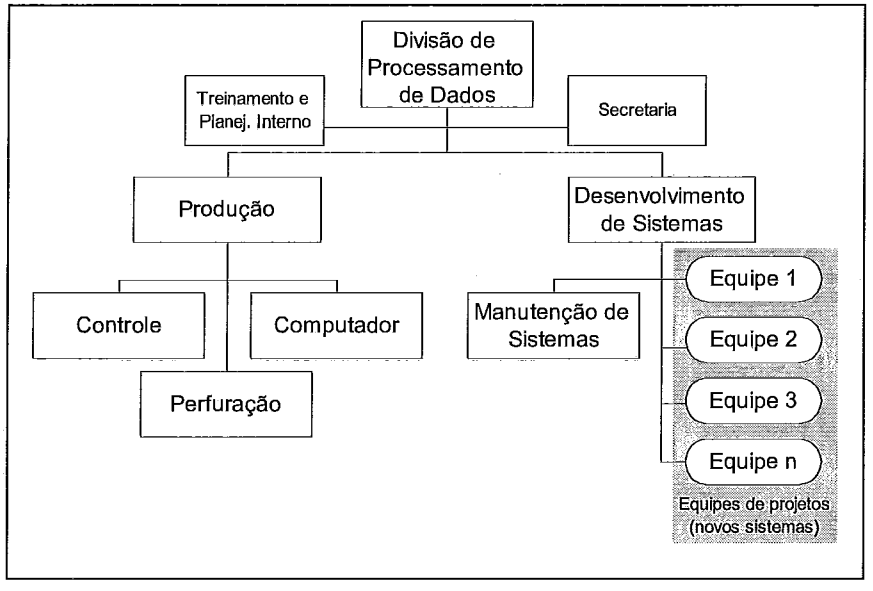


Figura 8 — Organização Recomendada para o Processamento de Dados
Adaptado de Booz Allen (1972, anexo XXX)

As equipes de projeto eram constituídas por analistas de sistemas, programadores e, quando fosse apropriado, pessoal das áreas usuárias. Os líderes dessas equipes se reportavam diretamente ao chefe da divisão de processamento de dados (BOOZ ALLEN, 1972, p.427).

> Os analistas, naquela época, eram vinculados a um projeto que tinha normalmente um coordenador de serviços [que cumpria a função de supervisor de desenvolvimento de sistemas]. O coordenador era de fato um coordenador de serviços, ao qual estava vinculado [, isto é, exercia efetivamente a função de supervisão e coordenação das atividades relacionadas aos sistemas e serviços de uma área de negócio específica]. Por exemplo, [...] serviços administrativos, serviços de cobrança. (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

À estrutura organizacional e à segregação de funções bem definidas da divisão de processamento de dados, somava-se um terceiro elemento complementar, uma metodologia, esquematizada na Figura 9, que visava orientar as equipes de projeto a cumprirem seus objetivos.

É possível inferir a partir do relatório da BOOZ ALLEN (1972, anexo XXIX), que as grandes etapas da metodologia poderiam ser descritas da seguinte forma:

1. *Estudo de Viabilidade do Sistema.* O analista realizaria os estudos necessários para julgar a viabilidade técnica e econômica do sistema. A organização recomendada para a função de processamento de dados definia que, tipicamente, participariam nesta etapa um analista de sistemas sênior (líder do projeto) e um representante da organização usuária.

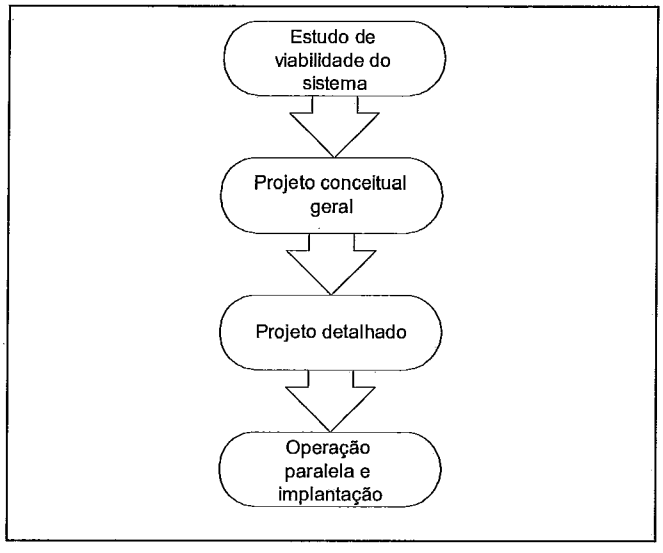


Figura 9 — Etapas do Desenvolvimento de Sistemas no BNDES dos anos 1970
Adaptado de Booz Allen (1972, anexo XXIX)

2. *Projeto conceitual geral.* O analista de sistemas deveria levantar os requisitos do sistema a partir das atividades que seriam automatizadas. Normalmente, nesta etapa havia participação do usuário, supostamente o conhecedor das atividades consideradas. O trabalho do analista de sistemas seria o de representar e formalizar o que deveria ser implementado. Fichas de entradas de dados (Figura 10) e protótipos de relatórios viabilizavam a validação dos requisitos. Era formalizada, em um nível mais alto de abstração, uma representação conceitual do sistema. Tipicamente participariam nesta etapa um analista de sistemas sênior (líder do projeto), um representante da organização usuária e dois analistas de sistemas.
3. *Projeto detalhado.* O analista de sistemas detalharia a representação conceitual já desenvolvida em uma especificação formal. O detalhamento deveria ser suficiente para possibilitar que o programador realizasse seu trabalho. Ao analista de sistemas cabia submeter-se à racionalidade de maximização da eficiência de utilização dos caros e limitados recursos computacionais. A especificação servia para controlar todo o processo de construção do sistema, a saber, programação e testes. A organização típica da

equipe de projeto nesta etapa previa a participação de um analista de sistemas sênior (líder do projeto), dois analistas de sistemas e dois programadores.

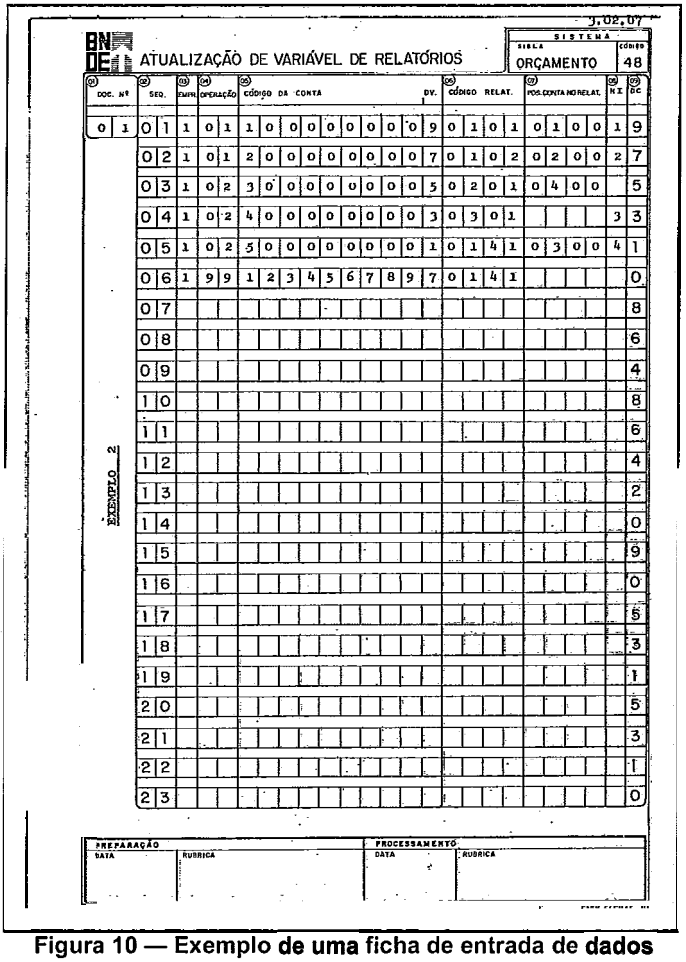


Figura 10 — Exemplo de uma ficha de entrada de dados
Fonte: DESIS — Departamento de Sistemas, Manual do Usuário, Sistema: Orçamento-Programa, 197?.

4. *Operação paralela e implantação*. Nesta etapa o sistema já estaria construído. O usuário seria envolvido novamente, devendo ser treinado e se adequar às possíveis alterações nas atividades decorrentes da utilização do sistema. Tipicamente deveriam participar um analista de sistemas sênior (líder do projeto), um representante da organização usuária, quatro auxiliares de administração (da organização usuária), um analista de sistemas e um programador.

> [Resumindo:] então tinha-se uma metodologia de desenvolvimento, simples, tranqüila, feita internamente com a ajuda da Booz Allen, mas você tinha também uma estrutura administrativa [de suporte bem definida] e você tinha o *responsável por* [uma determinada função]. (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

4.1.2 Os profissionais

Um raciocínio trivial nos fará admitir que de nada adiantaria uma perfeita coerência entre a estrutura organizacional e as funções e papéis previamente definidos se as pessoas não estiverem à altura de suas atribuições. Pessoal treinado e motivado é condição necessária ao bom desempenho da organização. Nesta seção, serão apresentadas breves considerações sobre a motivação e a formação dos profissionais da informática do BNDES nos anos 1970.

Para inferirmos a motivação daqueles profissionais, precisamos atentar para a vigência do clima de "milagre econômico" no Brasil e do papel do BNDES no cenário nacional, especialmente através dos PND's — Planos Nacionais de Desenvolvimento. Sob dísticos como "ninguém segura este país", empolgação e otimismo embalavam o corpo de funcionários do BNDES, que há pouco mais de vinte anos vinha atuando de maneira decisiva e notória para o desenvolvimento brasileiro.

> Durante as duas décadas de sua existência, o Banco Nacional de Desenvolvimento Econômico foi a principal instituição nacional responsável pelo financiamento do desenvolvimento econômico. Tem sido sua responsabilidade constante contribuir para o progresso econômico do Brasil. (BOOZ ALLEN, 1972, p. 2).

Um exemplo claro de motivação dos empregados nos chega através do relato de Ângela Oliveira, uma economista que já poderia ter se aposentado no início de 2007, que teve uma participação muito próxima e ativa na informática do BNDES (principalmente a partir dos anos 1980): "*Entrei no banco em 1974, entrei uma sonhadora. Eu fiz economia para vir trabalhar no BNDES, queria trabalhar no PND*" (entrevista concedida ao autor em 07.07.2006). Vários profissionais de informática daquela época também atestam sua motivação em trabalhar no BNDES. Alguns deles, inclusive, revelam a opção de terem renunciado a uma remuneração maior em seus trabalhos anteriores na iniciativa privada, quando foram para o BNDES.

Além da motivação, os profissionais de informática traziam uma formação muito aderente ao negócio de informática da época — aderente à estrutura organizacional existente, à forma de operação e trabalho e aos tipos de serviços demandados —, porque, em sua esmagadora maioria, eram formados nas próprias empresas de informática, tanto as fornecedoras de equipamentos, como a IBM e a *Burroughs*, quanto nos birôs provedores de serviços de informática. Naquela época, apenas começavam a existir alguns poucos cursos na área de computação. É o que relatam respectivamente Osmar Frota, ele próprio um egresso de um grande birô do início dos anos 1970, e Mário Esteves Filho, engenheiro de produção, atual superintendente da área que abriga os departamentos de informática no BNDES.

> Na década de 1960, [...] o pessoal de informática (os analistas de sistemas) no Brasil era formado em 3 linhas de pensamento sobre o desenvolvimento de sistemas: IBM, Burroughs e alguns outros fabricantes menores — Univac, Bull, Honeywell. Havia uma predominância muito grande da IBM. Não se tinha analistas e programadores formados em faculdades, nem existiam faculdades de informática, as empresas procuravam técnicos formados em engenharia, economia. As empresas no Brasil contratavam profissionais oriundos dessas 'escolas de formação'. Começaram a surgir os grandes birôs de serviço (processamento de dados, análise e programação) e as primeiras metodologias, eram empresas muito bem estruturadas [que, em seguida, também tornaram-se provedores de profissionais para o mercado]. (Entrevista concedida ao autor em 31.07.2006).

> Quando o banco reestruturou a informática, em meados de 1970, veio muita gente da Burroughs, contratados que depois acabaram efetivados no banco. Naquela época não existia formação em informática. O Ricardo Farias [chefe da divisão de processamento de dados, por exemplo,] era advogado. As pessoas se formavam na prática. (Entrevista concedida ao autor em 22.08.2006).

A formação desses profissionais de informática era calcada em um *saber prático.* Formados nas empresas, traziam enraizada a adequação aos ambientes de trabalho existentes, especialmente a cultura de obediência a padrões e cumprimento de papéis preestabelecidos, características que os profissionais de informática acabaram perdendo à medida que passaram a ser formados nas universidades. Vale retomar a observação de Hamming (1968)[64], sobre os graduados nos cursos de computação, dos EUA, que eram, a seu ver, "*incapazes de disciplinar seus esforços próprios para conseguirem fazer o que dizem que farão no tempo estabelecido e de uma maneira prática*". Osmar Frota percebe o mesmo em sua experiência:

> [Hoje em dia, se] você pega os analistas formados na PUC [Rio], eles falam diferente do pessoal formado lá na Nacional [UFRJ], que é diferente do pessoal da UFF que, por sua vez, é diferente do pessoal da escola de Minas Gerais, cada escola tem um método diferente. Aí faz-se uma confusão danada e ninguém se entende. (Entrevista concedida ao autor em 31.07.2006).

Devido à sua formação ocorrer nas empresas, parece aceitável portanto que os profissionais envolvidos com a operação dos computadores daquela época vivenciassem facilmente a segregação hierárquica de funções, processos e metodologias definidos para garantir a coordenação e o controle das atividades. A consecução de qualquer tarefa, simples como compilar um programa, normalmente demandaria a mediação, por exemplo, do perfurador de cartões e do operador. Dado o custo do processamento e sua natureza *batch*[65] — em que os resultados quase

[64] Vide nota 18.

[65] A filosofia dos sistemas de processamento em lotes — *batch* — surgiu em decorrência dos altos custos dos primeiros sistemas computacionais e da constatação de que os computadores passavam a maioria do tempo ociosos, esperando que os operadores provessem os recursos necessários ao processamento das tarefas, como, por exemplo, carregar os cartões do compilador antes da execução da tarefa de compilação. A solução foi agrupar todas as tarefas, com seus respectivos recursos demandados, em lotes que, uma vez

nunca eram obtidos na hora —, todos aceitavam pagar o preço da submissão aos pontos de controles estabelecidos, na tentativa de garantir o bom êxito das tarefas. Hoje em dia, com a tempestividade da microcomputação, perdeu-se esta referência, pois se está acostumado à realização imediata das operações necessárias, no microcomputador, sem interagir, comunicar e negociar o serviço com qualquer outra instância. Portanto, com a presença de intermediários, existe uma dinâmica de execução das tarefas que reforça a necessidade e a aceitação de hierarquias de funções especializadas e de pontos de controle. Como diz Renato Gouvêa, analista de sistemas que entrou no BNDES no início dos anos 1980: "*o pessoal do ambiente de grande porte aceitava mais porque era um ambiente mais controlado*" (entrevista concedida ao autor em 26.06.2006).

A tarefa de desenvolvimento de software não era exceção à regra, pois a ela também se aplicavam a segregação de funções e a obediência a uma metodologia de forma a viabilizar a coordenação e o controle. "*Existia um método para se trabalhar e a turma seguia, não tinha conversa, [...] nessa época tinha-se muito controle*" (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

> [Quando existiam separados] o analista de sistemas, o programador e o operador, havia a necessidade de se ter metodologia e descrição de fases para se [conseguir] passar para outra pessoa essa responsabilidade [sobre o serviço em questão]. (Melvyn Cohen, em entrevista concedida ao autor em 20.06.2006).

É útil antecipar a discussão que se estenderá até o final do capítulo e chamar a atenção, já neste ponto, para a nítida existência de elementos do discurso moderno neste caso do BNDES. Podem ser vistos a divisão burocrática de papéis e funções, o controle hierárquico das atividades, guiado por especificações que formalizavam representações dos sistemas, o controle centralizado com suporte do CPD — *centro* de processamento de dados — e os métodos e metodologias, tudo coadunando-se com uma visão mecanicista de mundo.

4.1.3 Hardware, software e quantitativo de pessoal da época

Ainda que de forma breve, para a compreensão da prática de desenvolvimento de software ocorrida no BNDES nos anos 1970 e no início dos anos 1980, é útil observar a evolução do hardware disponível, dos sistemas de software utilizados e desenvolvidos, e do número de profissionais diretamente lotados na informática.

---

submetidos ao processamento, ocupassem a máquina ininterruptamente. (TANENBAUM, A.S., 1995, *Sistemas Operacionais Modernos*, Prentice-Hall do Brasil, p.6).

O grande desenvolvimento da microeletrônica torna surpreendente para muitos que a instalação de computador do BNDES, há apenas 34 anos atrás, em 1973, era dotada de um computador, um *mainframe* IBM, que dispunha de apenas 16Kbytes de memória principal, sem memória em disco, utilizando apenas fitas magnéticas. Hoje em dia, por mais simples que seja, até um telefone celular tem pelo menos um cartão de 64Kbytes de memória e os microcomputadores alcançam, usualmente, a casa dos gigabytes de memória principal. Aquele *mainframe* do BNDES já era da linha IBM/360 mas operava emulando o antigo IBM 1401, monoprogramado, de 1960, devido à (não) portabilidade das aplicações então existentes. Eram disponibilizadas apenas as linguagens Assembler, FORTRAN e Autocoder[66] (Figura 11).

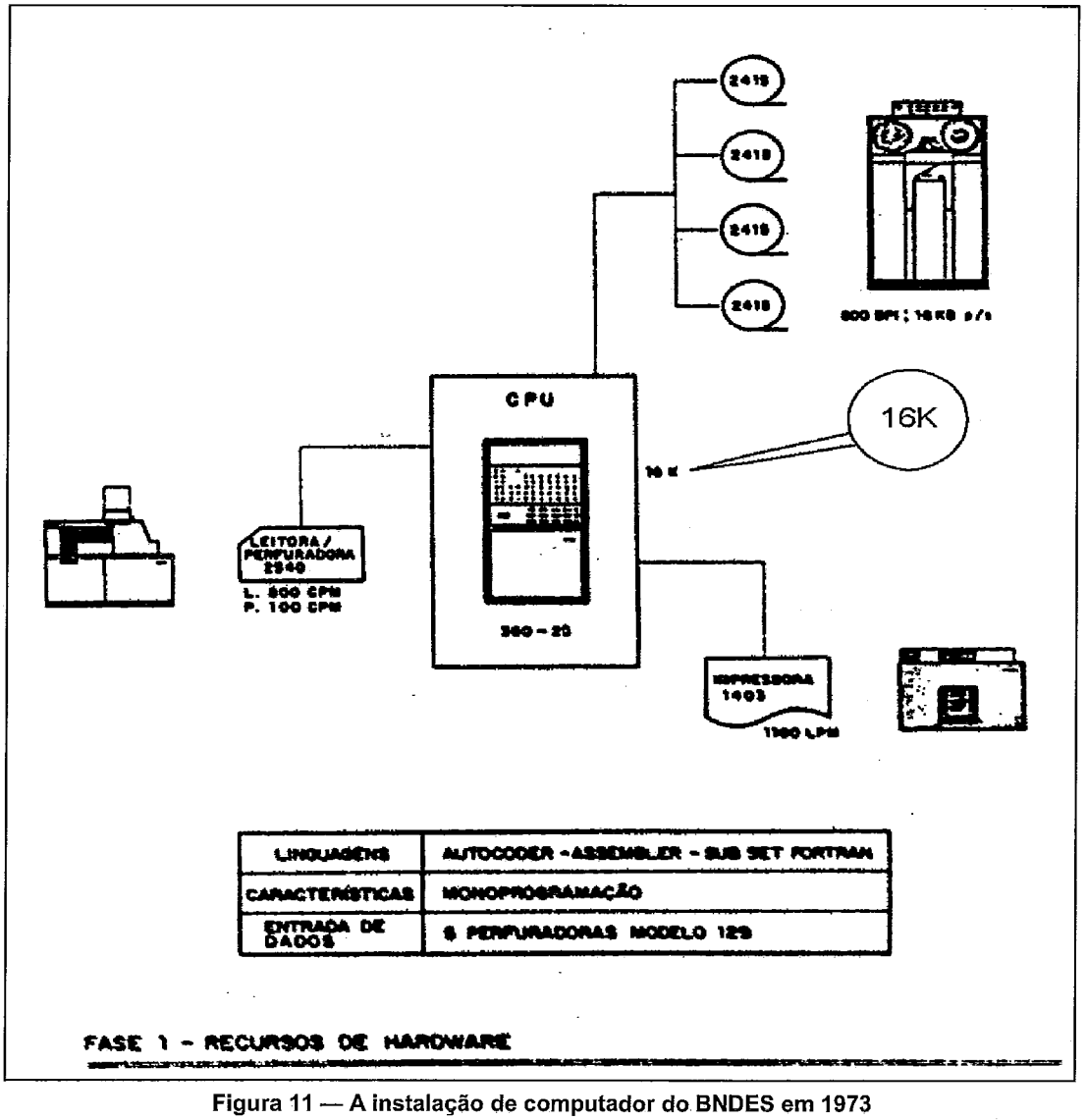


Figura 11 — A instalação de computador do BNDES em 1973
Fonte: PDI-1985, p.37.

[66] Autocoder era o nome do *assembler* fornecido pela IBM para alguns de seus computadores. Tinham esse nome porque foram os primeiros a prover facilidades de macro — simplificadamente, a substituição de um determinado padrão de caracteres por um outro.

Em 1973, os sistemas desenvolvidos eram tipicamente administrativo-financeiros, caracterizados pelo uso de arquivos seqüenciais (não se utilizava bancos de dados); por nenhuma integração entre si; por documentação deficiente; e pela não participação dos usuários em sua concepção. Na informática, organizacionalmente com *status* de divisão, estavam lotados 28 profissionais: 1 chefe de divisão, 2 chefes de seção, 9 analistas / programadores, 5 operadores, 4 digitadores, 5 controladores e 2 profissionais que desempenhavam outras funções. (PDI[67]-1985, p.36-38).

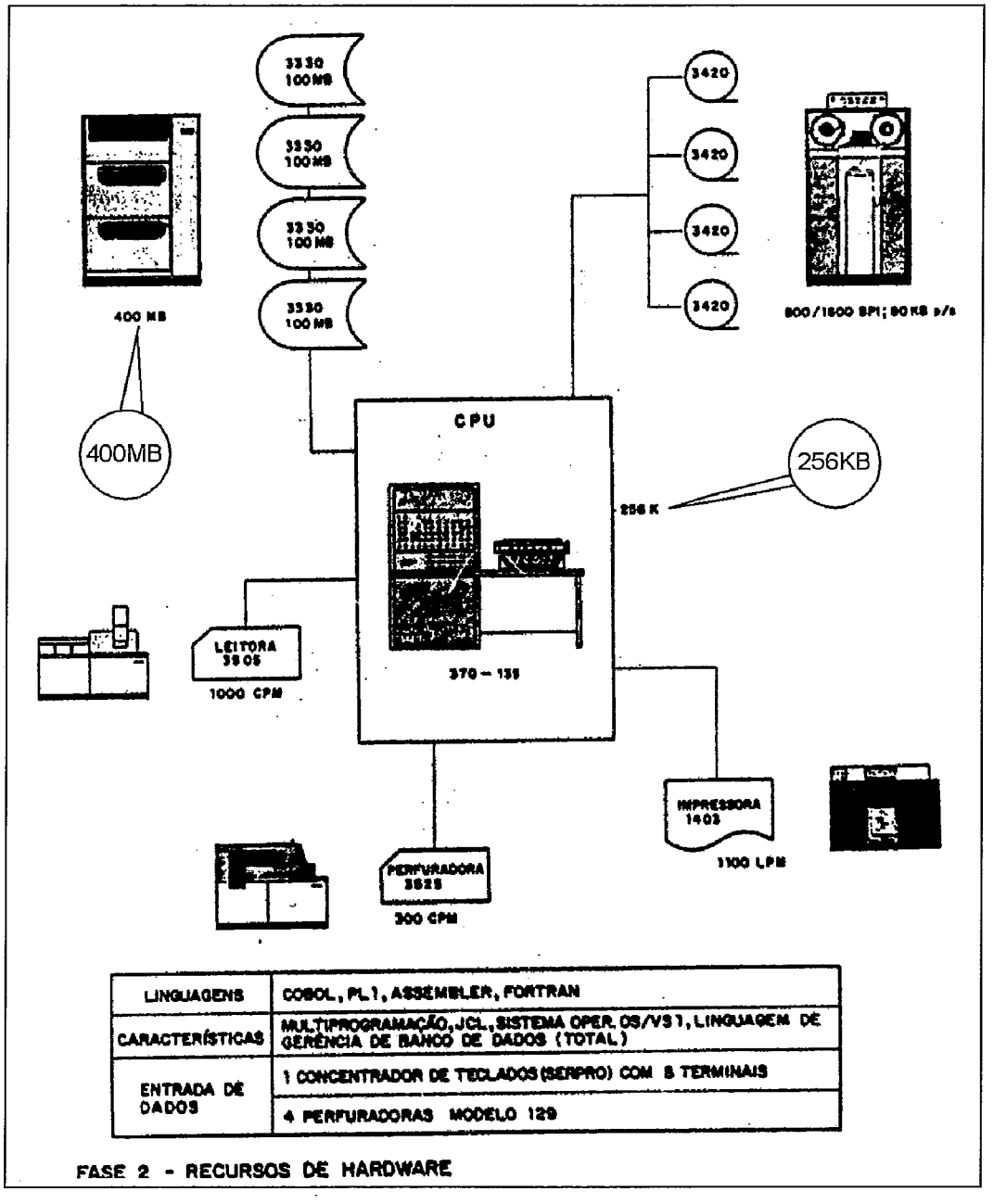


Figura 12 — A instalação de computador do BNDES em 1978
Fonte: PDI-1985, p.41.

[67] PDI — Plano Diretor de Informática

Em 1978, o computador já havia evoluído para um *mainframe* com 256Kbytes de memória principal e 400Mbytes de memória em disco (Figura 12). A linguagem COBOL já estava disponível, mas também eram utilizadas PL1, Assembler e FORTRAN, em um ambiente de multiprogramação. Nesta época, já eram utilizados bancos de dados, no caso, o sistema TOTAL de gerenciamento de bancos de dados hierárquicos. Existiam apenas 5 terminais de vídeo para acesso ao *mainframe*.

A ênfase no desenvolvimento de sistemas ainda recaía sobre as áreas de administração e finanças. Os sistemas passaram a utilizar bancos de dados e linguagens de programação de alto nível. De forma manifesta, deveria ser buscada a participação dos usuários na concepção dos sistemas, bem como ser gerada a documentação formal para os sistemas. Nesta época a informática do BNDES já havia progredido de *divisão* para *departamento*, o Departamento de Sistemas (DESIS) vinculado à Área de Serviços Gerenciais[68]. Passaram a estar lotados, 71 profissionais: 1 chefe de departamento, 3 gerentes, 4 coordenadores, 30 analistas / programadores e técnicos de nível médio, 11 operadores, 9 digitadores, 5 controladores e 8 profissionais executando outras funções. (PDI-1985, p.39-42).

Somente em 1982 iniciou-se a disponibilização, para os usuários finais, de terminais de vídeo para acesso ao *mainframe*, constituindo-se assim o embrião de um processo de descentralização dos recursos de informática. Naquela ocasião, o *mainframe* dispunha de 1Mbyte de memória principal e de 1Gbyte de memória em disco. Em 1985, já existiam 78 terminais de vídeo, além de 1 minicomputador COBRA-530 e 7 minicomputadores interligados ao *mainframe* (Figura 13). (PDI-1985, p.44, 87).

A ênfase que se buscou dar ao desenvolvimento de sistema a partir de 1985 passou a ser a disseminação dos sistemas de informações para todas as áreas do BNDES, e não apenas as de administração e finanças. Nesse momento, 100 profissionais estavam lotados na informática: 1 chefe de departamento, 3 gerentes, 5 coordenadores, 49 analistas / programadores e técnicos de nível médio, 11 operadores, 10 digitadores, 5 controladores e 16 profissionais executando outras funções (PDI-1985, p.43,45).

[68] Decisão de Diretoria Nº. 475/75.

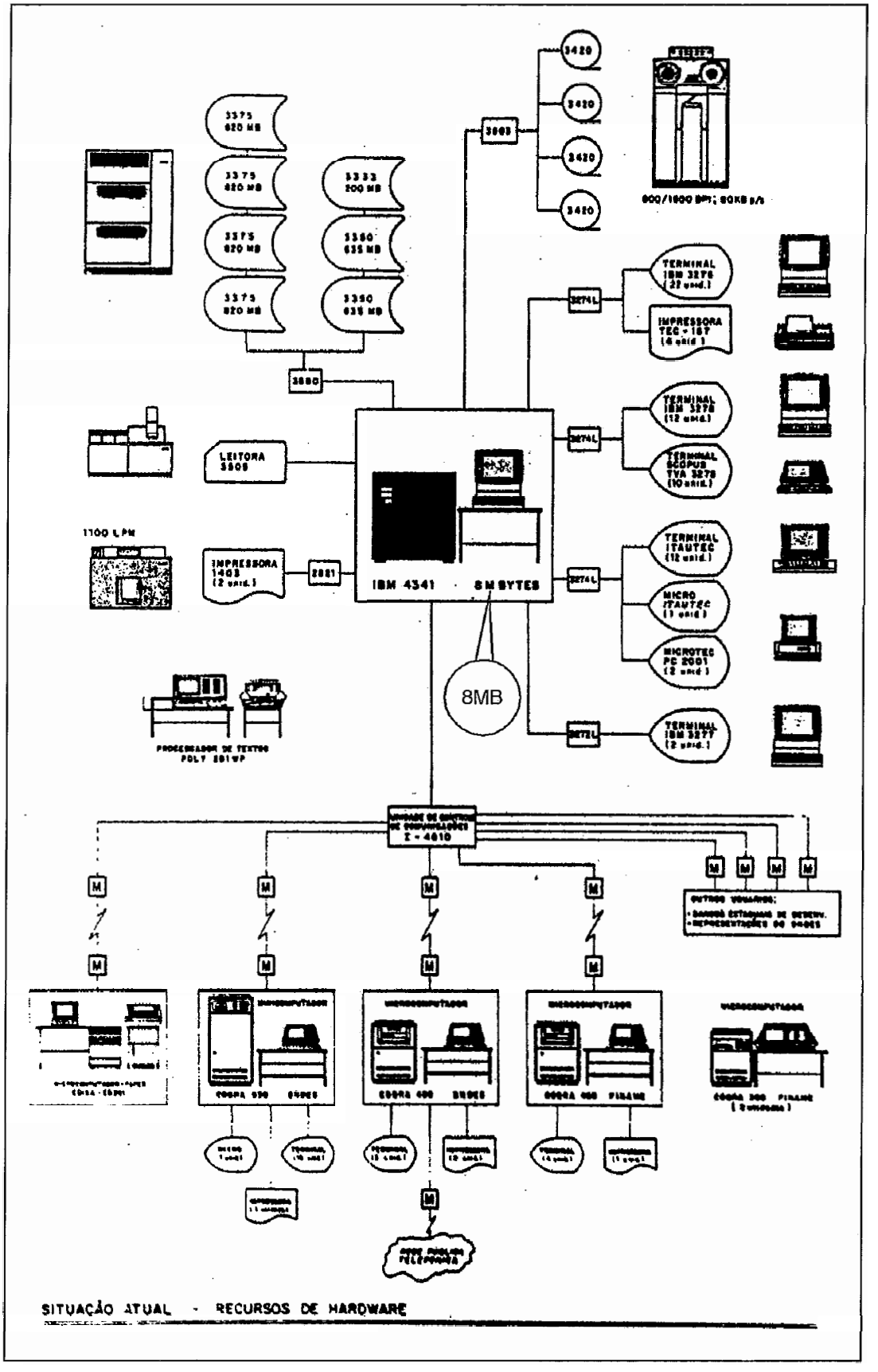


Figura 13 — A instalação de computador do BNDES em 1985
Fonte: PDI-1985, p.52.

A Figura 14 mostra a evolução do hardware do BNDES entre os anos 1978 e 1984.

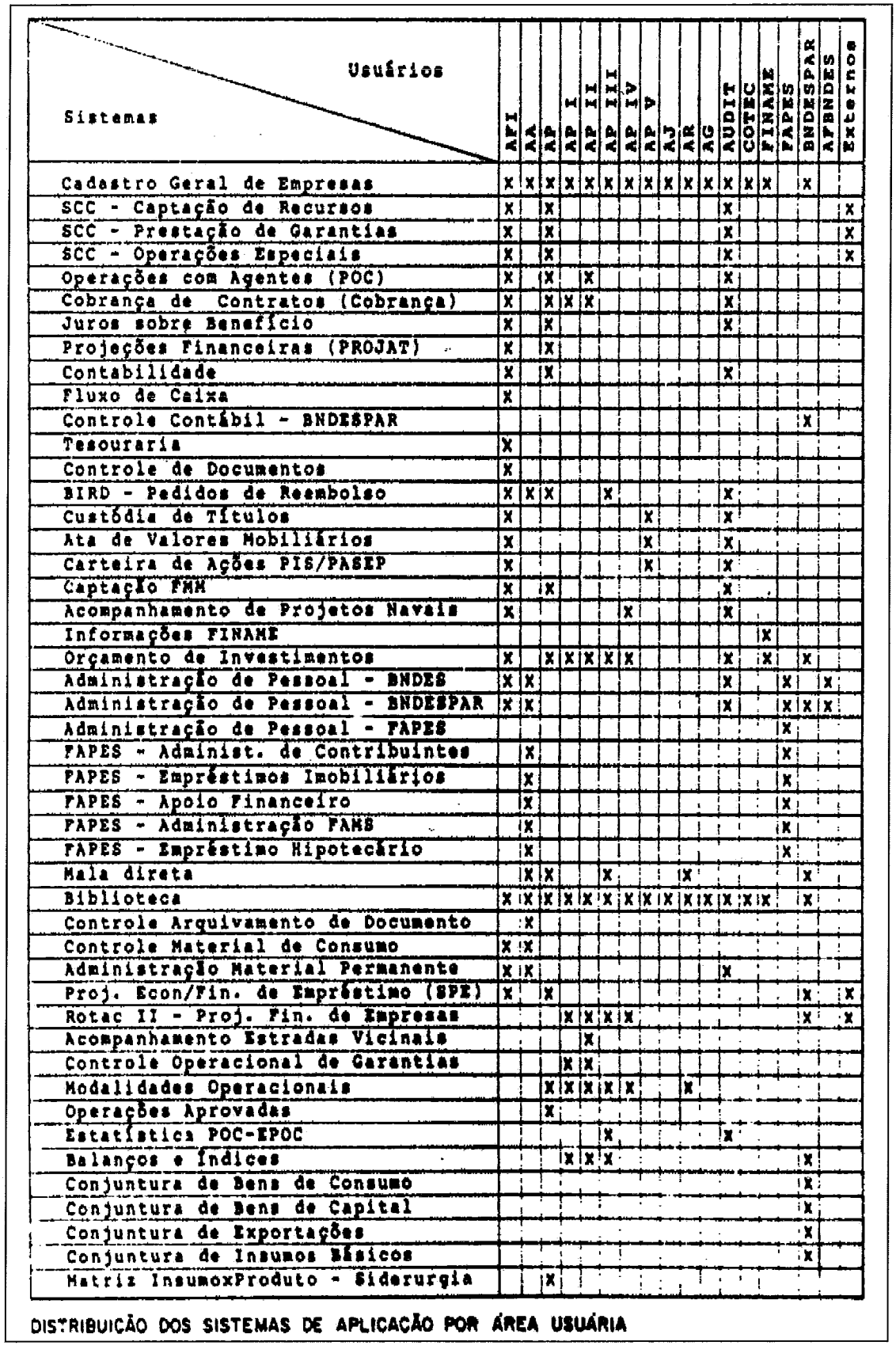
	78	84
Unidade Central	IBM - 370/148	IBM 4341 - MG2 Cobra 530
Discos	4 x 3330	4 x 3330 2 x 3350 4 x 3375
Fitas	4 x 5420	4 x 5420
Impressoras	2 x 1403	2 x 1403
Terminais de Vídeo	5	78
Terminais Impressora	-	6
Microcomputadores	-	1 ED 281 (FAPES) 2 PC 2001 1 Cobra 210 2 Cobra 480 1 Itautec I-7000

EVOLUÇÃO DO HARDWARE - 1978 a 1984

Figura 14 — Evolução do hardware do BNDES entre 1978 e 1984
Fonte: PDI-1985, p.87.

Em 1985, "*o BNDES possuía mais de uma centena de sistemas implantados, sendo os mais significativos (com mais de 15 programas necessários para execução) os demonstrados*" na Figura 15. Nesta época, grande parte da alocação da mão-de-obra de análise e programação, cerca de 60%, estava alocada à manutenção dos sistemas existentes (PDI-1985, p.29). As áreas usuárias listadas nas colunas da Figura 15 são: AFI — Área Financeira e Internacional; AA — Área de Administração; AP — Área de Planejamento; AP I, II, III, IV e V — são Áreas de Projeto, que focalizavam setores específicos da indústria e de negócios; AJ — Área Jurídica; AR — Área de Representações (o BNDES, à época, tinha representações em Washington, São Paulo e Brasília); AG — Área de Assuntos com o Governo; AUDIT — Auditoria Interna; COTEC — Consultoria Técnica (ligada à Presidência do BNDES). Além delas, constam as empresas FINAME e BNDESPAR, a Fundação de Assistência e Previdência dos Funcionários do BNDES e a Associação dos Funcionários do BNDES.

A Figura 16 dá uma idéia do esforço da informática em abranger todas as áreas de negócio do BNDES, entre 1979 e 1984.



Sistemas \ Usuários	API	AA	AP	AP I	AP II	AP III	AP IV	AP V	AJ	AR	AG	AUDIT	COTEC	FINAME	FAPES	BNDESPAR	AFBNDES	Externos
Cadastro Geral de Empresas	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		
SCC - Captação de Recursos	X		X									X						X
SCC - Prestação de Garantias	X		X									X						X
SCC - Operações Especiais	X		X									X						X
Operações com Agentes (POC)	X		X		X							X						
Cobrança de Contratos (Cobrança)	X		X	X	X							X						
Juros sobre Benefício	X		X									X						
Projeções Financeiras (PROJAT)	X		X															
Contabilidade	X		X									X						
Fluxo de Caixa	X																	
Controle Contábil - BNDESPAR																X		
Tesouraria	X																	
Controle de Documentos	X																	
BIRD - Pedidos de Reembolso	X	X	X			X						X						
Custódia de Títulos	X							X				X						
Ata de Valores Mobiliários	X							X				X						
Carteira de Ações PIS/PASEP	X							X				X						
Captação FMM	X		X									X						
Acompanhamento de Projetos Navais	X						X					X						
Informações FINAME														X				
Orçamento de Investimentos	X		X	X	X	X	X					X		X		X		
Administração de Pessoal - BNDES	X	X										X			X		X	
Administração de Pessoal - BNDESPAR	X	X										X			X	X	X	
Administração de Pessoal - FAPES															X			
FAPES - Administ. de Contribuintes		X													X			
FAPES - Empréstimos Imobiliários		X													X			
FAPES - Apoio Financeiro		X													X			
FAPES - Administração FAMS		X													X			
FAPES - Empréstimo Hipotecário		X													X			
Mala direta		X	X			X				X						X		
Biblioteca	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		
Controle Arquivamento de Documento		X																
Controle Material de Consumo	X	X																
Administração Material Permanente	X	X										X						
Proj. Econ/Fin. de Empréstimo (SPE)	X		X													X		X
Rotac II - Proj. Fin. de Empresas				X	X	X	X									X		X
Acompanhamento Estradas Vicinais					X													
Controle Operacional de Garantias				X	X													
Modalidades Operacionais			X	X	X	X	X			X								
Operações Aprovadas			X															
Estatística POC-EPOC						X						X						
Balanços e Índices				X	X	X										X		
Conjuntura de Bens de Consumo																X		
Conjuntura de Bens de Capital																X		
Conjuntura de Exportações																X		
Conjuntura de Insumos Básicos																X		
Matriz InsumoxProduto - Siderurgia			X															

DISTRIBUIÇÃO DOS SISTEMAS DE APLICAÇÃO POR ÁREA USUÁRIA

Figura 15 — Principais sistemas existentes em 1985 no BNDES
Fonte: PDI-1985, p.20.

Área/Departamento	1979	1984
1. Área Financeira/Internacional	53,6	37,5
DEFIN	29,2	22,8
DEPCO	17,2	11,3
DEPIN	1,7	1,1
Uso Geral	5,5	2,3
2. Área de Administração	13,9	11,0
DERHU	12,5	8,08
DEPAD	1,4	2,92
3. Área de Planejamento	3,3	8,2
4. Áreas de Projetos (AP-I, AP-II, AP-III, AP-IV e AP-V)	5,8	12,3
5. Departamento de Sistemas	7,2	13,6
6. FAPES	6,1	6,1
7. FINAME	9,5	3,4
8. BNDESPAR	0,6	7,9
	100,0	100,0

EVOLUÇÃO DA DISTRIBUIÇÃO DOS RECURSOS DE INFORMÁTICA

Figura 16 — Distribuição dos recursos de informática nas áreas do BNDES
Fonte: PDI-1985, p.15. [69]

Por último, como exemplos de aplicativos que instrumentavam os desenvolvedores de sistemas no BNDES, temos (PDI-1985, p.53ss):

- TOTAL 6.3 — Sistema de gerenciamento de bancos de dados;
- STELA-B — Sistema de teleadministração de banco de dados TOTAL, que permitia consultas ao banco de dados sem programação. Desenvolvido pela CONSULPUC (PUC/RJ);
- DICCAM — Dicionário de Campos, desenvolvido pelo Departamento de Sistemas do BNDES (DESIS), que efetuava atualizações e críticas genéricas no banco de dados TOTAL a partir de transações definidas pelos usuários. Possuía ainda um dicionário de campos e algumas funções de dicionário de dados, como geração de comandos para criação de estruturas de dados no banco.

[69] Os departamentos listados na Figura 16 são: Departamento Financeiro (DEFIN), Departamento de Contabilidade (DEPCO), Departamento Internacional (DEPIN), Departamento de Recursos Humanos (DERHU), Departamento Administrativo (DEPAD).

- COM-PLETE — Monitor de teleprocessamento com recursos para desenvolvimento de programas, gerenciamento de arquivos, processamento *batch*, consultas e atualizações *online* de arquivos.
- GSSUPDTE — Utilitário desenvolvido no DESIS BNDES que permitia o gerenciamento de bibliotecas de programas fonte.

Com essas informações é possível ter uma breve idéia do crescimento da instalação de informática do BNDES até meados dos anos 1980. Com o aumento do poder do hardware, aumentou também a quantidade de software disponibilizado para as áreas de negócio. Tudo acompanhado do aumento de quantitativo e do inevitável aumento de complexidade da instalação como um todo.

4.1.4 O desenvolvimento de software

Observar os esforços de desenvolvimento de software que aconteceram no período revela três características marcantes, que serão discutidas a seguir: (i) os sistemas desenvolvidos eram do tipo *transacionais*; (ii) esses sistemas tinham suas interfaces em papel; e (iii) era viável aplicar a racionalidade de maximização da eficiência de uso dos recursos computacionais. Essas três características viabilizavam uma quarta característica não menos importante, a dos analistas de sistemas conseguirem desenvolver suas atividades de projetação com grande autonomia, de forma bastante isolada da influência dos usuários.

Diferente de hoje em dia, os sistemas desenvolvidos naquela época automatizavam atividades que existiam, praticamente da mesma forma, independente da utilização ou não do sistema. Hoje os sistemas de informações são onipresentes, com uma interdependência muito forte entre as organizações e as Tecnologias de Informação (TI), ou Tecnologias de Informação e Comunicação (TIC), de forma que vários processos de negócio (e muitas vezes até as próprias empresas, como a Amazon.com) existem somente porque são viabilizados pela TI. Sem a TI esses processos de negócio (e empresas) não existiriam, razão pela qual se considera a TI insumo estratégico na maioria dos segmentos de negócio atuais. Então, se lá no BNDES dos anos 1970 as atividades de negócio já existiam antes mesmo dos sistemas que as automatizariam, hoje atividades e sistemas sofrem uma construção conjunta, de influência mútua no estabelecimento dos processos de negócio.

> É truísmo dizer que os computadores se tornaram ubíquos nas organizações atuais. Desde sua aplicação no processamento de dados administrativos em meados dos anos 1950, eles viraram um dos instrumentos chave para a

> melhoria das atividades de processamento de informações das organizações. Em menos de quatro décadas, sistemas de informações baseados em computadores evoluíram do suporte a sistemas de *back office*, já formalizados anteriormente, como folha de pagamento, até invadirem a organização totalmente.[xlix] (HIRSCHHEIM; HEINZ; LYYTINEN, 1995, p.1).

Imersos na realidade atual de TI como insumo estratégico, tendemos a interpretar que sistemas de informações são somente aqueles que suportam a gestão das organizações. Basicamente seriam os sistemas que auxiliam nos diversos níveis de tomada de decisão dentro da estrutura organizacional. A eficiência e qualidade deste tipo de sistema reflete-se diretamente na própria eficiência e resultados da empresa. Contudo, este não é o único tipo de sistemas de informações que existe, embora, atualmente, tenhamos esta impressão por conta de sua onipresença e visibilidade, posto que são mediadores, virtualmente, de toda interação das empresas com seus clientes, fornecedores, colaboradores, etc. Quais seriam os outros tipos de sistemas?

A disciplina de Sistemas de Informações tradicionalmente classifica os sistemas em três categorias, chamando de *sistemas de apoio à gestão* o tipo presentemente mais visível. Outros dois tipos seriam os *sistemas de apoio à decisões operacionais* e os *sistemas processadores de transações*, ou, simplesmente, *sistemas transacionais*, o mais rudimentar deles. (BIO, 1985, p.34ss).

O benefício que se obtém com os sistemas transacionais é a redução de custos operacionais através da substituição de mão-de-obra, viabilizada pelo uso dos rápidos recursos de processamento de dados. Quando se implementava esta, digamos assim, "primeira geração" de sistemas de informações, os transacionais, dificilmente se vislumbrava a possibilidade de integração entre eles e a possibilidade, cada vez maior, de se obter informações gerenciais para suporte à tomada de decisão. Com o tempo, essas informações passaram a ser reconhecidas, elas próprias, como um dos mais valiosos ativos das empresas (BIO, 1985, p.115).

Hoje parece estranho reconhecer o processamento de dados do BNDES dos anos 1970 como um mero serviço administrativo, apenas uma atividade de apoio, tarefa burocrática como outra qualquer, porque não temos mais como referência primeira os sistemas transacionais. Ao contrário, conforme já comentado, se pensarmos em sistemas de informações, tenderemos a pensar logo nos sistemas de apoio à gestão. Nos anos 1970, os processos de negócio estavam estabelecidos independente do suporte informático. O objetivo que se perseguia ao implementar um sistema de informações transacionais era automatizar atividades preexistentes, que manipulavam informações eminentemente operacionais. No primeiro momento, as atividades praticamente não eram transformadas com a introdução daqueles

sistemas. As atividades preexistiam aos sistemas e continuavam, no curto prazo, existindo da mesma forma após a implementação deles, os quais visavam apenas registrar e processar os dados delas *a posteriori.*

Uma informação operacional, gerada por um sistema informatizado ou não, tem como finalidade viabilizar o prosseguimento das tarefas executadas no ciclo operacional da empresa (BIO, 1985, p.120). Por exemplo, um funcionário do BNDES, após verificar a situação de adimplência de uma empresa, registrada em um relatório impresso, poderia prosseguir, ou não, com a operacionalização da liberação de recursos de financiamento para essa empresa. Outro exemplo: uma listagem com todos os salários devidos no mês permitia que todos os empregados fossem pagos. Se o tal relatório de adimplemento ou a tal listagem de salários foi produzida ou não por um sistema informatizado não faria a menor diferença, naquele momento, para a operacionalização das respectivas situações.

O fato de as atividades no BNDES dos anos 1970 existirem independentes do sistema que as automatizavam permitia, efetivamente, que os requisitos pudessem ser *levantados* — arrolados — a partir de uma realidade preexistente na empresa. A dinâmica dos negócios àquela época validava a premissa de que as atividades que seriam automatizadas não iriam mudar, pelo menos não no decurso da construção e operacionalização do sistema. Situação bem diferente dos dias atuais, onde aceita-se a impossibilidade de elicitação completa dos requisitos antes da construção do sistema, visto que ele próprio, inserido no contexto organizacional, muda imediatamente o processo de negócio/atividades que suporta. Desta forma, é razoável aceitar que o sistema nunca estará totalmente concluído, demandando continuamente revisões e adaptações para suportar o negócio/atividades que ele mesmo, por sua vez, acabará ensejando que mudem. (SCACCHI, 2004). Porém, no cenário dos anos 1970, a situação era diferente, pois as atividades existiam antes da implementação do sistema e não eram por ele alteradas de forma imediata. As atividades continuavam do mesmo jeito, sendo apenas registradas *a posteriori* no sistema quando ele estivesse em operação. Em um cenário de relativa independência entre atividades e os sistemas que cuidavam de registrá-las, era possibilitado ao analista de sistemas resgatar e formalizar requisitos preexistentes, a partir do conhecimento dos usuários e de suas observações sobre a execução das atividades.

O analista de sistemas Oswaldo Fonseca, que ingressou no BNDES no início dos anos 1980, revela que um outro aspecto importante sobre os sistemas daquela época era que "*a interface com os usuários era* [toda em] *papel. O usuário entrava com uma ficha de entrada de dados* [, com informações relativas às transações ocorridas para posterior digitação,] *e recebia* [após o processamento] *os relatórios* —

só papel" (entrevista concedida ao autor em 21.06.2006). Como visto na seção 4.1.3, até o início dos anos 1980 sequer havia terminais de vídeo disponíveis para os usuários. Muito diferente da interatividade de que hoje dispomos com as interfaces gráficas, a parte visível daqueles sistemas transacionais para seus usuários — fossem eles provedores ou receptores das informações operacionais — era toda em papel. Ou seja, a interface daqueles sistemas era materializada nos formulários de entrada de dados e nos relatórios de saída. Naquele tempo, o computador nem mesmo ficava no mesmo prédio em que funcionavam as áreas administrativa e financeira, as grandes usuárias dos sistemas no BNDES. O papel tinha uma relevância singular tanto na utilização quanto no desenvolvimento de sistemas daquela época.

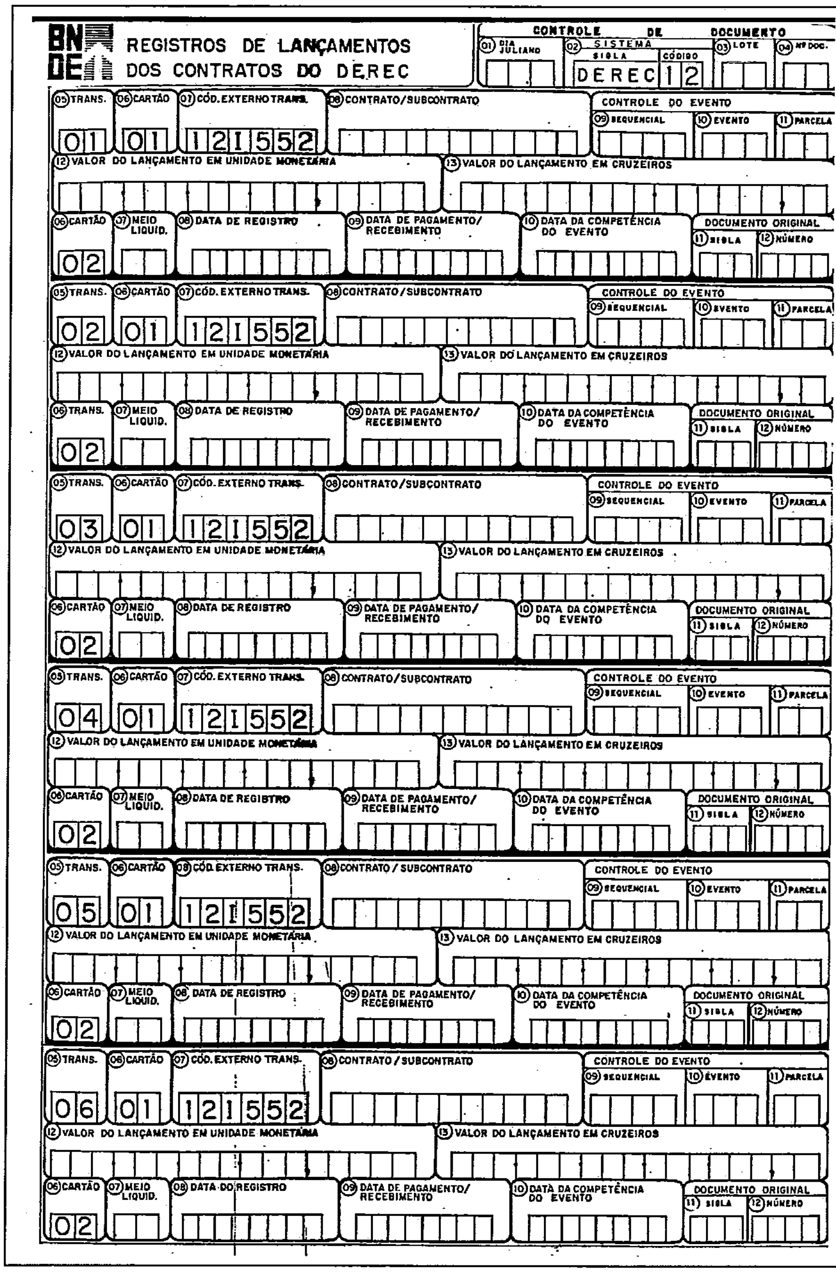
Um bom exemplo de sistema da época com "interface em papel" é o Sistema de Controle de Contratos (SCC), desenvolvido por Osmar Frota. Este sistema entrou em produção em abril de 1978 e está em operação até os dias de hoje, sendo responsável pelo controle de quase a totalidade dos ativos do BNDES. Todas as operações de inclusão, alteração, exclusão e consulta de dados neste sistema invariavelmente ocorriam com a intermediação de formulários e relatórios em papel.

As chamadas "fichas de entrada de dados" padronizavam a forma dos usuários operacionalizarem inclusões, alterações e exclusões de dados, no caso da Figura 17, relativas a operações de financiamento do BNDES. Estas fichas serviam de base para a geração de cartões perfurados. Em um segundo momento, os cartões perfurados foram substituídos por informações gravadas em fitas magnéticas, também geradas a partir das fichas de entrada de dados. Os cartões perfurados, ou a fita magnética, quando processados, comandavam a realização das operações de inclusão, alteração e exclusão nas bases de dados do SCC.

O SCC processava os contratos cadastrados em sua base de dados, além de calcular todo o fluxo de retorno financeiro de cada contrato e emitir os avisos de cobrança a serem enviados aos mutuários e à tesouraria, para que esta operacionalizasse os recebimentos. Inúmeros relatórios eram impressos e colocados à disposição para os diversos usuários do SCC, dentre eles aqueles que cuidavam do relacionamento com os mutuários, aqueles responsáveis pelo cadastramento das informações sobre os contratos, e a tesouraria que se responsabilizava pela operacionalização da cobrança. Todos eles realizavam suas atividades dependentes de informações registradas no SCC, consultando os inúmeros relatórios em papel que este sistema emitia.

Ressalta-se ainda que diferente de hoje, àquela época, com um cenário de negócio mais estável e sistemas *batch* transacionais com interface em papel, era possível a construção de "protótipos" — na verdade, relatórios de entrada e saída —

que de antemão materializavam exatamente a interação que o usuário teria com o sistema que seria construído. Aqueles protótipos em papel eram extremamente eficientes, pois cumpriam muito bem seu dever de reduzir, ao máximo, o espaço para interpretações equivocadas, por parte do analista, sobre o que o sistema *deveria* fazer, e, por parte do usuário, sobre o que o sistema *iria* fazer. Conseqüentemente, a elaboração do processamento que transformaria as entradas em saídas podia ser isolada, ficando sob controle e responsabilidade exclusivos do analista de sistemas, que gozava de ampla autonomia para projetar a solução mais adequada, segundo seu julgamento.


BNDE REGISTROS DE LANÇAMENTOS DOS CONTRATOS DO DEREC

CONTROLE DE DOCUMENTO
01 DIA JULIANO | 02 SISTEMA SIGLA CÓDIGO DEREC 12 | 03 LOTE | 04 Nº DOC.

05 TRANS. 01 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 CARTÃO 02 | 07 MEIO LIQUID. | 08 DATA DE REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

05 TRANS. 02 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 TRANS. 02 | 07 MEIO LIQUID. | 08 DATA DE REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

05 TRANS. 03 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 CARTÃO 02 | 07 MEIO LIQUID. | 08 DATA DE REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

05 TRANS. 04 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 CARTÃO 02 | 07 MEIO LIQUID. | 08 DATA DE REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

05 TRANS. 05 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 CARTÃO 02 | 07 MEIO LIQUID. | 08 DATA DE REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

05 TRANS. 06 | 06 CARTÃO 01 | 07 CÓD. EXTERNO TRANS. 121552 | 08 CONTRATO/SUBCONTRATO | CONTROLE DO EVENTO: 09 SEQUENCIAL | 10 EVENTO | 11 PARCELA
12 VALOR DO LANÇAMENTO EM UNIDADE MONETÁRIA | 13 VALOR DO LANÇAMENTO EM CRUZEIROS
06 CARTÃO 02 | 07 MEIO LIQUID. | 08 DATA DO REGISTRO | 09 DATA DE PAGAMENTO/RECEBIMENTO | 10 DATA DA COMPETÊNCIA DO EVENTO | DOCUMENTO ORIGINAL: 11 SIGLA | 12 NÚMERO

Figura 17 — Ficha de entrada de dados do Sistema de Controle de Contratos
Fonte: Manual do Sistema, 1979.

Atualmente é difícil conseguir protótipos tão eficientes como foram aqueles protótipos em papel, menos em decorrência da técnica de prototipação do que por conta da atual dinâmica dos negócios e da diferença entre os complexos sistemas atuais e aqueles sistemas simplesmente transacionais. Corroborando a idéia de Scacchi (2004) sobre a impossibilidade da completa elicitação dos requisitos dos sistemas, James Herbsleb (2005) reconhece que a própria introdução do sistema / protótipo na organização enseja uma transformação dos processos de negócio, realimentando transformações no próprio sistema que, mesmo já estando implantado, talvez ainda nem esteja totalmente desenvolvido. Diferente do que foi possível no BNDES dos anos 1970, onde os primeiros sistemas não mudaram de imediato os processos de negócio preestabelecidos, os sistemas e organizações atuais exercem restrições mútuas, de tal forma que não é mais possível imaginar uma situação que não seja a de uma evolução conjunta, concomitante, das estruturas "técnicas" — os sistemas — e dos processos de negócio da organização. Por isso, não é tão simples atualmente a obtenção de protótipos que encerrem, de forma congelada, todos os requisitos que o sistema real deverá atender.

Qual racionalidade norteava o julgamento do analista ao projetar a solução para aqueles sistemas do BNDES nos anos 1970? A resposta tem que levar em conta que o domínio da computação era algo bastante esotérico à época, bem diferente dos tempos atuais em que muito mais gente pode elaborar e implementar algoritmos de forma bastante amigável até em editores de texto. Durante a projetação do sistema — dos "mecanismos" que iriam transformar as entradas em saídas —, o usuário era deixado de lado[70] e o foco podia ser concentrado nos aspectos tecnológicos, pois a funcionalidade já estava plenamente representada na especificação. Portanto, o analista se valia de, e era restrito por, uma racionalidade decorrente dos altíssimos custos das instalações de computação e de suas limitadas capacidades de processamento e armazenamento. Não raras vezes, as necessidades dos usuários eram sobrepujadas pela necessidade de otimização do uso dos recursos computacionais. O analista de sistemas devia buscar sempre elevar a taxa de utilização dos recursos disponíveis. Essa racionalidade de desenvolvimento de sistemas levava o analista a ponderar mais sobre os recursos computacionais que existiam do que critérios hoje em dia naturalizados como facilidade de uso e interfaces amigáveis, mesmo porque a interface dos sistemas, como visto, era em papel.

[70] O fato do usuário poder ser "deixado de lado" refletia a visão taylorista de separação entre concepção e adoção, já criticado à época pela antiga escola sociotécnica da Inglaterra e dos países escandinavos.

Basta especularmos sobre as conseqüências de uma racionalidade dessas para entendermos porque, dentre outras funções especializadas propostas para a divisão de processamento de dados, vemos a função de *analista de procedimentos manuais*, cujo papel seria prover "*procedimentos manuais de* ***apoio aos sistemas mecanizados*** *e para dar assistência ao treinamento do pessoal das organizações usuárias na utilização de novos sistemas mecanizados*" (BOOZ ALLEN, 1972, p.430, grifo nosso). Além disso, a organização típica da equipe de projeto na fase de *operação paralela e implantação*, conforme ilustrado na Figura 9, consistia de um analista de sistemas sênior (líder do projeto), de um representante da organização usuária, de *quatro auxiliares de administração (da organização usuária)*, de um analista e um programador (ibid., anexo XXIX). Quatro auxiliares de administração e uma função exclusiva para os procedimentos manuais de *apoio aos sistemas* dão conta do quanto esses sistemas antigos voltavam-se bem mais para o uso dos recursos computacionais do que para as necessidades dos usuários.

Em resumo, a prática de desenvolvimento de software no BNDES dos anos 1970 consistia no julgamento da viabilidade do sistema, onde, basicamente, o analista de sistemas entendia os requisitos que o usuário estava demandando e, com ele, definia as entradas e saídas. Em seguida, o analista especificava, sem a participação do usuário, o processamento necessário e os testes, encaminhando a especificação para a programação, controlando hierarquicamente o desenvolvimento através das especificações. Quando o programador terminava a codificação, testes adicionais eram executados pelo próprio analista e pelo supervisor de desenvolvimento de sistemas — o responsável na informática pelo suporte à função de negócio — que, por fim, liberava o sistema demandado para o usuário. Na voz de alguns analistas de sistemas que viveram aquela experiência:

> O analista [de sistemas] definia com o usuário o que ele iria receber, desenhava o documento de entrada e desenhava o documento de saída, num gabarito de espacejamento. Mostrava tudo que ia acontecer, o protótipo estava ali, tudo era papel. [...] Uma coisa forte que tinha era a especificação do programa, porque passávamos a especificação para o *pool* de programadores. Aí a especificação tinha que ser formal, passávamos as especificações de programas e os casos de testes. [Com o termo "formal", o entrevistado denota que a especificação precisava estar em um nível de detalhe e precisão tal que viabilizasse o trabalho do programador, sem que ele necessitasse fazer levantamentos adicionais com o usuário] (Oswaldo Fonseca, em entrevista concedida ao autor em 21.06.2006).

> Durante os anos 70 o banco tinha uma metodologia de desenvolvimento de sistemas, [...] e os desenvolvimentos eram todos baseados numa metodologia. [...] Tinha uma fase de definição de programas onde, numa folha de papel, especificávamos os objetivos do programa, *input*, *output* e procedimentos. Todos os programas eram definidos e entregues para os programadores fazerem. Nós testávamos [exaustivamente]: tinha teste feito pelo programador, outros pelo analista, outros pelo coordenador [supervisor de desenvolvimento de sistemas]. Não se passava um programa para produção se não fosse muito testado. O coordenador [também] testava o

> sistema, era obrigado a fazer isso, pois ele fazia a reunião com o usuário e os analistas [para a entregar o sistema e obter o aceite do usuário]. (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

Em seu relato, o analista de sistemas Melvyn Cohen transparece o viés pelo atendimento às necessidades do negócio, buscando-se em primeiro lugar a eficiência na utilização dos recursos computacionais, ficando em segundo plano as necessidades dos usuários. Ele descreve seu envolvimento pessoal na implantação do referido Sistema de Controle de Contratos, o SCC.

> Um dia chegaram à conclusão que o sistema 22 [o primeiro sistema construído no BNDES para suportar as atividades de cobrança] precisava ser mudado [, pois não estava mais sendo capaz de suportar as novas formas de financiamento operadas]. O [Osmar] Frota propôs substituir o sistema 22 pelo sistema 43 [, o SCC, que acabara de desenvolver e] que começava a atender à cobrança na parte do *Proálcool*[71] [o sistema 22 fora incapaz de controlar os financiamentos relacionados ao Proálcool]. Mas o pessoal da Cobrança tinha muita resistência, pois o 22 era um sistema específico feito para a Cobrança, para a área financeira, tinha a cara da área financeira. O sistema 43 embaralhava um pouco a cabeça do pessoal, porque a nomenclatura que utilizava era diferente. O gerente da cobrança da época, João Paixão, falava que os relatórios [do sistema 43] eram coisa de japonês: tudo igual mas ninguém entendia. Eu ajudei muito a desmistificar a dificuldade de uso do [sistema] 43. Quando o Luís Felipe [da Rocha] assumiu a gerência [de cobrança], um pessoal mais novo, com outra cabeça, acabou amolecendo a resistência da área financeira e implantamos o sistema 43, que está aí até hoje. (Entrevista concedida ao autor em 20.06.2006).

Na mesma linha, refletindo sobre um momento posterior, já no final dos anos 1980, Renato Gouvea também expressa a prioridade pela eficiência no uso dos recursos sobre as necessidades dos usuários, contribuindo inclusive de forma negativa para a imagem da informática:

> A crítica que existia era que o DESIS [Departamento de Sistemas e Processamento de Dados] era muito lento e que *impunha* os sistemas aos usuários. O DESIS se acostumou, numa época em que o usuário tinha muito pouco conhecimento, a impor muito. (Entrevista concedida ao autor em 26.06.2006).

4.2 A estabilização de uma rede sociotécnica

Uma avaliação retrospectiva do período, entre meados dos anos 1970 e meados dos anos 1980, dá conta que tratou-se de um momento em que a informática do BNDES operou com um alto nível de organização, padronização e produtividade. Mesmo reconhecendo que a complexidade tecnológica, organizacional e dos negócios naquela época era bem menor, é coerente considerar aquela experiência como bem sucedida, posto que, com relativamente pouca gente, com uma máquina limitada e

[71] O Programa Nacional do Álcool, ou *Proálcool*, criado em 1975 pelo governo brasileiro, tinha o objetivo de estimular a produção do álcool para atender as necessidades do mercado interno e externo, bem como à política de sua utilização como combustível automotivo.

num prazo também relativamente curto, foram desenvolvidos sistemas que entraram em operação efetiva[72]. Os depoimentos dos que participaram daquela experiência corroboram esta visão:

> Se formos considerar o que existia, em termos de informática, antes e o que existe hoje, antigamente era muito mais organizado, muito mais, [relata Vânia Carvalho, analista que ingressou no BNDES no início dos anos 1980]. (Entrevista concedida ao autor em 10.07.2006).

> Os programas eram muito bem documentados, tinha uns padrões para você documentar — colocar um título, um *header* dizendo quem fez o programa, qual foi a última vez que foi alterado. Existiam algumas práticas institucionalizadas. [...] [Por exemplo,] o [Osmar] Frota seguia uma metodologia própria que funcionava, [...] tinha sistemas fáceis de manter, modulares, documentados. Tinha uma equipe muito reduzida e muita demanda, e conseguia responder [bem], pois tinham muita eficiência. (Renato Gouvêa, em entrevista concedida ao autor em 26.06.2006).

> Quando entrei no banco, me disseram: olha, você vai usar e definir os sistemas utilizando o tal do DICCAM [vide página 82]. [...] Tinha-se um método e a produtividade era enorme. Senão, [se não tivesse sido assim,] como conseguimos desenvolver tantos sistemas naquela época com uma máquina tão limitada que tínhamos e com tão pouca gente? Fico imaginando como o banco estaria hoje se os grandes sistemas que temos hoje não tivessem sido desenvolvidos na época em que tínhamos um mínimo de metodologia. A base estrutural de todos os sistemas grandes do banco foi feita naquela época. (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

> [Sidney Franco, analista que também veio para o BNDES no início dos anos 1980, afirma que] existiam padrões de utilização dos artefatos que os analistas e programadores tinham para desenvolver os sistemas. Existia o MIT [Manual de Instruções Técnicas, explorado mais adiante,] que continha orientações básicas e métodos de utilização para a execução das atividades. Então, todo mundo executava os trabalhos "da mesma forma", [...] era tudo conhecido, então tinha-se uma maior produtividade. (Entrevista concedida ao autor em 01.08.2006).

Alguém conhece algo somente "*'sobre a base de um determinado estado de conhecimento'; ou melhor, 'como membro de um meio cultural determinado', ou, o melhor de todos, 'em um estilo de pensamento determinado, em um determinado coletivo de pensamento'.*" (FLECK, 1986, p.86). Como engenheiros de software, estaremos mais acostumados a interpretar o sucesso resultante nesta experiência tutelados pelo *estilo de pensamento*[73] que norteia a engenharia de software. Este estilo de pensamento, o modernismo, como visto, com uma postura de fragmentação e simplificação, nos conduziria a admitir uma segregação ou classificação existente *a priori* entre "elementos, ou componentes, do conteúdo" e "elementos, ou componentes, do contexto". O conteúdo explicitamente seria enquadrado com um viés

---

[72] Cumpre reconhecer a relatividade existente entre as categorias de sucesso e fracasso que, portanto, podem variar dependendo do enquadramento escolhido. O quadro traçado, sobre o BNDES nos anos 1970, visa a suportar o argumento de que aquela prática de desenvolvimento de software pode servir de inspiração para uma engenharia de software modernista. Uma história que buscasse pontuar o "fracasso" poderia escolher, por exemplo, um enquadramento que destacasse a ausência de uma voz mais ativa por parte dos usuários da informática enfatizando os confrontos e contradições que surgiram naquela prática.

tecnocêntrico, e o que escapasse do quadro transbordaria para o contexto, normalmente os ditos elementos sociais, organizacionais, culturais. No caso narrado uma classificação candidata poderia ser:

Quadro 1 — Contexto e conteúdo separados

Conteúdo "técnico"	• segregação e prescrição apropriada de funções; • metodologia (um processo) de trabalho;
Contexto "social"	• estrutura organizacional bem definida e adequada; • adequação dos profissionais (formação / motivação).

Surge aparentemente uma "fórmula" para "difundir" aquela organização, padronização e produtividade, qual seja, a de garantir o "contexto social" apropriado à implantação do "conteúdo técnico", este último capaz de replicar os efeitos esperados, uma vez que supostamente detém, intrinsecamente, as qualidades necessárias para lográ-lo. Em outras palavras, para obtenção dos efeitos esperados, bastaria "implantar" um esquema que, *a priori*, prescrevesse apropriadamente as funções/papéis e uma metodologia de trabalho que coordenasse e otimizasse a execução das atividades. Dispondo do "patrocínio necessário", o esforço de implantação desse esquema seria o de estabelecer a estrutura organizacional adequada e garantir o cumprimento das prescrições prévias através de coisas como envolvimento, comprometimento e participação dos envolvidos. Essa é a que se pode chamar perspectiva difusionista do estilo de pensamento hegemônico na engenharia de software, como abordado na seção 3.3.

Em alternativa a essa divisão simplificadora entre contexto e conteúdo, o caso narrado pode ser investigado à luz dos *Estudos CTS (Ciência-Tecnologia-Sociedade)*, na versão da *Teoria Ator-Rede* (TAR). A teoria ator-rede não reconhece nos agentes sociais entidades dotadas de uma essencialidade, mas sim reconhece-os como redes moldadas por relações heterogêneas. Para a teoria ator-rede, no Quadro 1 estão listados elementos que, associados a diversos outros, compõem a rede sociotécnica da qual resulta o efeito de alto nível de organização, padronização e produtividade que existiram. Trata-se de uma perspectiva bastante diferente de vê-los como contexto e conteúdo separados, existentes *a priori* e dotados de essências explicativas ou causais da organização em estudo. Embora gere uma explicação mais complexa, a idéia das redes sociotécnicas também gera explicações melhores do que quaisquer simplificações em fatores segregados. Prosseguindo com a narrativa, será possível observar algumas contingências que existiram e a atuação de alguns atores

[73] Vide nota 1.

delimitando o espaço de atuação de outros, o que resultou na estabilização da rede, ou, o que dá no mesmo dizer, resultou no efeito de organização, padronização e produtividade observados.

Uma contingência bastante relevante que não caberia em nenhum fator único e segregado, uma vez que, por exemplo, repercute na organização das equipes, conforma relações entre vários atores, e influi no uso da metodologia, é que quase tudo ainda estava por ser feito na informática do BNDES naquele período: "[havia] *um grande 'campo verde'*[74] *que não tinha nada, pois o lado operacional do banco não era contemplado pela informática, nem mal nem bem, não tinham nada,* [sistema algum]" (Oswaldo Fonseca, em entrevista concedida ao autor em 21.06.2006). Essa situação fica bem evidenciada na sugestão de quantitativo mínimo de analistas e programadores, contida nas recomendações para a função de processamento de dados. Para projetos de novos sistemas eram recomendados 12 profissionais (5 analistas e 7 programadores), já para esforços de manutenção, como quase não haviam sistemas já desenvolvidos, bastariam apenas 3 profissionais (1 analista e 2 programadores), conforme registrado no Quadro 2.

Quadro 2 — Quantitativo mínimo para o desenvolvimento de sistemas

	Recurso de mão-de-obra necessários (Quantidade de pessoas)	
	Analistas de Sistemas	Programadores
Duas equipes de projeto para o desenvolvimento de novos sistemas importantes.	4	5
Duas equipes de projeto para desenvolvimento de [novos] sistemas menores.	1	2
Uma equipe de projeto para manutenção de sistemas mecanizados existentes.	1	2
Total	6	9

Fonte: Booz Allen (1972, p.428).

Na abordagem de desenvolvimento de sistemas apresentada, o analista de sistemas, conforme descrito na seção 4.1, atuava com grande autonomia por conta da atuação de outros atores enredados. Por exemplo, o *mainframe* e as restrições tecnológicas e financeiras justificavam a racionalidade de otimização dos recursos computacionais, mesmo em detrimento das necessidades dos usuários, garantindo ao analista autonomia de decisões durante a projetação do sistema. Mas, para isso, era imprescindível que atuassem, eficientemente, os protótipos em papel dos sistemas transacionais, que permitiam construir, através das especificações, representações eficazes dos sistemas que seriam implementados. Essas especificações, por sua vez, possibilitavam aos analistas um exercício hierárquico de controle e coordenação das

atividades de desenvolvimento de sistemas. Outro ator que propiciava aquela atuação dos analistas era o próprio "campo verde" (ou "terra virgem"), pois, quando já existem sistemas e necessidade de manutenção, torna-se diferente a atuação dos analistas, e de vários outros atores, tais como usuários, programadores, especificações e metodologia. Mas, por que isso ocorre? Fora da situação de "terra virgem", os analistas passam a atuar em atividades mais curtas, de manutenção de diversos sistemas, sem a estruturação característica de projetos de novos desenvolvimentos. Na manutenção, atividades de adaptação e correção de falhas dificilmente ensejam a produção de especificações e a obediência à metodologia. Sem a "terra virgem", ao executar manutenções, o analista tornava-se um "faz tudo", semelhante ao que passou a ocorrer a partir dos anos 1960 no cenário norte-americano no qual, como já foi citado, "[os analistas] *freqüentemente participavam de diversos aspectos do* [ciclo de] *desenvolvimento de sistemas, desde o levantamento dos requisitos até o projeto* [*design*] *e sua implantação.*" (ENSMENGER; ASPRAY, 2002, p.142). A analista de sistemas Vânia Carvalho expõe essa característica quando relata que, já no final dos anos 1980, sem a "terra virgem", numa situação em que praticamente todos se ocupavam da manutenção, os analistas atuavam de maneira bem distinta da época em que os sistemas foram desenvolvidos, não seguindo mais a metodologia que, portanto, àquela altura, não atuava mais na rede.

> [Já no final dos anos 1980, época em que] praticamente todo mundo era absorvido na manutenção dos sistemas, não é que não se queria usar a metodologia, é que não tinha muita coisa nova para se fazer e [, portanto, conseguir] utilizar a metodologia [que havia sido utilizada quando os sistemas foram inicialmente desenvolvidos]. (Entrevista concedida ao autor em 10.07.2006).

Era esperado do usuário que atuasse participativamente na etapa de definição conceitual geral do sistema (vide Figura 9). Além disso, era esperado também que ele aceitasse a autonomia do analista na concepção da solução e que utilizasse os sistemas à medida que fossem disponibilizados.

Aceitamos com naturalidade que o usuário deve participar para que seja viável o desenvolvimento de sistemas. A participação do usuário é considerada o segundo ponto mais importante nos projetos de software, perdendo apenas para o suporte da alta administração, conforme tabulado no relatório *Extreme Chaos*[75] do *Standish Group International*. Contudo, na prática, conseguir essa participação por muitas vezes não é fácil. Esse mesmo relatório aponta que, tradicionalmente, a causa número um das falhas nos projetos é justamente a falta de envolvimento dos usuários.

---

[74] "Campo verde": vide nota 7.
[75] *Extreme Chaos*: vide nota 39.

Porém, naquela informática do BNDES nos anos 1970, os usuários atuaram da maneira esperada — participativamente, *na etapa apropriada* —, como atesta a matéria apresentada a seguir, extraída do informativo interno da Associação de Funcionários do então BNDE. Assim como a própria informática, o Departamento Financeiro (DEFIN) encontrava-se em franca reestruturação em 1975, em busca de maior eficiência e segurança no controle interno e gestão financeira. Para isso, envidava esforços para dotar o BNDES de três novos sistemas — contabilidade, custódia de títulos e cobrança.

> Para a realização destas inovações, todo o Departamento vem-se movimentando em ritmo acelerado e alguns de seus funcionários têm até se deslocado para outros setores, provisoriamente, a fim de desenvolver com mais eficiência o trabalho programado: é o caso, por exemplo, do *coordenador da Contabilidade e do chefe da Cobrança, que estão alocados no Departamento de Sistemas*, há algum tempo, trabalhando 'duro' no desenvolvimento dos novos sistemas, em conjunto com os técnicos do DESIS. (AFBNDE INFORMA, 1975, grifo nosso).

Vários atores contribuíram para que os usuários atuassem como esperado. Por exemplo, a mobilização do corpo funcional do BNDES, em apoio à reestruturação em curso, aliada à situação de "terra virgem", favoreceram a participação dos usuários na etapa de definição conceitual geral dos sistemas. Com a inexistência de sistemas, o usuário vivencia e detém total conhecimento sobre as atividades que realiza. No momento em que já existe um sistema informatizando suas atividades, com freqüência ocorre uma expropriação de parte do conhecimento do usuário, que torna-se *registrado*[76] no próprio sistema. Com o passar do tempo, é comum, inclusive, que os usuários percam o conhecimento acerca de suas próprias tarefas automatizadas, não participando mais tão efetivamente na etapa de levantamento dos requisitos. Como atestam Osmar Frota e Renato Gouvea: "*os analistas de sistemas, por conta da metodologia* [de desenvolvimento que havia], *tornaram-se os grandes especialistas do banco* [em diversas áreas do negócio]." (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

> [...] Antigamente, o camarada começava a falar sobre o problema e o analista já sabia mais sobre o problema que o usuário. O usuário só trazia o abacaxi, o analista era quem o descascava todo. (Renato Gouvêa, em entrevista concedida ao autor em 26.06.2006).

Seguindo o "*script* de sua atuação esperada", após participarem da definição conceitual do sistema, os usuários conformavam-se com a autonomia do analista em conceber a solução. Eles se resignavam com a utilização do sistema "possível" de ser implementado, segundo o julgamento do analista, por conta do esoterismo do

[76] Ao invés de *registrado*, seria mais apropriado utilizar o termo *inscrito*, derivado do conceito de *inscrição* conforme definido a seguir, na página 100.

desenvolvimento de sistemas e da injunção das restrições tecnológicas e econômicas impostas pelo *mainframe*. Uma situação socialmente aceita, portanto, em decorrência da atuação dos mais diversos atores, como, por exemplo, protótipos eficientes, interfaces em papel, *mainframe*, atividades estáveis, "terra virgem", especificações, metodologia.

E a atuação esperada do *mainframe*? Essa pergunta pode causar estranheza, pois é mais comum pensar apenas na atuação dos elementos humanos. No entanto, os *Estudos CTS* tratam humanos e não-humanos com equivalência analítica. Logo, o *mainframe* tinha sim que perfazer uma atuação esperada, a saber, a de operar dentro da faixa de custo e performance aceitáveis. Para isso, era necessário, novamente, a atuação de outros atores, como as especializações de funções e profissionais (seção 4.1.1). Por sua vez, os profissionais dependiam de intermediários para a consecução de quase todas as tarefas que demandavam. Como já foi mostrado, até mesmo para uma simples compilação de um programa, dependeriam, pelo menos, do perfurador de cartões e do operador. Configuravam-se interesses recíprocos entre os profissionais, a organização e o *mainframe* que viabilizavam a existência / aceitação de hierarquias e pontos de controle explícitos, favorecendo, no conjunto, a atuação esperada de todos eles, para que o custo de operação da instalação fosse aceitável e as tarefas que todos demandavam chegassem a bom termo. O relato do analista de sistemas José Gonçalves nos permite entrevê-lo:

> As ferramentas antigamente eram as máquinas de escrever e os desenhistas — os camaradas das pranchetas — faziam toda a documentação, DFD's[77], etc. O que acontece com isso? Você cria especialistas, seja para desenhar, seja para explicitar o desenho. O método é necessário porque você tem que passar [a especificação, a tarefa] de uma pessoa para outra que era leiga. Então, naturalmente, você tem uma tendência de criar um controle de qualidade no próprio desenhista. O desenhista não vai querer fazer cada DFD de um jeito, ele próprio é um padronizador. Quando você passa a ter uma ferramenta que você [mesmo] faz e desfaz como quiser, imprime em 5 minutos e mostra, você cria para cada um a facilidade dele fazer do seu jeitinho. Isso tem impacto grande no processo de trabalho. (Entrevista concedida ao autor em 02.08.2006)

As diversas funções especializadas — analistas, programadores, documentadores, operadores, digitadores, perfuradores de cartões, preparadores de lotes, supervisores, auxiliares de administração, ... — encontravam guarida na estrutura organizacional que, por sua vez, se fortalecia em decorrência dos resultados viabilizados por todas as funções. O MIT (Manual de Instruções Técnicas), citado por Sidney Franco (p. 91), dependia do interesse dos envolvidos que, para atingirem seus próprios objetivos, aproximavam-se do MIT, o que resultava em menor esforço dos

profissionais para realizarem algumas de suas tarefas e uma utilização mais eficiente dos recursos da instalação.

Ao examinarmos uma rede sociotécnica, então, não importa a suposta essência que os elementos detém, mas sim as atuações que perfazem. De que adiantaria, por exemplo, partir de uma interpretação de usuário, de analista, de metodologia, ou de qualquer outro ator envolvido na prática de desenvolvimento de software, como entidades fixas, "essenciais", com características preestabelecidas, se suas atuações podem mudar à medida que se relacionam com outros atores? Essas atuações sim são importantes, pois podem contribuir para estabilização ou não da rede sociotécnica. No caso visto, o sistema quando passou a existir conferiu à rede uma nova dinâmica de *performances* dos usuários, e também de analistas, programadores e metodologia, dentre outros, cujo efeito foi uma prática de desenvolvimento de software no final dos anos 1980 bem diferente da ocorrida na situação de "terra virgem" dos anos 1970. Mais do que enumerar todos os possíveis atores enredados, o que importa para compreendermos a ordem que resulta da estabilização de uma rede sociotécnica é a dinâmica das associações entre tais atores quando, mutuamente, garantirem que cada um perfaça suas atuações esperadas. (LAW, 1992).

À medida que atores entram e saem da rede, ela continuará estabilizada somente se suas atuações continuarem favoráveis, e/ou se novos atores tiverem alianças suficiente fortes para perfazerem suas próprias atuações. Além disso, a rede precisa materializar novos elementos para manter-se estabilizada. No caso observado, vê-se, dentre outros, um conjunto de sistemas operantes, produtividade, eficiência e o MIT materializados na rede.

Dentre os diversos atores que compuseram a rede citada, o MIT (Figura 18) se destaca por conta de sua singularidade. Já em sua apresentação, traz registrado o seguinte objetivo:

> O MANUAL DE INSTRUÇÕES TÉCNICAS (MIT) tem por objetivo divulgar internamente os recursos técnicos, normas e padrões existentes em nossa instalação, fornecendo meios para um melhor desenvolvimento dos trabalhos que são aqui realizados. A inclusão de contribuições ao texto deste Manual é franqueada a todos os técnicos do Departamento [DESIS] e deverão ser encaminhadas à GERAD [Gerência de Administração de Dados, que também era responsável pela manutenção do MIT], que, de acordo com o módulo 1.01 da seção USO GERAL, tomará as necessárias providências. (MIT, 198?).

No módulo 1.01 da seção 'USO GERAL' vê-se que a inclusão de matérias no MIT devia obedecer aos procedimentos estabelecidos e ser aprovada pelo "Grupo

---

[77] DFD significa Diagrama de Fluxo de Dados, um dos instrumentos chave da análise estruturada. Outros diagramas importantes nesta abordagem: DER — Diagrama de Entidade e Relacionamento; DTE — Diagrama de Transição de Estados (YOURDON, 1992).

Permanente de Padrões e Normas" (GPPN), instituído com a finalidade de julgar a pertinência e qualidade das matérias. O número de membros do GPPN era igual ao dobro do número de gerências do Departamento de Sistemas. As decisões do GPPN eram resultado do consenso de seus membros, sendo que as reuniões deveriam, obrigatoriamente, contar com a participação de pelo menos 51% das gerências representadas (MIT, 198?).

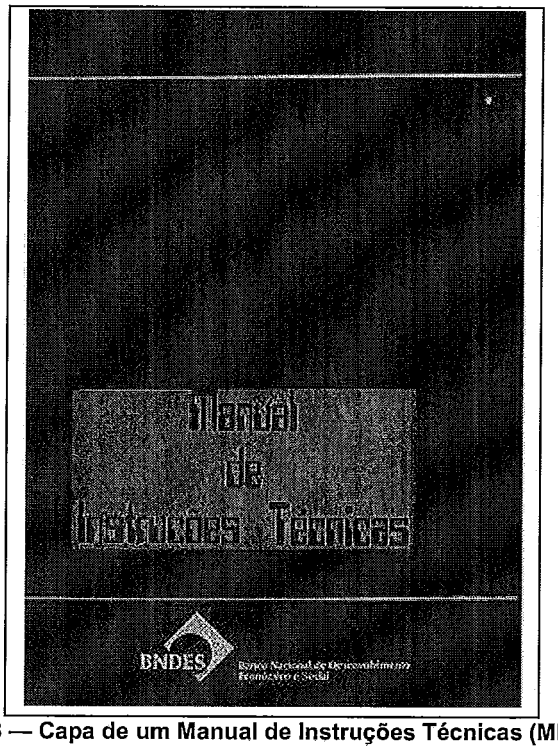


Figura 18 — Capa de um Manual de Instruções Técnicas (MIT, 198?)

Na percepção de Vânia Carvalho o MIT:

> [...] Era um manual só, mas ali tinha o quê? Todas as [rotinas] genéricas que [se] precisava, [a forma de configuração e uso de] todo o ambiente, como é que são as coisas tipo: o JCL[78] é 'tanto', no cartão 'coloca-se não sei o quê'. Em um manual cabia tudo, resumido, [...] com o caminho principal para você chegar aonde queria. [...] E todo mundo achava importante, se você queria saber alguma coisa, ia lá no MIT olhar. [...] As pessoas respeitavam muito [as normas e padrões], inclusive o chefe de departamento da época, Ricardo Farias, entendia aquilo como uma prioridade e uma necessidade. (Entrevista concedida ao autor em 10.07.2006).

O MIT era um ator singular naquela rede. Nenhum outro seria tão capaz de retratar, de maneira simplificada, a própria rede que vimos considerando. Dizendo de outro modo, nenhuma outra entidade melhor retrataria o efeito de organização, padronização e produtividade resultante da estabilização daquela rede, efeito registrado nas normas **(1)**, padrões **(2)** e melhores práticas **(3)** do MIT. Por exemplo:

[78] JCL é o acrônimo de *Job Control Language*, uma linguagem para permitir o controle das tarefas (*jobs*) submetidas ao sistema operacional dos computadores de grande porte (*mainframe*) da IBM. (BROWN, G.D., 1990, JCL Sistema/370. Rio de Janeiro, Editora Campus).

(1) ROTINA DE CATALOGAÇÃO DE PROGRAMAS PARA PRODUÇÃO — MÓDULO 2.07. [...]
1) O formulário "CATALOGAÇÃO DE PROGRAMAS PARA A PRODUÇÃO" será preenchido pelos técnicos, em uma via, e enviado à PROGRAMOTECA. [...]
2) A PROGRAMOTECA dará outro nome ao módulo fonte já existente na SYS1.FONTE.
3) A PROGRAMOTECA compilará o programa, gerando e gravando o módulo em sua Biblioteca específica – D95.GESOP.OBJETO1. [...] (MIT, 198?).

(2) PADRONIZAÇÃO DO FORMATO DE TELAS — MÓDULO 2.06
1) Objetivo: estabelecer um formato padrão para as telas de transações on-line [...].
2) A tela: uma tela pode ser visualizada como uma folha quadriculada de 24 linhas por 80 colunas, correspondendo a um total de 1920 posições de caracteres. [...]
3) Mensagens: a primeira linha da tela destina-se exclusivamente à exibição de mensagem de erro, advertência ou orientação ao usuário, devendo seu conteúdo apresentar as seguintes características: iniciar na coluna 2; possuir tamanho máximo de 79 posições; vir sob forma protegida; ser apresentada com brilho intenso; soar alarme do terminal.
4) Identificação da tela: a identificação da tela será feita nas 2ª e 3ª linhas, conforme se segue.
4.1) Conteúdo da 2ª linha: identificação da Empresa: constante "BNDES"; início na coluna 2; apresentada com brilho normal. [...] (MIT, 198?).

(3) EXPONENCIAÇÃO EM COBOL — MÓDULO 5.03
1) Introdução: especial cuidado deve ser tomado ao se definir em COBOL operações de exponenciação através do comando COMPUTE. Por meio de um exemplo real, demonstramos aqui como um simples programa COBOL pode ocupar a CPU por um tempo exageradamente longo.
2) Demonstração (Figura 19):

```
        BANCO NACIONAL DE DESENVOLVIMENTO ECONÔMICO E SOCIAL

EXPONENCIAÇÃO EM COBOL                                  MODULO 5.03


2. DEMONSTRAÇÃO

PROCEDURE DIVISION
    MOVE    3600    TO    EXPT.
    COMPUTE    X = 1.000013054  **  EXPT.
    STOP RUN.

O programa acima foi executado cinco vezes, utilizando em cada
execução uma definição diferente para a variável EXPT.
Observe-se o tempo de CPU em cada uma das execuções.

a) 01     EXPT      PIC    S9(4).
IEF3731   STEP /GO      /START  77347.1337
IEF3741   STEP /GO      /STOP   77347.1342 CPU  4MIN  06.68SEC

b) 01     EXPT      PIC    S9(4)  COMP.
IEF3731   STEP /GO      /START  77347.1357
IEF3741   STEP /GO      /STOP   77347.1401 CPU  4MIN  05.49SEC

c) 01     EXPT      PIC    S9(4)  COMP-3.
IEF3731   STEP /GO      /START  77346.1851
IEF3741   STEP /GO      /STOP   77346.1856 CPU  4 MIN  04.72SEC

d) 01     EXPT      COMP-1.
IEF3731   STEP /GO      /START 77347.1410
IEF3731   STEP /GO      /START 77347.1410  CPU   0MIN  00.78SEC

e) 01     EXPT      COMP-2.
IEF3731   STEP /GO      /START 77347.1412
IEF3741   STEP /GO      /STOP  77347.1412  CPU   0MIN  00.76SEC
```

Figura 19 — Demonstração de exponenciação em COBOL
Fonte: MIT (198?)

> 3) Nos casos a, b e c o tempo de execução foi 310 vezes maior em relação ao tempo de execução dos casos d e e. Os três primeiros casos foram resolvidos, aparentemente, através de multiplicações sucessivas. Os dois últimos foram resolvidos através de aritmética de ponto flutuante. [...] (MIT, 198?).

O compacto MIT constituiu o que Latour (2000, p.398) chama de *ponto de passagem obrigatório* para a operação daquela instalação de informática do BNDES dos anos 1970. Canalizando para si um movimento de acumulação das experiências locais de operação da instalação e de desenvolvimento de software, o MIT passou a ocupar uma posição privilegiada naquela rede, um aparente centro para onde fluíam representações das diversas práticas locais e de onde emanava uma influência efetiva sobre a realização prática das tarefas. Por isso, por se postar como intermediário entre a prática desempenhada e seu registro, ao mesmo tempo em que também mediava a orientação sobre as tarefas e sua consecução efetiva, o MIT favorecia um certo controle dos atores daquela instalação de informática, ou seja, era um ponto de passagem obrigatório para se atender às necessidades e desafios impostos na ocasião, a saber, o de desenvolver os sistemas necessários, posto que inexistiam, e tornar a instalação economicamente viável, operacional e produtiva.

À medida que passa a centralizar o registro cumulativo das contribuições dos técnicos, bem como as normas e os padrões já estabelecidos, O MIT possibilita uma apreensão sinóptica de toda a dinâmica da instalação. Os *Estudos CTS* esclarecem esse processo através do conceito de *inscrição*, que detém três qualidades preponderantes: *mobilidade*, *estabilidade* e "*combinabilidade*" (*permutabilidade*) (LATOUR, 2000, p.384).

> Inscrição é um termo geral referente a todos os tipos de transformações que materializam uma entidade num signo, num arquivo, num documento, num pedaço de papel, num traço. Usualmente, mas nem sempre, as inscrições são bidimensionais, sujeitas a superposição e combinação. São sempre móveis, isto é, permitem novas traduções e articulações ao mesmo tempo que mantém intactas algumas formas de relação. Por isso são também chamadas "móveis imutáveis", termo que enfatiza o movimento de deslocamento [tradução, vide pag. 28] e as exigências contraditórias da tarefa [alistar e manter o controle dos aliados] (idem, 2001, p. 350).

Todas as normas, padrões e melhores práticas que o MIT acumulava eram inscrições, eram signos que materializavam efeitos diversos existentes naquela rede sociotécnica. Com isso, tais efeitos ganhavam uma certa mobilidade, podendo sair do local em que foram produzidos e alcançar a atuação de outras equipes, que os podiam reaplicar em suas próprias práticas. Sendo *móveis*, as inscrições, para serem úteis, precisam ser *estáveis*, de forma que os elementos que agregam — sejam nas normas, padrões ou práticas recomendadas — mantenham intactas suas relações mútuas, o que permitirá a aplicação do efeito inscrito em outro local.

A posição privilegiada ocupada pelo MIT decorre da possibilidade de *combinação* (a terceira característica) das inscrições que agrega. Todas as inscrições materializam o início e o fim de um ciclo de acumulação — no caso, experiências práticas relacionadas com o uso daquela instalação de informática — em uma escala tal que se torna possível dominá-las e apreendê-las com um único golpe de vista. Nas inscrições, as experiências assumem uma forma plana, no papel, de modo que o MIT as agrupava tornando possível combiná-las entre si, gerando inscrições a partir de outras inscrições mais "básicas". Essa possibilidade ajuda a inverter o equilíbrio de forças entre as complexas tarefas e os profissionais: em um primeiro momento, sem as inscrições, a complexidade das tarefas dominam os profissionais; já em um segundo momento, com a acumulação de inscrições, são os profissionais que dominam as complexas tarefas, materializando os efeitos de organização, padronização e produtividade observados na rede. (LATOUR, 2000, p.369).

O MIT exercia um certo controle sobre a atuação dos diversos outros atores daquela rede porque inscrevia *programas de ação* que antecipavam suas atuações esperadas.

> Programa de ação é um termo da sociologia da tecnologia que tem sido usado para emprestar caráter ativo, e muitas vezes polêmico, aos artefatos técnicos. Cada dispositivo antecipa o que outros atores, humanos ou não-humanos, poderão fazer (programas de ação); no entanto, essas ações antecipadas talvez não ocorram porque os outros atores têm programas diferentes — antiprogramas, do ponto de vista do primeiro ator. Assim, o artefato [o MIT ou alguma de suas normas, por exemplo] se torna a linha de frente de uma controvérsia entre programas e antiprogramas. (idem, 2001, p. 353).

Quando um programa de ação é inscrito numa entidade qualquer, esta torna-se um ator também, impondo seu programa de ação aos outros. Nada garante, *a priori*, que o programa de ação será bem sucedido, pois os demais atores podem escolher não respeitá-lo ou usá-lo de uma forma não prevista, seguindo um antiprograma. Alguns atores conseguem inscrever fortes programas de ação, o que torna mais difícil para os outros não segui-lo. A flexibilidade das inscrições, ou a força de seus programas de ação, pode ser enrijecida através da variação do material que as suporta. Por exemplo, o MIT produziu inscrições mais fortes do que as que seriam produzidas simplesmente através de treinamento dos diversos profissionais, sem a existência de um registro de normas e procedimentos. No entanto, um sistema automatizado que pudesse "controlar todas" as atividades dos profissionais daquela instalação do BNDES nos anos 1970 produziria inscrições menos flexíveis ainda, isto é, tenderiam a garantir mais ainda o cumprimento do programa de ação. (ibid., p.205-206).

O conceito de programas de ação, mais fortes ou mais fracos dependendo da variação do material onde se inscrevem, permite uma melhor compreensão da eficiência dos protótipos em papel utilizados no desenvolvimento daqueles sistemas transacionais. Como as atividades do negócio já estavam inscritas nos diversos formulários e relatórios em papel antes da utilização dos sistemas transacionais, e mantinham-se da mesma forma após a entrada em operação dos sistemas, os protótipos — no caso formulários e relatórios de entrada e saída de dados — inscreviam programas de ação fortes o suficiente para guiar, com bastante fidedignidade e eficácia, o processo de tradução das operações processadas manualmente em operações processadas pelos sistemas transacionais (além de móveis e imutáveis, as inscrições precisam ser *fidedignas* (LATOUR, 2000, p.386)). Hoje em dia, não é fácil inscrever programas de ação tão fortes nos protótipos em meio eletrônico. Ao contrário, no processo de tradução que se verifica desde a elicitação dos requisitos e sua validação através dos protótipos, até a implantação do sistema final, o programa de ação inscrito no protótipo vai sendo transformado à medida que evoluem, concomitantemente, protótipo/sistema e as atividades de negócio que serão suportadas pelo sistema em construção. Diferente de seus ancestrais em papel, os protótipos em meio eletrônico dos sistemas atuais não inscrevem programas de ação tão estáveis e eficazes.

Antes de finalizar a seção, mais um conceito dos *Estudos CTS* será examinado. É que os atores que atuam nas redes são, eles próprios, redes também — daí uma versão dos *Estudos CTS* ser conhecida como *teoria ator-rede*. O argumento, em detrimento de uma suposta essência dos atores, ou de interesses apriorísticos estáveis, decorre de se considerar os atores em relações e atuações com os demais e que, portanto, podem mudar de acordo com uma mudança nessas relações (LAW, 1992). O *mainframe*, por exemplo, é um ator, uma *caixa-preta*, mas é também uma complexa rede sociotécnica que agrega inúmeros componentes de hardware e software, instituições, serviços e profissionais. Basta que ele pare de funcionar em uma instalação para que se entreveja parte da rede que o constitui, através da atuação de diversos profissionais que surgirão para solucionar o problema.

Na maioria das vezes, não atentamos para a rede que constitui um ator, seja ele um objeto como o *mainframe*, uma instituição como o BNDES, ou um processo de desenvolvimento de software bem estabelecido. A teoria ator-rede traz o conceito de *pontualização*, fenômeno relacionado com a necessidade de simplificação, para explicar o ocultamento das redes — multiplicidades — que compõem os atores — que, uma vez pontualizados, são vistos como singularidades.

> Todos os fenômenos são efeitos ou o produto de redes [sociotécnicas] heterogêneas. Mas na prática nós não lidamos com intermináveis ramificações de redes. De fato, na maior parte do tempo não estamos sequer em posição para detectar as complexidades de uma rede. Assim, o que está ocorrendo? A resposta é que se a rede age como um único bloco, então ela desaparece, para dar lugar à sua própria ação e ao aparentemente simples ator que perfaz a ação. Ao mesmo tempo, a maneira como o efeito é produzido também é ofuscado: sob estas circunstâncias isto sequer é visível ou relevante. Desta maneira é que algo muito mais simples — uma televisão funcionando, um banco bem gerenciado, um corpo saudável — vem, por um tempo, mascarar a rede que o produz.[1] (LAW, 1992).

Quanto mais estabilizada uma rede, mais sua atuação parecerá ser resultado do esforço de um ator pontual. Na história da informática do BNDES, em um determinado momento, o efeito de organização, padronização e produtividade observado poderia facilmente ser atribuído ao MIT, um ator que *pontualizava* aquela rede sociotécnica. Como a estabilidade de qualquer rede sociotécnica é precária, requerendo a todo instante ser refeita, se a ordem resultante desaparece, desaparece também o ator que a pontualiza.

4.3 Inspiração para uma engenharia de software moderna

Onde estava a engenharia de software no BNDES dos anos 1970? É bem verdade que como disciplina autônoma, naquela época a engenharia de software estava em seus primeiros passos. Mas o estilo de pensamento da engenharia de software hoje estabelecida faz com que os engenheiros de software esperem narrativas que destaquem a difusão de um "conteúdo técnico" — um modelo ou um padrão implantado —, supostamente existente *a priori*, para explicar o sucesso que se julga ter ocorrido (seção 3.3). De fato, é possível identificar a existência de elementos sugestivos da popular abordagem nos anos 1970 do *Chief Programmer Team*, de Terry Baker (1972). O trabalho era realizado por equipes pequenas, especializadas em determinadas funcionalidades / áreas de negócio, lideradas por um programador / analista chefe que dispunha de suporte para várias tarefas de apoio, como controle de fontes — existia uma "programotecária" — e documentação dos sistemas — existiam desenhistas que geravam fluxogramas e outros diagramas, além de secretárias que datilografavam os manuais —. O estudo de programas fonte dos sistemas daquela época revela uma estruturação *top-down* do desenvolvimento. Tudo isso em um cenário que materializou o objetivo prometido pela metodologia do *Chief Programmer Team*: produtividade. Porém, em momento algum, tanto nas entrevistas quanto no material da época pesquisado, foi possível identificar qualquer movimento de implantação dessa abordagem.

Na seção 4.2, no lugar de uma narrativa simplificadora para explicar a organização, padronização e produtividade do cenário observado, optou-se por explicitar uma rede sociotécnica, na qual contexto e conteúdo foram criados concomitantemente. Como relatado, a organização, padronização e produtividade observadas surgiram como efeito da rede sociotécnica. Um processo de desenvolvimento de sistemas indissociável da prática local foi construído para o atendimento das necessidades e desafios imediatos na ocasião, principalmente, como já foi dito, o de desenvolver os sistemas demandados e tornar a instalação operacional e produtiva.

Curiosamente, e aqui entramos num ponto bastante delicado, o efeito daquela rede sociotécnica favorece a replicação do estilo de pensamento dominante na engenharia de software, o pensamento moderno, que faz esperar explicações simplificadas para os casos tidos como bem sucedidos, baseadas na implantação de "conteúdos técnicos" em "contextos apropriados". O mesmo pode ser afirmado para os projetos "BIG-L" (capítulo 2). Vários princípios do pensamento moderno com sua visão mecanicista e positivista de mundo aproximam o caso local do BNDES dos anos 1970 do caso "universal" dos "BIG-L", nos EUA dos anos 1950.

Em ambos os exemplos, diferente de hoje em dia, o mundo parecia passível de manipulação através de representações modernamente inspiradas, considerando possíveis a existência de nítidas separações — por exemplo, sujeito x objeto, técnico x social, contexto x conteúdo, solução x problema, teoria x prática, sucesso x fracasso — e considerando viáveis princípios como controle centralizado, racionalismo tecnocêntrico, representação e formalização, estabilidade do mundo/objeto manipulado, valorização do método.

Um primeiro ponto de aproximação entre os casos do BNDES anos 1970 e dos "BIG-L" é a reputação de experiência bem sucedida. Como segundo ponto, nota-se em ambos claramente uma visão mecanicista explícita em frases do tipo: "*[...] as instituições responsáveis pela guerra conceberam o problema como se supostamente pudesse ser resolvido em termos* ***mecanicistas*** *da ciência física*"[lii] (EDWARDS, 1997, p.138, grifo nosso); "*Teorias cognitivas, bem como a tecnologia dos computadores, foram primeiramente criadas para suportar a* ***mecanização*** *de tarefas militares anteriormente realizadas por seres humanos.*"[liii] (ibid., p.147, grifo nosso); "*[a] RCA queria um pacote de* software *que permitisse aos programadores produzir seus fluxogramas* ***mecanicamente*** *[...]* "[liiii] (CAMPBELL-KELLY, 2004, p.100, grifo nosso); e "a *organização típica de uma equipe de projeto [do BNDES] durante o*

desenvolvimento de um sistema ***mecanizado*** *[...]*" (BOOZ ALLEN, 1972, anexo XXIX, grifo nosso).[79]

Profissionais formados na prática submeteram-se a uma estruturação *burocrática*[80] para garantir o bom funcionamento do esquema de comando e controle na consecução dos objetivos, tanto dos "BIG-L's", com suas diversas contratadas e subcontratadas, quanto daquela divisão de processamento de dados do BNDES.

A busca pela centralização do controle, que permite o exercício hierárquico e burocratizado do comando, tem forte presença no modernismo. Como materialização dessa busca temos os *centros* de comando do SAGE e o próprio CPD — *centro* de processamento de dados — do BNDES dos anos 1970. A idéia de centralização também está presente nos métodos de controle hierárquico dos projetos que, junto com a segregação de funções, em situação de "terra virgem", são guiados por especificações, como denotam a Figura 1 — Gerenciamento de Projetos no SAGE e a Figura 9 — Etapas do Desenvolvimento de Sistemas no BNDES dos anos 1970, que representam a mesma idéia, embora estejam em níveis de abstração e detalhamento diferentes. É o que Richard Coyne (1995, p.296) chama de orientação por métodos, um procedimento de projetação hierárquico que progride, em seqüências temporais de tarefas, da declaração do problema para o desenvolvimento de especificações, depois fluxogramas, em seguida pseudocódigo, programação, teste, e, finalmente, avaliação. Mas, para isso, tal procedimento requer uma nítida divisão, um nítido isolamento entre problema e solução.

---

[79] A visão mecanicista é tão dominante que até mesmo termos considerados equivocados continuam em uso. É o caso da expressão "*manutenção de software*". Esse termo está consolidado mesmo sendo reconhecido, por vários autores, como inadequado, pois o software não é algo material cujo desgaste requer reparos ou manutenção com reposição de peças (STROUSTROUP, B., 1994, *The C++ Programming Language*. New York, Addison-Wesley, p.381).

[80] Há muito tempo, a atividade humana organizada tende a se estruturar em torno de dois requisitos opostos: a divisão do trabalho em tarefas mais simples e a coordenação dessas tarefas para se atingir o objetivo do trabalho como um todo. Uma das formas mais conhecidas e utilizadas para estruturação das organizações é a burocracia. De inspiração totalmente mecanicista, uma organização burocrática pauta-se no preestabelecimento de regras que tentam determinar *a priori* comportamento e atuação dos envolvidos para se alcançar uma coordenação global das atividades. A premissa é que é sabido, de antemão, o que deve ser feito, existindo poucas incertezas acerca das tarefas a serem desempenhadas. As organizações burocráticas são projetadas visando a eficiência através da redução, ao mínimo possível, das interações diretas entre os atores. A coordenação deve ser garantida por intermédio da prescrição e cumprimento de regras pelos indivíduos ou grupos. Quando a regra não é suficiente, um nível hierárquico de decisão deve arbitrar, de forma que a estruturação hierárquica é vista como uma eficiente maneira de se organizar a comunicação. (DAHLBOM; MATHIASSEN, 1993, p.15-16). Ainda hoje é difícil pensar numa forma de estruturar o trabalho nas organizações diferente da chamada *burocracia mecanizada*. Fusão das idéias de Taylor sobre administração científica e de Weber sobre a burocracia, a burocracia mecanizada surgiu nos primórdios do século XX e garantiu, sob um determinado ponto de vista, um expressivo sucesso das organizações através da busca do melhor desempenho individual e de melhores formas de coordenação organizacional (NADLER, D. A.; GERSTEIN, M. S., 1994, "Projetos de Sistemas de Trabalho de Alto Desempenho: Como Organizar Pessoal, Trabalho, Tecnologia e Informação". In: Nadler et al, *Arquitetura Organizacional*. Editora Campus.).

Considerando o exemplo do BNDES, percebe-se ser possível àqueles analistas de sistemas, sem prejuízo para a utilização dos sistemas que desenvolviam, um posicionamento apartado do mundo no qual atuaria o sistema que seria construído. Eram distintos os mundos do analista — esotérico — e do usuário — cujas atividades preexistiam ao sistema. Isso possibilitava aos atores daquela rede atuarem como se existisse uma separação nítida entre o objeto que seria abordado — as atividades a serem automatizadas — e o sujeito da automação — o analista —. Permitia atuações que também possibilitavam considerar nitidamente separados o problema — automação das atividades — e a solução — sistemas e procedimentos —, o conteúdo — sistema — e o contexto organizacional de sua aplicação.

Se as atuações podiam considerar as separações nítidas, surgia a possibilidade de exercício de uma forte ênfase racionalista, posto que o racionalismo promove e pressupõe uma independência entre a razão e o mundo no qual atuamos. O racionalismo acaba sendo discurso e prática que sustentam o esforço de concepção e projetação de artefatos tecnológicos (COYNE,1995, p.18). Assim, nos "BIG-L's" vemos a aplicação de uma racionalidade técnica "fria", supostamente eximida, separada, das questões éticas inerentes ao desenvolvimento de artefatos bélicos[81]. Já no exemplo do BNDES dos anos 1970, os analistas de sistemas (*seniores* de preferência), sujeitos da razão, projetavam o sistema demandado sem a participação do usuário, respaldados pela racionalidade da computação vigente de maximizar a eficiência na utilização dos recursos computacionais. Eles deviam prover o sistema que melhor atendesse, segundo seus julgamentos, às necessidades formalizadas, pois eram os *experts* em análise e desenvolvimento de sistemas.

> O racionalismo promove uma visão particular sobre os clientes ou usuários finais de sistemas tecnológicos. Sua participação no processo de invenção e projetação de sistemas computacionais é minimizado. Os experts têm acesso às teorias e estão [supostamente] em melhor posição para entregar o projeto mais apropriado.[liv] (COYNE, 1995, p.28).

A atuação sobre as representações, mais do que diretamente sobre o mundo "real", o mundo "lá fora", é clara na simulação da guerra, na praxe modernista subjacente à política da Guerra Fria dos "BIG-L", e na atuação sobre as especificações e protótipos que guiavam o desenvolvimento de software no BNDES

---

[81] A epistemologia e a filosofia da ciência introduziram a clássica separação entre *contexto da descoberta* e *contexto da justificação*. Para os epistemólogos, metodologicamente torna-se imperativo relegar as alianças ou compromissos entre pesquisadores e o poder estabelecido, em particular com os militares, desconsiderando, assim, o que chamam de *contexto da descoberta*. Defendendo que a atividade dos pesquisadores transcende dimensões como autoridade, poder, prestígio, maquiavelismo, tudo presente no processo de produção do conhecimento, o que deve merecer atenção é apenas o chamado *contexto de justificação*, onde os resultados, e não o processo de produção do conhecimento, são discutidos.

dos anos 1970. Além de cumprir o papel de representação, as especificações que guiavam a condução dos projetos cumpriam o papel de materializar também um outro princípio fundamental para o ideal mecanicista, o da formalização clara, precisa, objetiva, de preferência matemática, das representações. Aos analistas de sistemas / programadores, do BNDES e dos "BIG-L'", não bastava fiarem-se numa percepção oral, ingênua e de senso comum, do problema. Ao contrário, seu papel seria o de prover uma representação "objetiva" e "correta" do problema, sobre a qual poderiam atuar "racionalmente".

Por trás das premissas de representação, formalização e racionalismo esconde-se uma noção metafísica, positivista, de um mundo (ou problema) *estável* (COYNE, 1995, p. 289), como pareciam ser o ambiente de negócio/desenvolvimento de sistemas dos anos 1970 no BNDES e o ambiente dos projetos "BIG-L". O primeiro, composto por entidades aparentemente dotadas de essências passíveis de descrição — no caso, atividades preexistentes que seriam automatizadas —. Já o segundo, com objetivos bem definidos e orçamentos suficientes para a consecução dos planos até o fim. A representação dos problemas materializava-se, primeiro, nas especificações dos sistemas e, em seguida, nos próprios sistemas construídos.

David Harvey (2006, p.38) explicita que a racionalidade tão considerada pelo modernismo, com sua visão de mundo mecanicista e positivista, é uma racionalidade tecnocêntrica, uma racionalidade incorporada na máquina, no poder da tecnologia contemporânea. Nas explicações retrospectivas para os casos tidos como bem sucedidos, aparece naturalizada a atribuição do sucesso à aplicação de um conteúdo "técnico" apropriado ao contexto existente. O tecnocentrismo é facilmente resgatado nos casos do BNDES e dos "BIG-L". No caso do BNDES, foram vários os relatos de entrevistados, reproduzidos no final da seção 4.1.4 e início da seção 4.2 (p.90), ressaltando a importância de método, padrões, organização, documentação, especificações formais e testes. Já para o caso dos "BIG-L", podem ser enumerados como reflexo do tecnocentrismo: valorização das técnicas para o gerenciamento dos projetos de programação (projetos de software) e a própria afirmação de Eckert sobre a necessidade de transformar a programação — o desenvolvimento de software — em uma disciplina de "engenharia". Além do mais, como afirma Edwards (1997, p.205), o próprio desenvolvimento do computador foi fecundado pelos militares por conta de seu potencial para automatizar o comando e controle.

Na argumentação acerca do sucesso atingido no BNDES (seção 4.2) chamou-se a atenção para um tipo de explicação que usualmente vem à mente, uma

---

CALLON (2001). Deste modo, exime-se os produtores das possíveis aplicações do conhecimento

explicação simplificadora, tutelada pelo estilo de pensamento dominante, que, em retrospecto, atribuiria o sucesso à própria engenharia de software, hoje já estabelecida. Da mesma forma, o suposto sucesso dos projetos "BIG-L", em retrospecto, também foi reputado à disciplina da engenharia, enfaticamente registrado na afirmação de Benington (1956): o sucesso do SAGE decorreu dos profissionais possuirem formação em engenharia. Como visto no capítulo 2, o sucesso dos "BIG-L" foi um dos motivadores da escolha da engenharia como metáfora para o desenvolvimento de software. Um ponto delicado na argumentação até aqui apresentada, é que as redes sociotécnicas em questão foram capazes de se estabilizar replicando efeitos deste mesmo estilo de pensamento, acusado de produzir explicações simplificadas. Dizendo de outro modo, as redes sociotécnicas consideradas foram capazes de produzir efeitos de um mundo moderno. Portanto, experiências semelhantes a dos "BIG-L" e a do BNDES nos anos 1970, guardadas as devidas proporções, contribuíram para fortalecer a rede sociotécnica da engenharia de software, que, em decorrência, acabou instituindo-se eminentemente modernista.

No próximo capítulo serão explorados dois outros momentos da informática do BNDES. Diferente do que se passou nos anos 1970, nestes dois outros momentos não se verifica a construção da validade das premissas modernas nas respectivas redes sociotécnicas. Os casos que serão apresentados mais parecem trazer uma ruptura do que a continuação da narrativa que veio sendo construída até aqui acerca do estabelecimento / uso da engenharia de software, tamanha a dissonância existente entre a apresentação dos cenários e uma suposta narrativa moderna sobre a aplicação de uma engenharia de software difusionista para solucionar os problemas enfrentados.

---

produzido.

Capítulo 5
Duas Pequenas Histórias

O mote deste capítulo é provocar a respeito do distanciamento, ou ruptura, entre as categorias de teoria e prática na engenharia de software, tais como estabelecidas numa ótica moderna. Bruce Seely reflete sobre esse distanciamento analisando algumas das proposições de David Parnas[82].

> David Parnas [...] comenta sobre as diferenças de interesse entre os desenvolvedores de software, pesquisadores e teóricos. Ele observou aquilo que pensava ser uma diferença substancial entre as atividades relacionadas com o software e o trabalho em outras modalidades de engenharia. "Ao passo que engenheiros encontram coisas de valor nas publicações de pesquisa, o mesmo não ocorre para a maioria dos desenvolvedores de software." Como resultado, Parnas acha que "o maior problema dos desenvolvedores são problemas que nunca tenho considerado, problemas que nenhum de meus professores ou colegas pensou ser merecedor de discussão." Parnas assumiu — incorretamente — que esta distância entre a pesquisa e o mundo real é exclusividade do campo de software. De fato, diferenças como essa têm surgido em vários campos acadêmicos de engenharia desde 1945, à medida que pesquisadores e teóricos trilham caminhos separados.[iv] (SEELY, 2002, p.91).

Serão apresentadas duas narrativas sobre a informática do BNDES que, se não expõem essa ruptura, ou distanciamento, pelo menos subvertem a linearidade simplificadora das narrativas difusionistas usuais sobre projetos de software. Estas últimas, em linhas gerais, destacam o problema existente e apresentam o suposto potencial do modelo "universal" escolhido para solucioná-lo. Normalmente, trazem uma descrição do projeto (plano) de implantação do modelo e, ao final, concluem sobre o sucesso ou o fracasso do projeto. No caso de fracasso, como visto na seção 3.3, a explicação recai, via de regra, sobre os "fatores não-técnicos".

Nas próximas seções, duas *pequenas histórias*[83] retratarão tentativas posteriores do BNDES restabelecer, por intermédio da difusão de modelos "universais", alguns dos efeitos obtidos em sua rede sociotécnica dos anos 1970. A primeira tentativa ocorreu no início dos anos 1990, a segunda, no início dos anos

[82] Vide nota 6.

[83] A teoria ator-rede refuta a idéia de ser possível separar nitidamente a realidade na clássica divisão entre sujeito e objeto, buscando solapar, inclusive, a idéia de que representações fidedignas dos objetos podem ser concebidas. Confrontado com o problema de "representar fidedignamente" a própria teoria ator-rede, o que seria incoerente, John Law (1999) contrapõe as *pequenas histórias* às *grandes histórias*. A teoria ator-rede, através do conceito de tradução, hipertrofia a idéia de que as representações — sejam elas, por exemplo, teorias ou modelos "universais" — acabam por vezes traídas pelos objetos que representam. Por isso, ao invés de buscar histórias únicas, que sumariam e generalizam, supostamente capazes de representar fidedignamente os objetos através de fatorações reducionistas e classificações ou agrupamentos estanques — as *grandes histórias* —, Law propõe as múltiplas *pequenas histórias*, de onde eventualmente podem ser identificados padrões comuns, úteis à compreensão das redes sociotécnicas.

2000. Em ambas as histórias a engenharia de software já existe como disciplina constituída, disponibilizando modelos "universais" cuja implantação supostamente possibilitaria restabelecer o efeito de ordem desejado. Na primeira história, o modelo "universal" em foco foi a *análise estruturada*, na segunda, foi o CMMI.

As narrativas lineares não dão conta da complexa dinâmica existente na prática situada de "implantação" (um termo mais adequado, como já apontado anteriormente, seria *projetação local*) dos modelos "universais". Muito diferente da possibilidade de conseguir isolar o "conteúdo técnico" dos contaminantes do "contexto social", na prática, no caso a caso, contexto e conteúdo estão imbricados. Até mesmo sucesso e fracasso são categorias difíceis de se distinguir nitidamente. Por isso, ao invés de grandes narrativas generalizadoras, as *grandes histórias*, uma aproximação da dinâmica cotidiana dos projetos de software poderia ser facilitada através de sua particularização em *pequenas histórias*.

Por se apresentarem tão diferentes de uma narrativa difusionista, as duas pequenas histórias a seguir parecem trazer uma ruptura com a própria narrativa construída ao longo desta dissertação, parecendo até que teria sido abandonado o tema da engenharia de software. Nelas, pouco aparecem os modelos "universais" considerados, análise estruturada e CMMI, e, mais ainda, as histórias se apresentam de modo muito diferentes entre si: na primeira o autor parece querer se ausentar; já na segunda, ao contrário, o autor parece querer contar sua própria experiência. Feitos tais alertas, sigamos com as histórias.

5.1 Primeira História — Implantação da *análise estruturada* no início dos anos 1990

A partir de meados dos anos 1980, vários sistemas de software atuavam no BNDES (vide Figura 15), alguns já com quase dez anos de operação. Conseqüentemente, inúmeras atividades da empresa tornaram-se dependentes da informática. Bastante diferente do ocorrido na década anterior, passou a existir uma maior e mais difundida percepção sobre a necessidade de utilização dos insumos informáticos. Sem constituir-se uma exceção ao que ocorria na informatização da maioria das empresas, no BNDES, até aquele momento, estavam automatizados apenas os sistemas que compunham o chamado *back office*, tais como os sistemas de apoio aos procedimentos administrativos, contábeis e financeiros. Não havia sistemas para suportar as áreas fim do negócio, aquelas cujo objetivo primário era operacionalizar o

fomento do desenvolvimento econômico e social, como claramente revelam as entrevistas:

> Os usuários [da informática do BNDES] estavam restritos ao DEFIN [Departamento Financeiro], DEPCO [Departamento de Contabilidade] e AA [Área de Administração], os sistemas eram "sistemas meio" , não existiam usuários do [sistema] OPE [um dos sistemas que hoje atende às áreas fim de negócio]. [...] Um grande desafio já era suportar as áreas de negócio do banco. (Oswaldo Fonseca, em entrevista concedida ao autor em 21.06.2006).

> Não tínhamos muito contato com os usuários [...]. [Eu] não percebia, na época, o papel da informática no negócio do banco. Isso não era claro [relata João Calvano, analista de sistemas que ingressou no BNDES no início dos anos 1990]. (Entrevista concedida ao autor em 21.06.2006).

> As áreas [de negócio] reclamam que a AF [Área Financeira] ocupa mais de 90% [dos recursos] da informática, mas espere aí, qual é o nosso negócio? Primeiramente acho que somos um banco, não? [Pergunta Ricardo Matsushima, contador que durante muitos anos chefiou o Departamento de Contabilidade]. (Entrevista concedida ao autor em 14.07.2006).

Decorrente dessa nova percepção, passava a existir uma maior pressão por sistemas e serviços de informática. Todas as funções de informática do BNDES eram desempenhadas no DESIS — Departamento de Sistemas — cuja organização era praticamente a mesma definida com a reestruturação do BNDES em meados dos anos 1970, uma instalação típica de CPD baseado em *mainframe* com processamento *batch*. Quando a demanda por sistemas e serviços começou a crescer, várias características originalmente respeitadas e valorizadas, como padronização de procedimentos, existência de pontos de controle, segregação de funções, começaram a contribuir negativamente para a imagem do DESIS. As mesmas características que eram vistas como pontos fortes na década anterior, começaram a representar uma fraqueza. Ganhava corpo uma imagem pejorativa de burocracia, lentidão e isolamento, acompanhada da sensação acelerada de que a informática do BNDES não estava atendendo satisfatoriamente aos usuários.

> A informática era muito distante dos usuários, as áreas tinham seus problemas e a informática não atuava neles, além da dificuldade do ambiente de grande porte. Começou a haver a necessidade de cada área prover uma interface com a informática, para acessar dados, prover soluções locais. O DESIS tinha muito pouco contato fora da área, pouca gente utilizava os sistemas diretamente [...], a crítica que existia era que o DESIS era muito lento [...] (Renato Gouvêa, em entrevista concedida ao autor em 26.06.2006).

> No início, a parte de informática era muito centralizada, tínhamos muito pouco contato com o pessoal de sistemas. [...] Na área de planejamento, eu precisava de acesso às bases de dados. Comecei a freqüentar o DESIS — aquele computador grande, cartões perfurados — com dois colegas estatísticos e começamos a fazer uns programinhas [para extração de dados]. Levávamos uma eternidade para aprontar um levantamento de dados. [...] O DESIS era muito distante, as pessoas, de modo geral, não iam lá, eu era uma das poucas que ia [...]. Havia muita pressão das áreas operacionais porque o DESIS não tinha nada amigável. (Ângela Oliveira, em entrevista concedida ao autor em 07.07.2006).

> [Alan Fischler, chefe do DESIS nos anos 1990, revela que] o DESIS era visto como aquela coisa jurássica ligada ao mainframe, sem flexibilidade. O que é verdade, o *mainframe* não dá flexibilidade, não existiam ferramentas que dessem flexibilidade ao *mainframe*. (Entrevista concedida ao autor em 11.07.2006).
>
> [O economista Marcelo Nardin, que ingressou no BNDES no início dos anos 1980, esclarece que, até então,] os sistemas eram alguma coisa que rodava lá no primeiro andar, no CPD, e estávamos falando basicamente em termos de caráter corporativo e *back office*. A informática servia para processar grandes processamentos corporativos, como: folha de salários; manter um inventário das operações, uma base de dados de caráter muito estático e pouco representativo; e, de forma muito tênue, promover alguns processos relacionados à cobrança, mas ainda muito pouco estruturados e automatizados. Era uma espécie de máquina de calcular de luxo. (Entrevista concedida ao autor em 10.07.2006).
>
> A turma [analistas de sistemas e programadores] queria sair [do DESIS]. Era resultado da frustração de se estar sempre criticado e de fato não conseguir atender às demandas, estar sempre cobrado (mesmo "carregando pianos") e frustrado. A informática [...] deixava muito a desejar (Melvyn Cohen, em entrevista concedida ao autor em 20.06.2006)

Na mesma época em que a imagem da informática piorava por não conseguir atender tempestivamente à demanda crescente, surgia um novo ator, insubordinado à padronização e ao controle dos anos 1970. Na verdade, sob pressão, aquela rede sociotécnica da informática dos anos 1970 vinha se desestabilizando. O novo ator enredado foi o *microcomputador*, como revela Marcelo Nardin:

> Por volta de 1983/1984, temos o advento da microinformática aqui no banco, foi uma revolução no dia-a-dia da organização, e uma revolução do uso da TI [Tecnologia de Informação] aqui no BNDES (Entrevista concedida ao autor em 10.07.2006).

Novos atores e associações passaram a existir, de modo que a ordem resultante da rede tornou-se bastante diferente daquela observada anteriormente. Dentre eles, além do microcomputador, estavam os novos aplicativos — processadores de textos, planilhas eletrônicas —, as novas linguagens e ambientes de programação — *Open Access*, Fox Pro, Paradox, MS Access, Delphi — e até muitos novos programadores, que surgiam em decorrência da facilidade de uso, flexibilidade, proximidade e disponibilidade da microinformática. Se na rede anterior todos atuavam sob forte mediação — diversas especializações de tarefas e de profissionais, protocolos e pontos de controle —, agora a mediação propiciada pelo microcomputador favorecia o estabelecimento de outra ordem menos burocrática, o que entrava em ressonância com o anseio da maioria afastada da informática. Diferente do *mainframe*, cuja instalação já impunha diversos elementos de padronização, controle, metodologias e mediações, o microcomputador, aparentemente dotado de uma aura libertária com suas respostas *on line*, aplicava um duro golpe na hierarquia e na centralização,

pressionando pela descentralização e conseqüente aumento da complexidade daquela instalação de informática.

> Na década de 1980 vivemos uma mudança radical no hardware. Começam a surgir os microcomputadores. A partir deste ponto é que criou-se a confusão. Tinha-se uma metodologia para se desenvolver sistemas de uma determinada forma e começam a aparecer equipamentos que obrigam você a desenvolver de outra forma. Nessa época, em que se começou a disseminar a microinformática, o banco ficou meio perdido. Por que? Em qualquer área operacional aqui do banco você encontra mestres, doutores, professores, pessoal muito especializado [capacitado]. [Soma-se a isso que] você começa a ter ferramentas [de programação] um pouco mais fortes e de fácil aprendizado e começamos a ter um monte de "analistas de sistemas" nas áreas operacionais. Foi um tal de usuário desenvolver coisas aqui dentro do banco entre as décadas de 80 e 90, principalmente quando compramos o *Open Access* — que tinha até um programador [linguagem/ferramenta de programação]. Pronto!... aquilo para o pessoal de engenharia, economia, que tinha utilizado FORTRAN na faculdade, quando pegou uma ferramenta dessa... foi um tal de desenvolver projeto [de novos sistemas] aqui em paralelo [com os sistemas corporativos] que foi uma grandeza! (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).
>
> [A microinformática contribuiu para a perda da cultura de organização, padronização e produtividade que existia] porque [antes] se tinha um ambiente só, aí você seguia aquilo [métodos e padrões] naquele ambiente. Depois vieram os micros espalhados pelo banco, [...] tinha época que as pessoas traziam software de casa e instalavam (Vânia Carvalho, em entrevista concedida ao autor em 10.07.2006).
>
> [A própria cultura de micro aqui já começa sem a preocupação com padronização,] fizemos uma transição descontrolada. Quando entrei [no BNDES, no início dos anos 1990] o DESIS já não trabalhava mais com isso [padrões, processos formalizados]. [...] Cheguei aqui e a rede era um monte de computador ligado um no outro. Cada um fazia o que queria... sem padrão nenhum, nomes de diretórios, *back'ups* nada. (José Gonçalves, em entrevista concedida ao autor em 02.08.2006).

A desestabilização da rede da década anterior causava a fuga de vários atores, o que contribuía mais ainda para sua própria desestabilização. Por exemplo, no final dos anos 1980 o MIT — o melhor representante da rede dos anos 1970 — e a segregação das funções de analistas de sistemas e programadores não atuavam mais na informática do BNDES.

> Num momento começa a surgir a crença de que os projetos andariam mais rápido se os programadores estivessem do lado, aí acabaram com o *pool* de programadores. Isso começa com a experiência e conhecimento dos programadores, o pensamento era: se o "garoto" [o programador] é bom quero que ele fique trabalhando comigo, pois minha produtividade aumentará bastante. A divisão analista/programador parece que existiu até mais ou menos 1987, foi uma desespecialização [decorrente da extinção do *pool* de programadores]. Aí esses camaradas [programadores] começam a trabalhar normalmente como analistas de sistemas [também]. Uma auditoria de 1991 aponta um grande desvio de função [o que era problemático para o BNDES que ficava exposto judicialmente, pois o profissional havia feito o concurso para uma função e estava desempenhando outra]. A Ivone Saraiva, superintendente da AA [Área de Administração] da época, regularizou a situação. (Oswaldo Fonseca, em entrevista concedida ao autor em 21.06.2006).
>
> Quando a informática [no BNDES] começou a crescer, o MIT [Manual de Instruções Técnicas] acabou. Mais ou menos no final dos anos 1980, começou a ser introduzida a microinformática. As pessoas começaram a desenvolver [sistemas, inclusive fora do DESIS], a usarem outras técnicas,

> outras ferramentas, então começou a haver um certo descasamento entre todos esses trabalhos. Por que o MIT acabou? [Primeiro] porque a pessoa que fazia a manutenção dele se aposentou, o Willian "capita" ("capita" porque foi capitão da marinha). Então o MIT passou a ser pouco atualizado. [Segundo,] como a informática é muito dinâmica, foram entrando novas práticas, novas ferramentas, sem estarem refletidas nesse manual, nesse método de trabalho. Então não só a aposentadoria do Willian ocasionou isso, mas as pessoas que depois estavam responsáveis não conseguiram se dedicar à atividade de manutenção do MIT [na nova dinâmica da informática] (Sidney Franco, em entrevista concedida ao autor em 01.08.2006).

A partir da movimentação de velhos e novos atores, outra rede foi sendo construída. Novas áreas e novos programadores, graças à atuação do microcomputador e seu instrumental, eram enredados a revelia do DESIS, muitas vezes ele próprio visto como o problema. Algumas iniciativas passaram a mostrar a possibilidade concreta dos microcomputadores, isolados num primeiro momento e depois interligados em redes, atenderem ostensivamente todo o BNDES, como revelam os depoimentos a seguir:

> Com a microinformática, foi uma evolução danada, pois passamos a trazer os dados do computador central para os micros e trabalhávamos no micro. Trabalhava com estatísticas sobre os agentes financeiros, nas horas "vagas" ia olhando como o trabalho era feito, pensava no que achava que iria precisar, fui montando campos necessários, fui "contaminando" os colegas da sala com a idéia de criar um sisteminha. Aí os colegas começaram a me ajudar, até que criamos o DocRec [Documentos Recebidos] e um outro para já sair no micro alguma coisa de liberações [de recursos de financiamento]. Aí comecei a trabalhar até muito tarde. A salinha do micro, era uma sala só, um cubículo, era perto do elevador privativo dos superintendentes. Um dia o superintendente resolveu ir lá ver quem era que estava naquela sala e fazendo o quê, pois sempre via a luz acessa. Aí viu uma série de informações que eu já tinha disponível, [...] ele adorou, tinha informações que ele precisava, [...] e resolveu que eu deveria ficar mais "independente", na verdade, trabalhar só nisso. [...] Eu fui para um outro departamento, com colaboradores, para ficar só com isso. Aí o superintendente da AP-III [Área de Projetos III, operações indiretas] levou [à diretoria] a idéia da criação de uma gerência de informática, de sistemas. Eu fui contra, não queria, queria que fosse uma gerência de informação. Não queria entrar em choque com o DESIS. Eu sabia que seria a primeira fora do departamento de sistemas com uma coisa equivalente [— desenvolvimento de sistemas —] e achava que isso iria pulverizar [o desenvolvimento, os sistemas] no banco. Mas o superintendente foi irredutível, achava que era uma evolução, achava que era uma coisa muito mais forte: sistemas, não só informações. Ele queria, na verdade, uma reforma administrativa, a descentralização da informática (Ângela Oliveira, em entrevista concedida ao autor em 07.07.2006).

> A microinformática do BNDES começou corporativamente por uma gerência na BNDESPAR, à época uma empresa independente do BNDES, [... iniciando] uma descentralização do processamento de dados relacionado às operações com o mercado acionário. Isso mostra um caminho com grandes vantagens para o usuário. Primeiro porque o usuário de negócio do BNDES não era atendido, praticamente. O usuário atendido era o do *back office* do BNDES. Quem estava no negócio mesmo não tinha tecnologia nenhuma, usava uma máquina de calcular e uma máquina de escrever. Com o advento da microinformática a BNDESPAR descola e cria sua gerência de sistemas, a primeira descentralizada, o que mostra um caminho. Todas as áreas operacionais passam a reivindicar ter seus núcleos de processamento de dados descentralizados. [...] A microinformática vem alterar o monopólio do CPD, criando um parceiro muito mais próximo do cliente, o gerente de sistemas descentralizado, com muito mais foco no atendimento das necessidades do cliente, usando plataforma baixa [microinformática] mas com grande capacidade, capacidade crescente, de processamento. Entre

> 1983 e 1986, ou 1987, mais ou menos, havia um micro por andar no BNDES, cujo crescimento da utilização cresce exponencialmente... (Marcelo Nardin, em entrevista concedida ao autor em 10.07.2006).
>
> Pegaram o exemplo da BNDESPAR, que foi um sucesso na época [...]. Mas não tem milagre, a BNDESPAR era uma coisa pequena, havia poucas pessoas, com possibilidade de compra separada [o BNDES tem que licitar], tudo sem burocracia, tudo, dentro de um certo limite, independente (Alan Fischler, em entrevista concedida ao autor em 11.07.2006).

Diante deste quadro do final da década de 1980, a administração do BNDES entendeu ser necessária uma reforma na estrutura da informática e deliberou a criação de núcleos locais de informática em cada área de negócio. Esses núcleos locais foram batizados de GESIS — Gerências de Sistemas — e ficaram subordinados hierarquicamente às superintendências das áreas de negócio, não ao departamento de informática corporativa, o DESIS. Em tese, as GESIS deviam subordinação "técnica" ao DESIS, porém, na prática, isso não ocorreu. Não foram definidas, claramente, as atribuições do DESIS e das GESIS quando elas foram criadas. As GESIS foram criadas oficialmente em 1989.

> A coisa não estava bem definida, e pior, as pessoas que tinham colaborado para pensar isso foram retiradas e chamaram o Milton Dias para ser chefe desse departamento. Só que o Milton não tinha vivência e experiência no negócio [de informática], mas tem uma capacidade de gestão que foi importantíssima para lá. Aí ele deu uma parada antes das pessoas serem nomeadas para as GESIS e gerou uma conversa para ver como isso [a descentralização da informática] seria feito. Na época, chamaram um sujeito de fora, era da PUC mas já havia trabalhado na DATAMEC, para ajudar numa metodologia, para repensar o DESIS, a informática. Ele, na realidade, gerenciou um *braimstorm* com as pessoas da informática para repensarem a informática. Basicamente definiram que funções a informática teria que desenvolver e, definidas essas funções, como teríamos que nos reestruturar para atendê-las. Sendo que um dado do problema era certo: vão existir DESIS e GESIS na solução, tivemos que engolir isso (Melvyn Cohen, em entrevista concedida ao autor em 20.06.2006).

Entre julho e outubro de 1989, o DESIS empenhou-se em esforços que pudessem ordenar a descentralização que passaria a vigorar na informática do BNDES. Alguns profissionais do DESIS foram convidados para assumirem a gerência de algumas GESIS, mas a maioria dos profissionais que as compuseram eram aqueles que, de alguma forma, já vinham atuando localmente como mediadores de suas áreas com o DESIS ainda centralizado. Na maioria dos casos, esses profissionais não tinham formação de base em informática. Por isso o DESIS, com apoio do Departamento de Desenvolvimento de Pessoal, promoveu o treinamento de seus profissionais e dos profissionais das GESIS em análise de sistemas, através do CIATAS — Curso Intensivo de Atualização em Análise de Sistemas —, em 1989. Outra medida foi desenvolver a MEDES — Metodologia de Desenvolvimento de Sistemas do BNDES —, baseada nos princípios da *análise estruturada* de sistemas,

uma das abordagens mais difundidas da época, na tentativa de ordenar a operação na nova estrutura.

A seguir, um trecho do relatório de atividades do DESIS no ano de 1989 destaca a idéia da *difusão* das "melhores práticas" da engenharia de software, materializada em cursos e na metodologia de análise de sistemas, como meio de se restabelecer a organização, padronização e produtividade da informática.

> [...]
> 2 – A reestruturação do DESIS
> [...] O processo de reestruturação do DESIS não podia prescindir de alguns parâmetros básicos, tais como: a definição exata dos papéis a serem desempenhados pelas GESIS e pelo Comitê de Sistemas e o estabelecimento de uma Política de Informatização contendo princípios e diretrizes que passassem a nortear o uso da informática no BNDES. Nesse sentido, desenvolveu-se um trabalho, *contando com a participação de representantes das diversas Áreas do Banco* [...]. A partir daí, foi possível dar-se início efetivo ao processo de reestruturação do DESIS, *com base numa orientação explícita da Alta Administração. Esse processo envolveu de forma participativa todos os técnicos e pessoal administrativo do departamento* e encontra-se, hoje, concluído. [...].
>
> 3 – Providências Complementares
> Em paralelo às medidas descritas acima, foram tomadas três providências importantes ao longo de 1989, a saber: a realização, em articulação com o DEDES [Departamento de Desenvolvimento de Pessoal], do Curso de Análise de Sistemas, a implantação da MEDES/BNDES — Metodologia para Desenvolvimento de Sistemas do BNDES — e a criação da Gerência de Planejamento no DESIS. O curso teve como propósito fornecer o instrumental necessário para que técnicos de outras formações pudessem atuar também na área de sistemas, tendo em vista a escassez interna de profissionais especializados e a impossibilidade de realização de concurso público. *A implantação da metodologia [MEDES] visou assegurar, desde o primeiro momento, que a descentralização da informática não degenerasse em sua própria desorganização*. Esse intento apóia-se no fato de que a adoção de uma metodologia que homogeneíze o desenvolvimento e a documentação dos sistemas em uso no Banco, evita que se estabeleça um vínculo pessoal entre o analista e o sistema por ele desenvolvido, garantindo-se, assim, o cunho institucional que deve ser sempre buscado, particularmente, no que tange aos sistemas. [...] (DESIS, 1989, grifos nossos).

No entanto, mesmo com o apoio da alta administração e o envolvimento de representantes das áreas de negócio, na prática o que se viu com a criação das GESIS foi uma ruptura com o DESIS, que, por sua vez, não conseguiu exercer nenhuma coordenação técnica, e muito menos qualquer controle sobre as GESIS. Nem mesmo vingou a idéia de utilização de uma metodologia para favorecer a homogeneização do desenvolvimento e documentação de sistemas. Com a instituição formal das GESIS, cresceu exponencialmente o desenvolvimento de sistemas locais, que já vinha ocorrendo desde o início da utilização dos microcomputadores no BNDES. Os problemas decorrentes da abrupta distribuição da informática não demoraram a aparecer:

> As GESIS começam [surgem] para resolver um problema e viram, [elas mesmas,] em si, um problema. [...] Costumo chamar esse processo que

> vivemos de descentralização selvagem. (Marcelo Nardin, em entrevista concedida ao autor em 10.07.2006).
>
> [Essa descentralização abrupta derivou-se de uma apressada] associação que foi feita: deu certo [na BNDESPAR] porque era independente [do DESIS], vamos replicar esse modelo pelo banco. E aí criaram-se as GESIS, uma em cada área, totalmente independente do DESIS [...], a descentralização absoluta, sem coordenação centralizada, em que cada área fazia seus sistemas independentes. Aquela coisa [os sistemas, os fluxos de informações, as próprias informações] não se encontrava [integrava] em lugar nenhum. (Alan Fischler, em entrevista concedida ao autor em 11.07.2006).

Pode-se notar que não houve *traduções* suficientes para garantir a atuação dos novos atores na informática do BNDES de modo a manter a estabilização da rede urdida nos anos 1970. Os objetivos e interesses dos novos atores não se deslocaram no sentido de manter a padronização e o controle que antes existia. Ao contrário, os que passaram a atuar na rede — como os microcomputadores, as GESIS, os novos programadores — e alguns outros que a deixaram — como a "terra virgem" e o MIT — precipitaram uma outra ordem resultante, bastante diferente da observada na década de 1970.

A própria estrutura organizacional foi alterada, mudando, às vezes radicalmente, a atuação dos mais diversos elementos da rede, como analistas, usuários, sistemas, metodologias, padrões. Não bastasse o DESIS ter perdido o controle central com a distribuição da informática em diversas GESIS, não atuaram conforme esperado os atores aos quais se atribuiu a possibilidade de coordenação e controle por parte do DESIS. É o caso da MEDES, do treinamento em análise estruturada, da *política de informatização*[84] e do *comitê de sistemas*[85]. Resumidamente, a política de informatização estabelecia as regras a serem seguidas por DESIS e GESIS, dentre outras coisas, atribuindo a cada um suas responsabilidades. O comitê de sistemas, além de priorizar os projetos, deveria ser o árbitro, sempre que necessário, para garantir o cumprimento da política de informatização. Esses atores (MEDES, treinamento, política, comitê) não foram aliados fortes, ou, vale dizer, não efetivaram *traduções* suficientes para refazer o efeito de ordem desejado na rede sociotécnica da informática do BNDES.

Uma rede sociotécnica tão diferente só poderia produzir, no final dos anos 1980, um efeito também muito diferente do observado na década de 1970. Não se podia querer que metodologias, padrões, computadores (micros e *mainframe*), analistas e programadores (antigos e os novos que surgiram com a microinformática), funções

---

[84] Definida na Resolução de Diretoria 701/89.

[85] Instituído pela Resolução de Diretoria 690/89 e com atribuições e composição definidas pela Resolução de Diretoria 702/89.

especializadas e outras "desespecializadas", pontos de controle, estrutura organizacional, controle hierárquico dos projetos, sistemas legados que passaram a existir, processos, atividades de negócio, entre tantos outros, com atuações tão distintas do que se observou nos anos 1970, produzissem o mesmo efeito.

> Criaram-se um monte de Gerências de Sistemas (as GESIS) em tudo que era lugar sem uma metodologia padrão de desenvolvimento de sistemas e com características diferentes [entre si]. Duas GESIS, AF [Área Financeira] e AA [Área de Administração], herdaram os analistas de desenvolvimento que continuaram a utilizar o grande porte, como se fossem o velho DESIS. As outras GESIS partiram para a microinformática sem o menor método, cada uma vai para um canto, cada uma faz uma coisa diferente, não houve uma coordenação em termos de desenvolvimento de sistemas, os gerentes de sistemas deixam de ser comandados por um único departamento, passam a ser comandados pelos superintendentes das áreas. (Osmar Frota, em entrevista concedida ao autor em 31.07.2006).

> Com o advento da microinformática tivemos o boom das gerencias de sistemas — GESIS. No Finsocial não tinha uma, mas tinha eu lá fazendo um Banco de Dados e outros... Rapidamente *minha secretária*, eu era gerente, *foi especializada* para ser alimentadora do Banco de Dados do AO (*Open Access*), *naturalmente passou a ser uma programadora*, passou a dar suporte para o usuário interno e virou uma espécie de GESIS interna do Finsocial. Esse Banco de Dados, feito para um determinado departamento do Finsocial, para educação e saúde, foi "comprado" pela área rural. [... No entanto,] em função da descentralização selvagem, na década de 1990, vemos uma guerra aberta entre DESIS — que estava perdendo a guerra — e as gerências de sistemas (GESIS) que praticamente viram CPD's em si, com o direito a decidir sobre coisas que não deveriam ter direito. Mas, no vazio, passaram a decidir acerca de arquitetura de sistemas de informação, de tecnologia, de processos de trabalho. A organização, então, tende a grande disparidade nas áreas de negócio. Passam a se constituir como empresas autônomas, num processo extremamente complicado. [...] A guerra que ocorreu nos anos 1990 [DESIS x GESIS's] às vezes invade o campo interpessoal, se torna uma conversa de surdos, se passa a ter uma disputa aberta e no final das contas não se sabe mais nem o que se está disputando. Isso praticamente impede a discussão. (Marcelo Nardin, em entrevista concedida ao autor em 10.07.2006).

Nessa altura, portanto, no DESIS pós-descentralização, não mais havia aquela organização, padronização e produtividade surgidas nos anos 1970. Um ano após a criação oficial das GESIS, lembrando que alguns núcleos isolados já vinham desenvolvendo aplicativos nos microcomputadores desde 1985, a situação da informática recém descentralizada parecia alarmante. Diante desse quadro, o DESIS contratou, em novembro de 1990, o serviço de consultoria da *Andersen Consulting*[86] na tentativa de tornar eficiente, operacional e padronizada a estrutura distribuída de informática no BNDES.

> A consultoria da Andersen era uma tentativa de colocar uma ordem nesse caos. A percepção dos usuários, de área operacional, era: puxa estou supersatisfeito com a minha GESIS, resolve meus problemas, me dá suporte naquilo que eu uso: planilha, processador de texto, ... não usavam nada, nada, absolutamente nada de grande porte [ou seja, nada do DESIS]. Logo, os usuários eram muito favoráveis à essa estrutura. Só que algumas GESIS começaram a desenvolver sistemas realmente independentes. Aí o DESIS

[86] IP AA/DESIS 003/90 — Instrução Padronizada com a proposição, aprovada pela diretoria, de contratação da *Andersen Consulting*.

> começa a perder o controle do que estava sendo feito nos próprios sistemas corporativos: cobrança, contabilidade [por exemplo]. O BNDES estava quase perdendo a identidade, como empresa, na parte de informática. Aí fez-se o tal plano da Andersen, [...] o objetivo era preservar a descentralização, porém com limites e controle (Alan Fischler, em entrevista concedida ao autor em 11.07.2006).

A consultoria realizou um diagnóstico da situação e referendou a estrutura de distribuição adotada:

> A estrutura adotada pela informática do Sistema BNDES segue uma tendência bastante moderna, principalmente no segmento bancário (onde tem sido implantada com bastante sucesso), que é a descentralização das funções de desenvolvimento/manutenção e apoio ao usuário para as áreas usuárias finais de informática. (ANDERSEN CONSULTING, 1991, tópico III, item 2).

Porém, no caso do BNDES, dada a forma traumática com que a descentralização foi implementada, o sucesso esperado não estava se materializando, o que dificultava o desenvolvimento e a estabilização da função informática (daria no mesmo dizer: a estabilização de sua rede sociotécnica). Ainda segundo a consultoria:

> O DESIS não exerce nenhum tipo de supervisão técnica sobre as GESIS. O processo de planejamento (operacional) de informática não é consolidado por todo o sistema BNDES. A causa provável disso é que o processo de descentralização foi conduzido de forma rápida e traumática, sem a definição clara das atribuições das GESIS e do DESIS, não mantendo neste ou em qualquer outra entidade a coordenação geral da função informática. Como conseqüências: desmotivação (traumas) nos profissionais de informática; duplicação de recursos e informações ao longo de toda Organização; dificuldade na caracterização do que é corporativo e específico (ibid., item 2.1).

> A disseminação da microinformática no Sistema BNDES ocorreu de forma muito agressiva, baseada no baixo custo das máquinas [...]. Insatisfeitos com o tempo de resposta do *mainframe* [na verdade, com o tempo de resposta do DESIS], muitos usuários buscaram nos micros a solução para seu problema. [...] Este objetivo de suprir as necessidades dos usuários não foi acompanhado por estruturas eficientes de apoio. Atualmente, muitos são os controles que passaram, desordenadamente, do *mainframe* para os micros. As conseqüências deste fato são ineficiência administrativa e aumento do custo operacional (ibid., tópico III, item 8.4).

A consultoria estruturou, hierarquicamente, três planos de projetos cuja execução, em princípio, novamente tornaria padronizada e mais efetiva a informática do BNDES. O primeiro plano era o *plano de tecnologia*, que abordava projetos de hardware, software, segurança e infra-estrutura; o segundo, o *plano organizacional*, abordava a readequação da estrutura organizacional, formalização das atribuições do DESIS e GESIS, a necessidade de estruturar os trabalhos executados no DESIS e nas GESIS por projetos e, também, a necessidade de implantação de uma metodologia de desenvolvimento e manutenção de sistemas. O terceiro plano era o *plano de migração*, que abordava a transição da situação atual para a desejada, ordenando os diversos projetos necessários, estabelecendo cronogramas e recursos.

Na ocasião, já nos anos 1990, a engenharia de software era uma disciplina bem estabelecida, fortemente respaldada no instrumental da análise de sistemas. Transparece nos relatórios e planos da consultoria a abordagem difusionista presente na engenharia de software, onde imputava-se à implantação de um modelo "universal", no caso uma metodologia para o desenvolvimento de sistemas, a obtenção do reordenamento desejado. Não bastasse ser temática recorrente nos planos e relatórios da consultoria, a implantação da metodologia encabeçava a lista dos projetos de infra-estrutura, denotando o caráter prioritário que lhe era devotado (ANDERSEN CONSULTING, 1991, tópico VII, item 5).

> Sente-se falta de metodologia, documentação e padronização no uso e desenvolvimento de sistemas. A documentação técnica / funcional é, em geral, fraca, desatualizada e não atende à nenhuma padronização. A comunicação entre o desenvolvimento, a produção e o suporte é dificultada. O envolvimento dos usuários não é o ideal. A produção e o suporte não conseguem operar com total produtividade. A manutenção dos sistemas é encarecida e existe demora na solução de problemas. [Em decorrência, como necessária] ação: *deve-se selecionar uma metodologia e implantá-la antes da implantação do presente plano* (ibid., tópico III, item 9, grifo nosso).

A necessidade de utilização de uma metodologia parecia inquestionável, ainda mais com as promessas que a acompanhavam. Por exemplo, a primeira versão da MEDES, desenvolvida pelo DESIS em 1989 como parte das providências complementares à descentralização da informática, propunha:

> O principal *compromisso da MEDES é resolver os problemas da organização*, através do desenvolvimento de sistemas de informação de qualidade. A razão de ser de uma metodologia é garantir a qualidade do sistema que vai ser produzido. Se fosse trivial obter tal qualidade pouca ou nenhuma importância seria dada a este tema. Entretanto, ao contrário do que se desejaria, *é muito freqüente a constatação de que o sistema, quando pronto, não atende aos requisitos mínimos desejados. Daí a necessidade de aprimoramento da capacidade metodológica, tanto aos níveis da organização como de seus técnicos e usuários* (MEDES, 1990, p.2, grifos nossos).

Reforçando a idéia do DESIS, expressa dois anos antes no relatório de atividades de 1989, a consultoria também valorizou a implantação da MEDES, da metodologia de desenvolvimento de sistemas, como elemento chave para a solução dos problemas à época enfrentados pela informática do BNDES. No cronograma do *plano de migração* torna-se explícita a abordagem *difusionista* do projeto de "*aquisição de metodologia para o desenvolvimento de sistemas*". Esse projeto considerava suficiente um analista e apenas quatro meses para implantar a metodologia, tratando-o, ainda, como uma mera aquisição de um artefato (ANDERSEN CONSULTING, 1991, tópico IX, item 4). Deste modo, o projeto denotava total desconsideração para o desafio existente na prática desse tipo de empreitada, um *esforço de tradução*, um esforço de (re)projetação local do modelo, como já afirmado. A abordagem da consultoria, tão simplificadamente quanto as narrativas usuais sobre as causas de

sucesso/fracasso dos projetos de software, devotava grande poder ao modelo "universal", considerado capaz, *per si*, de solucionar muitos dos problemas da informática. Cabe registrar que esse projeto produziu uma nova versão da MEDES em 1993 (MEDES, 1993).

Sequer foi percebido, ou, se foi, não foi relatado nos planos e relatórios da consultoria, o clima de "guerra" entre DESIS e GESIS. Que *traduções* seriam necessárias para aproximar os objetivos desses dois atores (DESIS e GESIS) tão antagônicos? Dificilmente uma metodologia com o claro objetivo de controlar o desenvolvimento de software praticado nas GESIS seria um elo forte na rede, dispondo da capacidade de mantê-la para que a "*descentralização da informática não degenerasse em sua própria desorganização*" (DESIS, 1989). Os desenvolvedores nas GESIS, que aprenderam na prática a resolverem seus problemas fazendo pequenos aplicativos nos microcomputadores, dificilmente iriam se submeter à tentativa do DESIS de manter, segundo a idéia que faziam, seu controle, monopólio e influência burocrática. A seu jeito, a secretária, ou qualquer outro profissional, programava e materializava soluções para seus problemas locais, a revelia do DESIS que, também segundo a idéia que faziam, não os atendia. Não fosse o bastante, existia (e ainda existe) no BNDES a cultura de que a não padronização nas áreas de negócio é um valor. Uma cultura, portanto, avessa à idéia de utilização de uma metodologia padronizadora.

> O BNDES é uma empresa, seu negócio é muito *taylor made*. Isso faz com que o usuário se acostume a ter um tratamento artesanal, é assim que ele trabalha e quer ser compreendido dessa maneira. É inegável que o negócio do BNDES exige uma flexibilidade na ponta. [...] Desenvolvimento [fomento do desenvolvimento econômico e social] não é alguma coisa que está pronta num livro na estante, vou pegar meu *book* [de processo] e ver como a empresa me manda fazer isto. Se for assim a gente não faz nada no BNDES. [...] No BNDES o negócio não está lá fora, está aqui dentro. Desenvolvimento é o que nós identificarmos junto com quem de direito, governo e sociedade, como desenvolvimento. Uma vez identificado, temos que inventar um jeito de fazer. Nesse sentido, isso aqui [o BNDES] é uma grande ferramentaria, toda hora criamos uma ferramenta nova. (Marcelo Nardin, em entrevista concedida ao autor em 10.07.2006).

Mesmo desconsiderando o antagonismo DESIS x GESIS, uma investigação no próprio DESIS daquela época revelaria a inviabilidade de implantação de uma metodologia, nos moldes estabelecidos, para retornar ao controle e padronização existentes nos anos 1970. O BNDES havia ficado uma década sem realizar concursos e, portanto, sem contratar novos profissionais. Quando trinta e cinco novos analistas de sistemas foram contratados, no início dos anos 1990, encontraram um ambiente "traumatizado" pela descentralização. Por um lado, esses novos analistas tinham um perfil muito apropriado, devido à sua formação acadêmica, ao uso da metodologia que se tentava implantar. Por outro lado, sofriam uma "pressão" por parte dos analistas

antigos, na ocasião ocupando postos de gerência e coordenação, que resistiam à metodologia. Isso influenciava diretamente as relações que se estabeleciam entre analistas novos e antigos.

Os analistas que vivenciaram a "gloriosa" época da informática do BNDES nos anos 1970, sob o "trauma" da descentralização não viam na MEDES a *inscrição* de *programas de ação* (seção 4.2) capazes de reproduzir o sucesso que haviam materializado. Deste modo, a relação que passou a existir com os analistas recém chegados tornava-se algumas vezes conflituosa desde o início, por conta da diferença de formação entre analistas novos e antigos. Os novos achavam natural tentar utilizar a MEDES nos problemas que enfrentavam, tanto porque havia a orientação manifesta no BNDES para seguir essa metodologia, quanto porque tinham sido treinados academicamente para isso. Porém, enfrentavam às vezes conflitos culturais com os analistas antigos, em muitos casos seus próprios chefes. Para os analistas já antigos no BNDES, a concepção e projetação dos sistemas era melhor *inscrita* segundo uma abordagem funcional, cujos instrumentos clássicos eram os diagramas hierárquico funcionais (DHF) e os fluxogramas que utilizavam. Quando os novos analistas utilizavam diversos outros diagramas da análise estruturada, como os DFD's, DER's e DTE's, por exemplo (vide nota 77), não parecia, aos analistas antigos, que tentavam ser produtivos em suas atividades. O analista de sistemas João Calvano, naquele momento um dos recém chegados, aponta esta questão:

> A geração que entrou conhecia muito isso [a análise estruturada], a maioria fazendo mestrado [nessa área]. Cláudio Cézar [Almeida, um importante ator na segunda história, que virá a seguir] tinha a visão de que era importante, além de [ele próprio] estar ligado ao meio acadêmico. Existiam algumas pessoas que viam nisso a necessidade da gente mudar, e que o momento [com a entrada de gente nova] seria propício. Mas nessa época, estava-se num momento de transformação, falava-se muito em *downsizing*. Isso metia medo na casa, o pessoal pensava: vou perder tudo, vou ter que fazer tudo de novo! A IBM oferece uma estação RISC-6000 para o banco testar, havia a idéia de mudar a plataforma, não havia software livre, Windows rastejando, a possibilidade era UNIX. [...] Chegam 35 funcionários novos, quase todos com menos de 3 anos de experiência de mercado, num concurso em que metade dos aprovados tinham mestrado, que falavam uma linguagem diferente da linguagem da turma que já estava no banco. Nas discussões, quando se tentava ir para o quadro e fazer um DFD, ou um DER, tentando abstrair o que todo mundo falava como "cartão COBOL" [referência herdada da época dos cartões perfurados]... essa era a linguagem usada, cartão número tal... Quando se tentava abstrair para pegar a essência do negócio, colocar no nível de cima [mais abstrato], as pessoas resistiam: está chegando uma "molecada" nova que não quer programar não, só querem ficar definindo e escrevendo "coisas" [os diagramas da análise estruturada]. Esta turma que está chegando está querendo alguma coisa mesmo? (Entrevista concedida ao autor em 21.06.2006).

A título de epílogo desta primeira história, é necessário dizer que a MEDES não vingou, não se tornou uma prática efetiva de desenvolvimento de software no BNDES. Logo, não foi possível difundir as "melhores práticas" e preceitos vigentes na

engenharia de software da época através da "implantação" da análise estruturada. Outra forma de dizê-lo seria: a MEDES não conseguiu *inscrever programas de ação* suficientemente fortes para novamente ordenar, como se propunha, o desenvolvimento de software no BNDES. De fato, a "guerra" entre DESIS e GESIS perdurou até o início dos anos 2000, quando houve, de forma não menos "traumática", uma nova centralização da informática decorrente de uma profunda mudança na estrutura de todo o BNDES. Durante a década de 1990 restaram poucos vestígios daquela rede tão bem estruturada nos anos 1970.

5.2 Segunda História — Implantação do *CMMI* no início dos anos 2000

Segue agora uma narrativa sobre a implantação, ainda em curso, de um processo de software no BNDES. O autor não reclama isenção em relação aos fatos narrados, uma vez que esteve envolvido desde o início, no papel de integrante do SEPG — *Software Engineering Process Group*[87].

O atual projeto que visa à implantação de um processo de software (IPS) no BNDES tem sua origem em 2001, época em que a empresa sofreu uma profunda reestruturação organizacional com a criação de uma estrutura matricial cliente x produto, alterando também a organização da informática. Antes vigorava a estrutura descentralizada, com o DESIS — Departamento de Sistemas — subordinado à Área de Administração e 14 GESIS — Gerências de Sistemas — distribuídas por todas as áreas de negócio. Com a reestruturação, as gerências de sistemas distribuídas foram extintas e a informática do BNDES novamente voltou a ser centralizada, desta vez, alcançando o *status* de área na organização. Assim, o maior posto executivo da informática do BNDES deixou de ser o de *chefia de departamento* e tornou-se o de *superintendência de área*.

Foi criada a área de Tecnologia de Informação (TI) com três gerências executivas, abandonando-se o uso do termo departamento, a saber: GESIS — Gerência Executiva de Sistemas (no lugar do antigo DESIS), GEOPI — Gerência Executiva de Operações e Infra-estrutura e GEINP — Gerência Executiva de Integração de Processos. Em contrapartida à extinção das antigas GESIS (as gerências de sistemas distribuídas) nas áreas de negócio, foram criadas assessorias de TI, cujos assessores ficariam lotados nas próprias áreas de negócio para exercer o

[87] SEPG é o grupo encarregado de guiar a implantação e utilização de um processo de software (CARTER et al., 2002).

papel de interlocução com a área de TI. Os assessores de TI, como passaram a ser chamados, eram, na maior parte das vezes, os gerentes das gerências de sistemas distribuídas que acabavam de ser extintas.

Também foi criado um Comitê de TI (uma nova versão do já inexistente Comitê de Sistemas dos anos 1990) no qual participavam o superintendente da área de TI, os chefes dos departamentos de TI e os assessores de TI. Esse comitê reunia-se regularmente para tomar conhecimento das demandas das áreas de negócio e priorizar os projetos de informática. O analista de sistemas Cláudio Almeida, um dos assessores de TI, que ingressou no BNDES no início dos anos 1980, revela que nessas reuniões

> os assessores levantaram para o superintendente de TI, *novamente*, a necessidade de se ter uma estrutura de trabalho [mais formalizada para o desenvolvimento de software]. Apesar da responsabilidade do desenvolvimento não ser nossa [dos assessores de TI], éramos mais analistas de negócio do que desenvolvedores, cobramos isso porque sabíamos que a coisa não iria andar bem, se não tivesse uma estrutura [mais formalizada para o desenvolvimento de sistemas]. O superintendente, [Antônio] Faoro, acata a idéia e me escala para coordenar o grupo que iria tomar conta disso. Não sei se por vingança ou como ameaça, porque *eu era um dos que mais cobrava* [por processo de software]. (Entrevista concedida ao autor em 26.06.2006).

O trecho grifado acima é importante para esclarecer dois pontos. O primeiro é que o superintendente da época não tinha como objetivo seu estabelecer um processo de software. De fato, por suas características pessoais e profissionais, o superintendente, Antônio Faoro, não demonstrava muito interesse pelo assunto, como vemos nos relatos: "*O Faoro não tinha o mínimo interesse pela parte de processo*" (José Gonçalves, em entrevista concedida ao autor em 02.08.2006); "*Só não acredito muito na visão do Faoro coadunando com isso* [processo de software]" (Sérgio Viveiros, em entrevista concedida ao autor em 10.08.2006). O segundo ponto diz respeito a Cláudio Almeida ser um entusiasta do assunto. Ao dizer que existia a necessidade de estabelecimento de uma estrutura mais formalizada para o desenvolvimento de software — *novamente* — ele faz referência à iniciativa anterior, nos anos 1990, com a MEDES. Como ele mesmo relata, sua participação foi muito ativa na iniciativa da MEDES: "*montamos uma metodologia baseada na análise estruturada* [...] *me interessei muito pelo assunto e acabei dando aula sobre engenharia de software*" (Entrevista concedida ao autor em 26.06.2006).

O superintendente, então, estabeleceu um grupo de trabalho composto por Cláudio Almeida, como coordenador, por mais uma assessora de TI, Lílian Mendes, por gerentes de desenvolvimento e por um gerente de administração de dados. Contudo, a idéia de que a TI/BNDES precisava de um processo de software mais formalizado era controversa.

> [Com o Antônio Faoro] tivemos um momento propício, pois sempre existiu muita divisão de pensamento. Tinha muita gente que achava importante termos uma metodologia, um esquema bem estruturado de trabalho, e tinha gente que achava ser desnecessário, que atrapalha, que é só burocracia, que limita criatividade. Porém, era um momento em que muita gente nova estava entrando no banco [devido aos concursos ocorridos a partir de 1997], com uma cabeça diferente, com formação acadêmica boa na área [de engenharia de software], e que queria colocar isso em prática. Momento melhor que em 1990 [iniciativa MEDES], onde também existiam algumas pessoas que tinham essa formação, mas eram minoria. (Cláudio Almeida, em entrevista concedida ao autor em 26.06.2006).

Não havia uma configuração de forças que cabalmente instituísse ou refutasse a necessidade da TI/BNDES estabelecer um processo de software mais formalizado. O assunto não ocupava de forma determinante a pauta das reuniões do comitê de TI, sempre às voltas com projetos críticos e urgentes, como atendimento à regulamentações legais e o suporte a novos produtos. No entanto, no início de 2002, o Banco Central do Brasil (BACEN), órgão que regulamenta diversos aspectos das instituições financeiras, fortaleceu o papel do grupo de trabalho liderado por Cláudio Almeida, pois, baseado em uma auditoria que realizou em 2001, apontou a necessidade do BNDES estabelecer maior controle sobre seu processo de software.

Em meados de 2002, sempre com o impulso de Cláudio Almeida, o grupo entendeu ser apropriado instituir um processo de software aderente ao CMM — *Capability Maturity Model* (naquela época o CMMI ainda não tinha sido lançado). A idéia do grupo era contratar uma consultoria que implantasse um processo de software e que a TI/BNDES obtivesse uma avaliação CMM Nível 2 — processo gerenciado — num prazo de vinte e quatro meses. Além disso, deveriam ser adquiridas ferramentas CASE — *Computer Aided Software Engineering* — para suportar o processo de desenvolvimento de software. "[Antônio] *Faoro dá carta branca para a estruturação do projeto com a consultoria*" (Cláudio Almeida, em entrevista concedida ao autor em 26.06.2006). Houve um detalhamento do que seria exatamente o projeto para que pudesse ser iniciada a contratação da consultoria. No entanto, no final daquele ano de 2002, devido a alternância de poder no governo federal, os esforços desse grupo de trabalho ficaram suspensos.

No início de 2003, sob um novo governo federal e uma nova administração, o BNDES tornou a passar por mais uma profunda reestruturação organizacional. Novamente o impacto na área de informática foi muito expressivo, pois a recém criada área de TI foi extinta e as três gerências executivas que a compunham passaram novamente a se subordinar à Área de Administração, posteriormente denominada Área de Administração e Informática (AAI). Além disso, voltaram a se chamar departamentos: DESIS — Departamento de Sistemas —, DEOPI — Departamento de Operações e Infra-estrutura — e DEINP — Departamento de Integração de

Processos. Essa mudança, no entanto, tornou o cenário mais favorável à implantação de um processo de software, como veremos. A figura dos assessores de TI em todas as áreas de negócio também foi extinta, e a maioria deles foi lotada no "novo" DESIS como gerentes de desenvolvimento de sistemas. A própria chefia desse departamento foi ocupada pela (ex) assessora de TI Lílian Mendes, que havia integrado o grupo de trabalho coordenado por Cláudio Almeida. Este último, teve sua influência bastante aumentada, pois tornou-se o único assessor para assuntos de informática do novo superintendente da AAI, o engenheiro Nelson Duplat.

Investido de suas novas atribuições, em março de 2003, Cláudio Almeida conseguiu a criação de um SEPG — *Software Engineering Process Group* — composto por dois integrantes, como ele mesmo relata. "*Muda o governo, o* [Antônio] *Faoro saiu, virei assessor do novo superintendente,* [Nelson] *Duplat, e remanejamos uma gerência de informática para cuidar especificamente da criação do processo de software*" (Entrevista concedida ao autor em 26.06.2006). Cabe notar que os desenvolvedores do BNDES, em sua esmagadora maioria, sequer sabiam da existência deste grupo que buscaria implantar um processo mais formalizado para guiar o desenvolvimento de software. Até mesmo o gerente designado para o SEPG, Sérgio Viveiros, analista de sistemas que entrou no BNDES no início dos anos 1990, tomou conhecimento da questão apenas no momento em que recebeu a designação.

> Para mim foi uma surpresa muito grande, lá em 2003 quando isso aconteceu. Nem sei em qual mudança foi, pois de 2000 para cá mudamos tantas vezes... Quando o [Nelson] Duplat assumiu, veio o convite para assumir a gerência criada para tratar de engenharia de software. Um assunto do qual estava totalmente afastado, não tinha mais nada a ver com isso. À época estava cuidando do Helpdesk, nem estava mais próximo do desenvolvimento em si, estava no DEOPI. [...] Eu não acompanhei o trabalho do grupo [coordenado por Cláudio Almeida], não vi nenhuma atuação dele e sequer era minha área de interesse na época. (Entrevista concedida ao autor em 10.08.2006).

O outro integrante do SEPG é o autor deste trabalho, que pediu para participar do projeto, tendo acompanhado as últimas reuniões do grupo liderado por Cláudio Almeida, ainda em 2002. A descrição da gênese do SEPG é importante para entendermos os desdobramentos do projeto de implantação de processo de software (IPS) no BNDES. O SEPG foi criado com *status* de gerência na estrutura organizacional e, doravante, será referenciado pela denominação com que é conhecido internamente no BNDES: GEPES — Gerência de Processo de Engenharia de Software — vinculada ao DEINP.

Os dois integrantes da GEPES, ambos com mais de dez anos de experiência como desenvolvedores de software e com uma vivência de pelo menos cinco anos na cultura de desenvolvimento de sistemas no DESIS do BNDES, coerentes com sua

formação teórica em ciência da computação, entenderam que o esforço de IPS demandaria, de um lado, especialização técnica no campo da engenharia de software e, de outro lado, uma mudança cultural na organização. Trataria-se um esforço de implantação de um modelo, no caso o CMMI, para que a difusão das melhores práticas supostamente nele incorporadas pudessem ocorrer, mas, para isso, teria de haver também uma mudança cultural. A partir dessa interpretação para o problema, e ainda desconhecendo as questões abordadas na seção 3.3 sobre os modelos de *difusão* e *tradução*, a GEPES estabeleceu suas diretrizes, dividindo-as nas duas categorias a seguir:

Diretrizes técnicas:

- capacitação no CMMI;
- contratação de consultoria especializada;
- viabilização de orçamento para o projeto;
- garantia da adequação "técnica" do processo à realidade do BNDES.

Diretrizes relacionadas à mudança cultural:

- patrocínio;
- envolvimento dos desenvolvedores;
- participação;
- comprometimento dos envolvidos.

Desde sua criação, a GEPES foi se convencendo daquela que seria sua principal premissa: envolver os desenvolvedores induzindo-os a estabelecerem eles mesmos o processo de software. Seu primeiro contato com os desenvolvedores aconteceu entre os meses de abril e julho de 2003, quando quase todos foram entrevistados. Na ocasião optou-se por não entrevistar os gerentes, tendo ocorrido 28 entrevistas com os técnicos. Havia um duplo objetivo nessas entrevistas. O primeiro era apresentar a GEPES e sua missão de implantar um processo de software no BNDES. O segundo objetivo era construir um *diagnóstico* (McFEELEY, 1996) da situação existente. O diagnóstico realizado indicou que seria apropriado utilizar a abordagem CMMI[88], pois vários problemas tratados em áreas específicas desse modelo surgiram como questões relevantes nas entrevistas, como, por exemplo (GEPES, 2003a): a necessidade de formalização do planejamento e acompanhamento de projetos, indicado por 78,6% do entrevistados; a coordenação

[88] Na verdade, após o diagnóstico realizado em 2003, o modelo de melhoria escolhido foi o CMM, antecessor do CMMI, pois, àquela altura, o CMMI mal havia sido lançado. Como a execução do projeto IPS demorou para se iniciar, mais tarde optou-se por utilizar o modelo CMMI, dado que o CMM seria descontinuado pelo o SEI — *Software Engineering Institute*.

entre grupos (particularmente a existência de problemas de comunicação), indicado por 67,9% dos entrevistados; o gerenciamento eficiente dos requisitos, indicado por 57,1% dos entrevistados; a gerência de configuração e garantia da qualidade, indicado por 21,4% dos entrevistados.

Após formar sua própria visão acerca do que seria o esforço de IPS, a GEPES confrontou-se com seu primeiro desafio. O grupo de trabalho liderado por Cláudio Almeida já havia estabelecido uma estratégia para a abordagem do problema. Esse grupo havia referendado um planejamento detalhado para o projeto de IPS produzido por uma consultoria que o apoiou. Basicamente, caberia à GEPES apenas contratar uma consultoria para executar o que estava planejado.

> Na verdade, já existia uma IP [Instrução Padronizada, instrumento interno ao BNDES que, dentre outras coisas, instrumentaliza a proposição de contratações para a Diretoria] praticamente pronta para contratação de uma consultoria que implantaria CMM. Isso me foi passado na reunião [em que me designaram gerente da GEPES]. [...] O foco maior era a contratação da consultoria. (Sérgio Viveiros, em entrevista concedida ao autor em 10.08.2006).

No entanto, ocorreu um forte desacordo entre o escopo, os objetivos, a estratégia de abordagem e as premissas contidos naquele planejamento e a visão da GEPES sobre o que deveria ser um esforço de IPS no BNDES. Assim, a GEPES teve que iniciar um processo de negociação sobre os rumos do projeto IPS, patrocinado pela chefia do DEINP, departamento ao qual estava vinculada, com o próprio Cláudio Almeida, então o único assessor do superintendente, e a chefia do DESIS. Compreensivelmente, o projeto IPS ficou paralisado enquanto não surgia um acordo. Nesse período, Cláudio Almeida demandou da GEPES a estruturação de um curso sobre gerenciamento de projetos, cujo resultado foi o treinamento dos chefes de departamento de informática e de todos os gerentes e líderes de projetos entre os meses de outubro e dezembro de 2003. O curso estruturado pela GEPES serviu como padrão para treinamento sobre gerenciamento de projetos no BNDES, tendo existido várias turmas nos anos seguintes.

Paralelamente, Cláudio Almeida organizou e coordenou um processo de planejamento participativo da informática, o PDI–2003, que demandou um enorme envolvimento de todos os profissionais de informática e de muitos integrantes de áreas de negócio, entre maio e agosto de 2003. Portanto, além de paralisado por conta do desacordo em relação à estruturação original do projeto IPS, as demandas do PDI–2003 sobre toda a informática, incluindo a GEPES, inviabilizaram qualquer progresso do IPS. Todavia, ocorreu mais uma troca de comando no BNDES, em outubro de 2003, e o então superintendente da área de Área de Administração e

Informática (AAI), Nelson Duplat, deixou o cargo. Logo depois, Cláudio Almeida também deixou a assessoria de informática transferindo-se para outra área.

Com a saída de Nelson Duplat e Cláudio Almeida, fazia-se necessária uma "*reafirmação do patrocínio*" (McFEELEY, 1996, p.4) para viabilizar os trabalhos do IPS. Infelizmente, porém, a nova gerência sênior sequer recebeu a GEPES para se inteirar do projeto. Naquele momento, não foi destacado nenhum assessor de informática, embora tenha sido deliberado que se consolidasse todo o esforço coletivo despendido no processo participativo de planejamento. Em janeiro de 2004 ocorreu a consolidação do PDI–2003, que destacou o projeto IPS como um dos projetos estruturantes que deveria ser executado prioritariamente:

> Implementar, *com a máxima urgência*, uma metodologia de desenvolvimento de sistemas e gerência de projetos (sem a qual os novos sistemas correm o risco de padecer dos mesmos problemas que os atuais). (PDI-2003, p.30, grifo nosso).

Após a consolidação do PDI–2003, a GEPES recorreu ao nível gerencial imediatamente abaixo da gerência sênior na tentativa de negociar o escopo e a estratégia de condução do projeto. Após quase um ano desde sua criação, em grande parte decorrente do envolvimento demandado pelo PDI–2003, além de sua capacitação "técnica" e a estruturação do curso de gerenciamento de projetos, a única coisa que de fato a GEPES havia conseguido até aquela ocasião foi convencer a chefia do DESIS de que a abordagem proposta pelo grupo de trabalho liderado por Cláudio Almeida precisava ser revista, ao menos pelas seguintes razões:

- era direcionado apenas à plataforma baixa, num cenário em que o ambiente *mainframe* representava mais de 80% dos esforços de desenvolvimento e manutenção de sistemas;
- um processo de software definido externamente por uma consultoria, sem a participação efetiva do corpo de desenvolvedores, portanto focalizando mais o próprio modelo escolhido do que a realidade do desenvolvimento de sistemas no BNDES, apresentava um grande risco de inadequação à realidade, logo um grande risco de ser apenas prescrito e não seguido pelos desenvolvedores.

A GEPES explicitou sua proposta de definir e implantar o processo de software de forma incremental e participativa[89]. Na prática, somente a partir deste momento o DESIS pautou a discussão sobre o esforço de IPS, com participação ativa de alguns de seus gerentes de linha e técnicos. No entanto, instaurou-se uma enorme

[89] Nota Técnica nº 03/2003 da AAI/DEINP em 12.09.2003 (GEPES, 2003b).

controvérsia, posto que esse reduzido mas influente grupo de gerentes de linha do DESIS defendia a rápida aquisição de um conjunto de ferramentas CASE — *Computer Aided Software Engineering* — e a adaptação de um processo de software "padrão de mercado", implantando-o através de projetos piloto. Além de não seguir a ordem recomendada no CMMI, tratava-se de uma abordagem de tentativa e erro, que ignorava o valor da participação ativa dos desenvolvedores e acreditava que o processo de software padrão ("universal"), por si só, seria o garantidor dos benefícios esperados. Eram inócuos argumentos como a "*necessidade de envolver todos ao longo do processo de mudança*" (SOMMERVILLE, 2004, p.680) e que o processo de software padrão não garante nada por si só:

> Padrões e procedimentos são a base para a gerência de qualidade, porém *gerentes de qualidade experientes reconhecem que existem aspectos intangíveis sobre a qualidade do software* [...] *que não podem ser corporificados nos padrões*[lvi] (ibid., p. 642, grifo nosso).

Instaurou-se um (novo) impasse que durou meses sem que argumentos "técnicos" resolvessem a situação. Não havia ainda sequer um planejamento para o projeto IPS, nem seu escopo estava negociado. Com a demora, até o entusiasmo que os desenvolvedores mostraram no início do projeto estava acabando. Embora a capacitação "técnica" no CMMI tenha sido alcançada com facilidade pela GEPES, esta não conseguiu, ao longo do projeto, materializar sua premissa de envolvimento dos desenvolvedores. Resultaram inócuas as tentativas de motivação, envolvimento e convencimento das pessoas em favor do projeto de IPS. Fugiam do alcance da GEPES as recomendações de praxe:

- Obtenção do apoio de um "campeão" — não existia nenhum líder natural no grupo que pudesse ser tomado como exemplo de dedicação ao esforço de definição e/ou uso de processo de software.
- Vinculação do projeto IPS à estratégia da área de informática da empresa — num curto período de menos de dois anos, a informática do BNDES obteve, e perdeu logo em seguida, o *status* de área autônoma na organização. Mesmo com a realização do PDI–2003, dada a descontinuidade na gerência sênior, não era claro qual seria a visão de futuro que a informática teria.
- busca de patrocínio — a gerência sênior que criara a GEPES tinha sido destituída e a atual, como foi dito, sequer quis recebê-la para se inteirar do projeto (foi a única com esta postura).

Na tentativa de materializar algum produto, a GEPES, que já havia viabilizado o treinamento em gerência de projetos para mais de trinta profissionais da informática,

criou alguns modelos de documentos para auxiliar o trabalho dos gerentes e líderes de projetos. Mesmo sem prescrever um processo para o planejamento, monitoração e controle de projetos, esses modelos passaram a ser utilizados por boa parte dos gerentes e líderes, atuando positivamente para a GEPES.

Neste período, final de 2003, a gerência sênior da época determinou que, em um futuro próximo, fábricas de software fossem contratas pela informática do BNDES. O DESIS partiu então em busca de uma consultoria para ajudá-lo a estabelecer uma forma segura de operar com fábricas de software. Essa consultoria apresentou um parecer sobre a forma de operação futura, muito aderente à proposta da GEPES de estruturação do processo de software. Associando-se a esses dois novos atores — parecer da consultoria e definição de se usar fábricas de software —, a GEPES reiterou sua proposta para o projeto IPS com pequenas modificações. Assim foi possível estabelecer o seguinte enredamento: rejeitar o escopo e a estratégia que a GEPES propunha equivaleria, para o DESIS, a rejeitar a utilização segura de fábricas de software na empresa, dada a coincidência entre o parecer da consultoria e a proposta da GEPES. O grupo de gerentes de linha que divergia da proposta da GEPES ficou numa situação em que negá-la equivaleria a negar a determinação da gerência sênior de viabilizar a utilização de fábricas de software. Como resultado, somente assim, quase um ano depois de instituída, a GEPES, que não obteve sucesso através de argumentações "técnicas", finalmente conseguiu estabelecer um acordo que julgava adequado sobre o projeto IPS e pôde, em dezembro de 2003, iniciar o processo de contratação da consultoria para o projeto.

Esse enredamento constituído pela GEPES constituiu um claro movimento de *tradução* (seção 2.3) bem sucedido. De fato, talvez este tenha sido o momento em que o IPS esteve mais próximo de uma tomada de força rumo à estabilização da rede sociotécnica do processo de software que buscava implantar. Contudo, diversos outros atores que também deveriam ter sido enredados não geraram novas *traduções* apropriadas à conseqüente estabilização da rede desejada. Pouco tempo depois de conseguir negociar o escopo e a estratégia para o projeto IPS, ocorreu mais uma nova troca de gerência sênior que desfez a ordem precária que havia surgido com a *tradução*: usar fábrica de software com segurança significava realizar o projeto IPS. Mas, naquela altura, novos aliados atuavam dificultando o ressurgimento do impasse, dentre eles, o plano para o projeto, que já estava pronto e tacitamente aceito, e a contratação já em andamento da consultoria para o projeto IPS.

Poderíamos achar que a situação começava a melhorar para a GEPES. No entanto, não havia patrocínio, prestígio e liderança reconhecida pelo grupo de desenvolvimento, com a quase totalidade dos desenvolvedores, devido ao longo prazo

transcorrido desde a criação da GEPES, já sem a menor motivação para com o IPS. Para complicar ainda mais, com a última troca de gerência sênior, os três departamentos de informática foram transferidos para uma outra área, a recém criada Área de Controle (ACO), e, conseqüentemente, a informática passou a ser orientada por outra assessoria jurídica. Em abril de 2004, com tudo pronto para a GEPES proceder a contratação da consultoria de apoio ao projeto IPS, a nova assessoria jurídica (da ACO) discordou do encaminhamento do processo que havia sido orientado pela assessoria jurídica anterior (da extinta AAI). Resultado: mal acabara de ser planejado, o projeto IPS amargou 5 meses de atraso. O cenário só piorava!

Resolvido o problema da contratação, somente em agosto de 2004 pode-se dizer que teve início a execução do projeto IPS, porém numa situação bastante desfavorável. Não bastasse tratar-se de um projeto que normalmente enfrenta muita resistência, uma vez que visa à mudança na relação dos desenvolvedores com seu próprio trabalho, àquela altura o entusiasmo inicial observado em boa parte dos desenvolvedores já havia esmaecido. A imagem da GEPES já estava bastante desgastada[90].

É importante notar que, quando finalmente a execução do IPS teve início, a GEPES encontrava-se de certa forma sem lastro frente a um problema reconhecido pelo grupo como um problema real. Na gerência sênior não havia mais a percepção, que tivera Cláudio Almeida, sobre a importância da implantação de um processo de software no BNDES. No nível operacional, os desenvolvedores não tinham mais a expectativa que a GEPES pudesse viabilizar alguma solução para os problemas que eles apontaram nas entrevistas, realizadas há mais de um ano.

A GEPES tinha que planejar o projeto de IPS, pois havia sido criada exatamente com a incumbência de materializar um processo de software mais formalizado, um processo que, dentre outras coisas, defendia exatamente a necessidade de planejamento para os projetos. Contudo, encontrando-se sem amparo efetivo da gerência sênior e sem reconhecimento do nível operacional, coube a ela própria declarar, no plano do projeto IPS, as necessidades existentes no DESIS que supostamente ensejariam um esforço de IPS. Embora seus integrantes tivessem vivência no ambiente do DESIS e embora estivessem fundamentados no diagnóstico feito entre abril e junho de 2003, o fato da própria GEPES caracterizar o problema, estabelecer a solução e os benefícios e vantagens que se poderiam esperar gerou

[90] Ao longo de seu primeiro ano e meio de existência, a GEPES foi incumbida também de estruturar e iniciar a execução de um projeto de redocumentação de um sistema legado. Em julho de 2004, sem gerar documentação alguma, embora tenha demandado bastante sua dedicação, este projeto foi abortado. Logo, serviu apenas para consumir esforços da GEPES e desgastar ainda mais sua imagem.

uma tensão maior ainda com a premissa que tanto valorizava — garantir o envolvimento de todos. Na luta para viabilizar o projeto IPS, mesmo negociando exaustivamente seu escopo e sua estratégia, o problema em si e o que deveria ser sua solução acabou sendo unilateralmente declarado pela própria GEPES. O DESIS "aceitou" essa declaração quando aprovou o plano para o projeto. O que se percebe retrospectivamente, é uma aceitação tácita, não unânime e efetiva, ambivalente, como fazia esperar a controvérsia existente sobre a importância ou não de se estabelecer um processo de software formalmente definido. Nas palavras de Felipe Calheiros, analista que ingressou no BNDES no final dos anos 1990, percebe-se a ambivalência: o DESIS, como ente institucional, aceitava a necessidade do processo de software, porém, os profissionais que o compunham não pensavam tanto assim:

> Acho que isso [a importância de termos um processo de software formalizado] acaba sendo uma opinião pessoal de cada um. É quase uma "guerra santa" entre umas pessoas que acreditam e outras que não acreditam. Das que acham que o certo é fazer do jeito que elas acham certo e das outras que acham que o certo [...] é ter uma padronização. Acho que pela quantidade de resistência que se tem ao processo, parece existir uma parcela significativa que não o quer. Acho que como grupo queremos. Mas acho que a [própria] chefia do DESIS não queria, mas ia junto com os outros, [ou seja, seguia o movimento já iniciado de implantação de processo]. (Entrevista concedida ao autor em 07.08.2006).

Mesmo tutelados pelo modelo de difusão (seção 3.3), ainda que não estivessem conscientes a respeito, os integrantes da GEPES valorizavam mais a situação local do que o modelo de processo, ou de melhoria de processo, a ser seguido. Denotam-no tanto o episódio em que a GEPES discordou do escopo e objetivo da abordagem estabelecida pelo grupo de trabalho liderado por Cláudio Almeida, quanto o impasse gerado com os defensores da idéia de se adquirir um processo de software "padrão de mercado" e implantá-lo através de projetos piloto. Mas, ao declarar o problema, a possível solução e os supostos benefícios, a GEPES acabou pautando-se mais nas promessas do modelo de melhoria que iria seguir do que na realidade local existente, aceitando tacitamente que a implantação do modelo levaria à difusão dos benefícios "tecnicamente" por ele garantidos, desde que fosse criado o adequado "contexto" de implantação.

A estratégia de condução do projeto IPS que a GEPES conseguiu aprovar com o DESIS revela bem a aceitação da idéia de divisão entre contexto e conteúdo, típica do *difusionismo*. Considerou-se que os fatores determinantes para o sucesso do projeto orbitavam o "contexto" organizacional, cultural, social, político no qual o conteúdo "técnico" — o CMMI — deveria ser "implantado". No plano de ação do projeto IPS (GEPES, 2004) pode ser visto que sua estratégia está baseada em um

tripé, onde os dois primeiros elementos focalizam o "contexto" de implantação, enquanto o último focaliza o "conteúdo" a ser implantado, conforme se segue:

1. *Construção participativa do processo de software* — de forma que todos no DESIS deveriam ser envolvidos nos trabalhos de concepção e implantação do processo no BNDES. À GEPES caberia a proposição inicial do processo e sua consolidação final a partir da reação do DESIS à proposição feita;
2. *Implantação incremental do processo de software* — como tentativa de infligir menor sobrecarga ao trabalho do DESIS e uma mudança cultural mais suave e diluída ao longo do tempo;
3. *Utilização do CMMI como modelo de referência* — a conformação final do processo de software deveria estar aderente ao CMMI, pois, além do diagnóstico realizado em 2003 ter sugerido a propriedade do uso desse modelo, em abril de 2004, o então superintendente, Antônio Miguel, afirmou que "*mesmo o projeto IPS não tendo a avaliação oficial CMMI como meta primária, seria importante viabilizar essa possibilidade que teria um certo valor para o BNDES, nas operações internacionais de captação de recursos*".

Fiel a sua principal diretriz de induzir os próprios desenvolvedores a prescrever e estabelecer o seu processo de software, a GEPES propôs, e viu ser criado, um grupo que ficou conhecido como *grupo homologador*. O grupo homologador foi criado com possibilidades efetivas de gestão no projeto IPS, concretamente podendo decidir, ainda que de forma colegiada, sobre seu rumo. O grupo homologador era coordenado pela chefe do DESIS, Lilian Mendes, e tinha mais quatro gerentes e dois técnicos do DESIS como integrantes. Também fazia parte do grupo homologador a então assessora de informática.

O nome "*grupo homologador*" deriva da responsabilidade que teria de homologar o processo na medida em que fosse sendo prescrito. A idéia era que esse grupo garantisse ao projeto IPS sua pertinência ao próprio DESIS e não à GEPES. O grupo homologador deveria ser o primeiro a defender e a difundir a importância de se estabelecer um processo de software mais formalizado, pois era composto por uma parcela significativa de pessoas que tinham a incumbência de gerir equipes de desenvolvimento, além da própria chefe do DESIS.

Outra idéia era que o grupo homologador permitisse maior agilidade na discussão das proposições feitas pela GEPES. Antes de abrir a discussão para todos os desenvolvedores, efetivamente garantindo-lhes a possibilidade de sintetizar o processo de software que julgassem mais adequado, a GEPES discutia suas propostas com o grupo homologador. Por ser um grupo menor, as discussões

poderiam ocorrer mais rapidamente e possibilitar melhorias, ajustes e direcionamentos que seriam incorporados antes da discussão geral com todos os desenvolvedores. Também foi criada uma lista eletrônica de discussões para facilitar a participação.

Porém, nem o grupo homologador nem a lista de discussões foram aliados da GEPES no sentido de perfazerem as atuações por ela esperada, qual seja, a de induzir o desenvolvimento participativo do processo de software.

A primeira tarefa planejada do IPS a ser executada foi a realização de palestras informativas sobre processos de software, ministradas pela consultoria contratada. Ao todo foram 11 palestras. A primeira, introdutória, versava sobre a engenharia de software de forma mais geral. Da segunda em diante, foram abordados tópicos específicos relacionados com o processo de software, como engenharia de requisitos, testes, medições, qualidade. Embora fosse bom o quórum das palestras, o clima para o início do projeto, como sabido, não era adequado. Havia uma insatisfação e estranheza capturadas em comentários do tipo: "*tanto tempo* [um ano e meio após a criação da GEPES] *para se fazer só isso* [as palestras]!?" Era patente a falta de entusiasmo dos desenvolvedores com o projeto IPS.

À medida que as palestras ocorriam, a GEPES iniciou a prescrição dos subprocessos de planejamento de projetos (PP) e de monitoração e controle de projetos (MCP), adaptados às necessidades do BNDES. O projeto IPS previa dezoito subprocessos, correspondentes às dezoito PA's — *process areas* — dos níveis 2 e 3 do CMMI[91]. PP e MCP foram os primeiros subprocessos definidos. Naquela ocasião transpareceu o primeiro erro de estimativa incorrido no planejamento do IPS. A GEPES estimou que faria a adaptação dos subprocessos PP e MCP em dois meses e, por diversas razões, levou seis meses para fazê-lo. Durante todo esse período, ela distanciou-se novamente dos desenvolvedores, periodicamente reunindo-se apenas com o grupo homologador.

Quando os subprocessos PP e MCP, já aceitos pelo grupo homologador, ficaram prontos para serem discutidos com os desenvolvedores e a GEPES os apresentou, a esperada discussão não aconteceu, ou seja, a esperada participação dos desenvolvedores não se materializou. Mesmo se tratando do primeiro momento em que o grupo de desenvolvedores teria a chance de moldar efetivamente a seu

[91] O CMMI, em sua *representação por estágios*, estabelece as seguintes *áreas de processo* para se atingir o nível 2 de maturidade: gerenciamento de requisitos; planejamento de projetos; monitoração e controle de projetos; gerência de acordo com fornecedores; medição e análise; garantia da qualidade do processo e do produto; e gerência de configuração. Para o nível 3 de maturidade, acrescentam-se: desenvolvimento de requisitos; solução técnica; integração do produto; verificação; validação; foco no processo organizacional; definição do processo organizacional; treinamento organizacional; gerenciamento de riscos; análise de decisão e resolução; ambiente organizacional para integração. (CHRISSIS et al., 2003).

gosto os subprocessos que estavam sendo propostos, a GEPES não conseguiu mobilizar a participação que desejava e atuar como um catalisador para o estabelecimento do processo de software. A GEPES foi incapaz de constituir *traduções* que levassem os desenvolvedores a atuar segundo a premissa que estabelecera: induzir eles próprios a definirem o processo. Nem mesmo mensagens postadas na lista de discussão eletrônica criada para o projeto IPS tinham eco. A ausência de discussão revelava enfaticamente a situação da GEPES e do IPS.

Buscando novos aliados, a GEPES pediu apoio, em dezembro de 2004, ao departamento de comunicação do BNDES que contratou uma consultoria especializada em comunicação e gestão de mudanças para ajudá-la. A consultoria entrevistou a gerência média, a gerência de linha e os desenvolvedores. Descortinou-se o já sabido: àquela altura não havia resistência ao projeto IPS porque a maioria das pessoas sequer considerava que ele continuava existindo. Vários desenvolvedores acreditavam que a GEPES em breve seria extinta, pois quase dois anos haviam se passado e, na prática, foram realizadas apenas algumas palestras e prescritos os subprocessos relacionados ao gerenciamento de projetos.

A consultoria esclareceu que era crucial considerar as *individualidades* dos desenvolvedores. É certo que deveriam ser convencidos da importância do projeto IPS para a organização, porém a GEPES deveria mostrar para cada um os benefícios pessoais que poderia ter. Era necessário considerar também valores, expectativas e anseios individuais dos desenvolvedores. Essa recomendação, de certa forma, convidava a uma interpretação do problema de forma menos simplificada do que a divisão em um "conteúdo técnico" a ser implantado no "contexto organizacional". Ora, considerar individualizadamente os desenvolvedores implica sensibilidade para suas atuações e relações com os demais atores envolvidos com o IPS. Trata-se de uma perspectiva bastante diferente de considerar o "contexto organizacional" como uma essência passível de manipulação, uma totalidade que engloba desenvolvedores e demais "componentes do contexto" de implantação do processo de software. Diferente, portanto, da perspectiva da GEPES que via o "contexto organizacional" exatamente como uma totalidade que se buscava sujeitar à mudança cultural para que o "conteúdo técnico" pudesse ser implantado adequadamente.

Em janeiro de 2005, ocorreu mais uma troca de gerência sênior. O novo superintendente, o contador Luiz Dorneles, se mostrou bastante favorável ao projeto IPS. Orientada pela consultoria de comunicação, a GEPES promoveu uma (re)apresentação do projeto IPS para todos os desenvolvedores, contando com a participação do superintendente, das chefias do DESIS e do DEINP (o departamento da GEPES) e do grupo homologador.

Em abril de 2005, o novo superintendente deliberou a criação de um escritório de projetos[92], e, pouco depois, estabeleceu formalmente os subprocessos relacionados ao gerenciamento de projetos (PP e MCP) como norma interna de procedimento no BNDES para os departamentos de informática[93]. A gestão desses subprocessos passaria à competência do escritório de projetos. Surgia, após muito tempo, o que se podia interpretar novamente como uma ambiência favorável ao projeto.

Reanimada, a GEPES vislumbrou novamente a possibilidade do projeto IPS contar com o respaldo da gerência sênior e tomar um ritmo renovado de execução com a atuação do recém criado escritório de projetos. A próxima atividade que ela previa desempenhar, e que acabou sendo transferida para o escritório de projetos, seria o treinamento e o assessoramento dos gerentes e líderes de projeto nos subprocessos PP e MCP. Para a GEPES não era simplesmente uma questão de ministrar um curso sobre PP e MCP. Era fundamental proporcionar assessoria relacionada a esses subprocessos aos gerentes e líderes de projeto à medida que precisassem, favorecendo assim sua aculturação no novo processo de trabalho. Mas o escritório de projetos, por sua vez, não se aproximou dos objetivos da GEPES e não viabilizou esse assessoramento. Conseqüentemente, não houve atuação alguma no sentido de favorecer a mudança na forma de trabalhar dos gerentes e líderes de projetos da informática. O escritório de projetos entendeu que seu papel não deveria ficar restrito apenas aos departamentos de informática, mas deveria focalizar também as áreas de negócio do BNDES, e concentrou-se na definição de uma metodologia para o gerenciamento de projetos no BNDES como um todo, replicando parte dos esforços despendidos na elaboração de PP e MCP, que focalizavam exclusivamente os departamentos de informática. A conseqüência para o projeto IPS foi que, não bastasse ter demorado tanto para definir esses dois subprocessos, sua implantação efetiva, através da aculturação das equipes, não foi trabalhada.

Os gerentes e líderes percebiam a atuação do escritório de projetos e a "obrigatoriedade" de PP e de MCP apenas em decorrência de um aplicativo implantado pelo próprio escritório de projetos. Esse aplicativo atuava disponibilizando na intranet do BNDES as informações sobre os projetos executados pelos departamentos de informática. Os gerentes e líderes de projeto preenchiam os vários documentos estabelecidos nos subprocessos de PP e MCP e tornavam-nos visíveis

[92] Um escritório de projetos (do inglês *Project Management Office*) é uma unidade organizacional que centraliza e coordena o gerenciamento de projetos sob seu domínio (PMBoK, 2004, *A Guide to the Project Management Body of Knowledge*. Project Management Institute Inc.).
[93] INSTRUÇÃO DE SERVIÇO ACO/SUP Nº 02/2005.

através do aplicativo do escritório de projetos. Como não existiu o assessoramento no processo e houve uma certa cobrança de publicação das informações no aplicativo, transpareceu a idéia de que seguir os processos PP e MCP era uma tarefa adicional, que não agregava valor, uma mera exigência burocrática de preenchimento de documentos. Alguns interpretaram que o gerente do projeto, além de realizar suas tarefas, teria agora a atividade adicional de preencher e publicar os documentos solicitados, seguindo os modelos preestabelecidos. Era mais um duro golpe nos esforços da GEPES, pois, além de perder a prerrogativa de assessorar as pessoas no processo de gerenciamento de projetos, foi incapaz de compor com o escritório de projetos uma atuação alinhada com seus (da GEPES) objetivos. Na prática, muitos viram a idéia de processo formalizado apenas como burocracia e tentativa de controle.

Enquanto isso, a GEPES estava encerrando a prescrição de dois outros subprocessos — gerenciamento / desenvolvimento de requisitos e gerenciamento de acordos com fornecedores — quando, em meados de 2005, o BACEN reiterou sua cobrança por maior formalização e especialização de papéis no desenvolvimento de sistemas do BNDES. Reconhecendo que somente no longo prazo o IPS — que já acumulava expressivo atraso em relação ao que havia sido informado ao BACEN após a consolidação do PDI–2003 — iria atender às suas recomendações, o BACEN recomendou fortemente que o BNDES definisse e implantasse um processo simplificado que garantisse, ainda que de forma incipiente, uma maior formalização de seu processo de desenvolvimento de sistemas. Em resposta, o BNDES se comprometeu a institucionalizar efetivamente os subprocessos PP, MCP e gerenciamento de requisitos em curto prazo. No entanto, o BACEN não reconheceu esses três subprocessos como um processo simplificado nos termos a que se referia, pois não seriam contemplados aspectos de diversos outros subprocessos essenciais a uma formalização mínima do processo de software. Por isso, em julho de 2005, as chefias do DESIS e do DEINP deliberaram a paralisação do IPS e o estabelecimento, pela GEPES, de um processo simplificado que atendesse às exigências do BACEN em curto prazo.

Sabendo da total impossibilidade de prescrever e implantar um processo simplificado em curto prazo, haja vista os resultados do próprio IPS, foi apresentada à chefe do DEINP, Margarida Sa Freire, uma proposta em que deveria caber à GEPES apenas a prescrição do processo simplificado. A implantação, sobretudo no curto prazo como recomendado pelo BACEN, deveria ficar a cargo da própria chefia do DESIS, a única instância hierárquica que poderia dispor de mecanismos concretos para estabelecer o cumprimento, pelos desenvolvedores, de um processo imposto, qualquer que fosse ele. Mesmo porque um dos pontos reiterados pelo BACEN, muito

além da mera prescrição de um processo de software, era a necessidade de especialização de papéis e funções, o que possibilitaria a existência de uma prática de desenvolvimento de software mais formalizada. Como isso era matéria relacionada à organização interna do DESIS, nem o DEINP nem a GEPES teriam a menor possibilidade de ingerência a respeito. A chefe do DEINP acatou o argumento viabilizando que a GEPES atuasse apenas na prescrição de um processo simplificado, e que caberia ao DESIS a responsabilidade por sua implantação.

A prescrição do processo simplificado foi aceita formalmente pelo DESIS em janeiro de 2006. O BNDES informara ao BACEN que implantaria o processo simplificado até junho de 2006, meta que acabou não sendo atingida. O que de fato aconteceu em junho de 2006 foi mais uma troca de gerência sênior, a última no espaço deste relato. Desta vez, além do superintendente e de sua assessoria de informática, mudaram também as chefias de todos os departamentos de informática. Como conseqüência, até agosto de 2006 ainda não estava claramente estabelecido qual papel a GEPES teria dali em diante. A situação parecia indicar que:

- o IPS como fora planejado não seria continuado;
- os assuntos relacionados aos subprocessos de gerenciamento de projetos (PP e MCP), que estavam no escritório de projetos, voltariam à tutela da GEPES. Com isso a GEPES poderia, finalmente, realizar o assessoramento aos gerentes e líderes de projeto, favorecendo a implantação efetiva desses subprocessos;
- os subprocessos de requisitos e gerenciamento de acordos com fornecedores (vide nota 91), já prescritos, seriam homologados e também se disponibilizaria treinamento e assessoramento adequados à sua implantação efetiva;
- a implantação do processo simplificado voltaria a ser atribuição da GEPES, e, supostamente, as condições necessárias à consecução dessa empreitada seriam garantidas pela gerência sênior.

No entanto, como a situação toda continuava extremamente indefinida até o final de 2006, somado a todo desgaste acumulado no período, por conta de não se ter conseguido até então institucionalizar um processo de software para o BNDES, os dois integrantes da GEPES solicitaram seu desligamento do projeto IPS em dezembro de 2006. Como passou a haver o entendimento de que o projeto IPS tornara-se obrigatório, em decorrência das recomendações reiteradas em auditorias do BACEN, o desligamento dos integrantes da GEPES não pôde ser imediato, ocorrendo somente em março de 2007, quando foram transferidos para outros projetos.

Finalizando esta narrativa, que sob o enfoque dos *Estudos CTS*, poderia-se dizer uma narrativa de "não-traduções", chama muito a atenção que, desde a criação do projeto IPS até o desligamento de seus dois integrantes originais, o tempo todo a GEPES esteve envolvida em batalhas que a engenharia de software tradicionalmente difusionista julgaria políticas, sociais, culturais e organizacionais. Mas a literatura sobre engenharia de software, bem refletida na formação dos integrantes da GEPES, está ostensivamente pautada nas questões que denomina "técnicas", restando apenas alertas enfáticos sobre a "importância" de questões como patrocínio, motivação, liderança, dentre outras, como antídoto para as dificuldades semelhantes às narradas nesta história. Ou seja, uma postura que corrobora e reafirma o modelo de difusão, advogando que o "sucesso" dos projetos seja atribuído à excelência intrínseca das soluções "técnicas universais" e que o "fracasso" possa sempre ser imputado ao "não-técnico", aos "fatores" supostamente "externos" à engenharia de software.

5.3 Algo comum às duas *pequenas histórias*

Tomemos as duas histórias apresentadas para discutir os "fatores não-técnicos", cuja importância é tão citada na literatura de engenharia de software. Serão considerados exemplos de tais "fatores" o *patrocínio* da alta administração e o *envolvimento* daqueles que serão "afetados" pelo projeto. A engenharia de software usualmente difusionista trata ambos de forma arquetípica, tomando-os por essencialidades, por entidades capazes de existir de maneira apriorística, independente dos projetos. Deste modo, um projeto que malogrou pode ter como causa explicativa a "inexistência do patrocínio" e/ou a "inexistência do envolvimento". Caberia a pergunta, contudo: é possível, sem ambivalência, definir que houve ou que não houve "patrocínio" e "envolvimento" nas duas histórias apresentadas? A resposta é negativa.

Dizer que é difícil definir nitidamente patrocínio e envolvimento como essencialidades equivale a dizer que é impossível, nas histórias, definir nitidamente que houve, ou que não houve, "patrocínio" e "envolvimento". É mais fácil reconhecer que houve e que não houve, conquanto não devemos tomá-los por algo essencial, mas sim, pela atuação dos atores que os materializaram na rede sociotécnica em questão. Diferente de uma essência fixa e imutável, importa a atuação do "envolvimento" e do "patrocínio" à medida que viabilizam novas alianças e *traduções*.

O envolvimento foi materializado, por exemplo, nos treinamentos sobre análise de sistemas da primeira história e nas entrevistas e palestras da segunda história. Foi

também materializado na definição *colegiada* e *participativa* que culminou com a elaboração da Política de Informática (DESIS, 1989) da primeira história. O próprio PDI–2003, um esforço de planejamento *participativo*, da segunda história, estabeleceu o IPS como um projeto prioritário. O que dizer também do colegiado, referido sob a denominação *grupo homologador*, que homologou os subprocessos de planejamento, monitoração e controle de projetos? Por outro lado, também é fácil tirar das histórias exemplos de não envolvimento. Os novos programadores que surgiram com o uso do microcomputador nas GESIS, em "guerra" com o DESIS, não se envolveram a ponto de usar a MEDES, vista na primeira história como tentativa de controle. Na segunda história, deixou de haver também o envolvimento dos desenvolvedores à medida que o tempo foi passando e a GEPES não conseguia materializar o processo de software.

Analogamente, o mesmo pode ser feito com o patrocínio. Em ambas as histórias ele se materializou no orçamento necessário aos projetos, em regulamentações internas, no estabelecimento de políticas e processos, na criação de comitês e grupos. No entanto, muitas vezes pareceu não existir em uma organização tão sujeita a mudanças da alta administração. Ademais, uma explicação difusionista para as dificuldades do projeto de implantação da análise estruturada e de implantação do processo de software tenderia a se respaldar em alegações sobre a inexistência de patrocínio, um lugar-comum. Essa ambivalência — existiu ou não existiu —, bem como uma sensibilização para a necessidade de patrocínio para o projeto, podem ser inferidas na visão de Sérgio Viveiros.

> Patrocínio na prática, não no discurso, [...] é o *interesse* do patrocinador pelo projeto, seu entendimento sobre o que é o projeto, seu *alinhamento* com os objetivos do projeto, internalização pelo patrocinador do caminho que será trilhado. [O patrocinador] tem que ter o mínimo entendimento do problema e um convencimento [...] de que aquilo resolve [...], achar *que a solução para o problema passa pelo que será feito*. [...] Não sei se o que tivemos [no IPS] se chama patrocínio. Pode ser patrocínio mas falta [...] um outro pedaço. Nunca tive reunião com nenhum superintendente de acompanhamento do projeto. Sempre tivemos reuniões de apresentação, quando o superintendente entrava, ou quando tinha algum problema relacionado com a licitação da consultoria. Mas acompanhamento do projeto, o superintendente demonstrar interesse e valorizar aquele projeto como importante? Nenhuma. (Entrevista concedida ao autor em 10.08.2006)

Foi explorado, à luz dos *Estudos CTS*, que a organização, a padronização e a produtividade que existiram na informática do BNDES dos anos 1970 foram efeitos resultantes da estabilização de uma rede sociotécnica. Além disso, também foi explorado que, quando se examina uma rede sociotécnica, importam mais as atuações que os diversos atores perfazem do que a essência que supostamente os constitui. A estabilização da rede depende dessas atuações que necessariamente mudam à medida que os atores se inter-relacionam. Devido a essas questões, fica enfraquecida a perspectiva difusionista de considerar envolvimento e patrocínio

entidades essenciais, arquetípicas, capazes de explicar o fracasso dos projetos por não lhes preexistir. Sob o enfoque sociotécnico, mais importante que considerar a suposta essência de envolvimento e patrocínio, seria considerar a atuação dos atores que eles materializam na rede, ou até mesmo considerar a atuação do envolvimento e do patrocínio eles próprios tomados como atores. Caso essa atuação garanta as performances esperadas dos diversos outros atores envolvidos, a conseqüente estabilização da rede sociotécnica produziria efeitos que permitiriam dizer que houve patrocínio e/ou houve envolvimento.

Seria difícil imputar à falta de patrocínio o malogro do projeto de implantação da análise estruturada. Embora não tenha sido relatado, a alta administração deliberou formalmente, em pelo menos dois momentos, a execução do planejamento realizado pela *Andersen Consulting*. O primeiro momento foi em 1991, quando a diretoria autorizou a execução do plano[94]. No segundo momento, já em 1993, o Diretor VP (Vice-Presidente) ratificou a intenção de se dar continuidade à execução do plano[95]. Ora, esses dois instrumentos formais da alta administração seriam a própria encarnação do patrocínio. Essa interpretação, no entanto, estaria em tensão com a observação de que na prática ambos os instrumentos foram pouco efetivos, pois muito do que foi planejado não foi realizado. Houve "*a ênfase quase que exclusiva ao hardware*" (FARIAS, 1994, p.68), ou seja, conseguiu-se realizar mais facilmente apenas os tópicos do planejamento que diziam respeito à infra-estrutura física.

De qualquer modo, uma narrativa difusionista que dissesse que o esforço de implantação da MEDES fracassou, teria dificuldades de estabelecer nitidamente que faltou patrocínio, provavelmente tendo que se justificar em outros "fatores não-técnicos". Mas, com a perspectiva sociotécnica, seria trivial a interpretação de que os dois instrumentos (aprovação da diretoria para a execução do plano em 1991 e a nota do Diretor VP em 1993), embora formais, foram incapazes de gerar *traduções* que levassem os demais atores da rede a atuarem como esperado, resultando um ordenamento que permitisse a interpretação de que a MEDES fora implantada.

Sob o enfoque sociotécnico, caberia a interpretação de que a estabilização da rede não favoreceu a *pontualização* (seção 4.2) da MEDES, tanto quanto ocorreu com o MIT (Manual de Instruções Técnicas) até meados dos anos 1980. O mesmo pode ser argumentado para o processo de software. Ambos, MEDES e processo de software, foram incapazes de canalizar para si, como fizera o MIT, o movimento de

---

[94] IP AA/DESIS 005/91 — Instrução padronizada com a proposição, aprovada pela diretoria, de execução do plano desenvolvido pela *Andersen Consulting*.

[95] NOTA Diretor VP 001/93 que ratificou na reunião ordinária da diretoria, em 25/01/1993, a deliberação de se executar o planejamento realizado.

acumular as experiências locais e de moldar a realização das atividades de desenvolvimento e manutenção de software. As *inscrições* que geraram não impuseram programas de ação fundamentais para a solução dos problemas do desenvolvimento de software, e, portanto, não se tornaram *pontos de passagem obrigatórios*.

Nos anos 1990, por exemplo, o problema que reclamava equacionamento era a explosiva demanda por sistemas e serviços de informática pelas diversas áreas de negócio. Para dar conta desse desafio, a rede sociotécnica se estabilizou com a atuação da microinformática e das gerências de desenvolvimento distribuídas, as GESIS. Nessa rede, a MEDES definitivamente era um ponto de passagem obrigatório. Muito pelo contrário, vista como tentativa do DESIS de manter seu controle e burocracia, de fato, a MEDES foi um elo fraco que acabou deixando de existir na rede.

> Acho que a necessidade é que faz realmente a metodologia existir. *Porque enquanto ela não for necessária, você consegue resolver o problema sem metodologia*. [...] A falta de metodologia de trabalho é uma questão do banco [não só da informática]. [...] De alguma forma a informática responde à demanda real que o banco tem [mesmo sem metodologia ou processo formal]. (José Gonçalves, em entrevista concedida ao autor em 02.08.2006)

Além do mais, a perspectiva sociotécnica, quando esclarece que contexto e conteúdo são indissociáveis, instrumentaliza a percepção de que ao implantar um "conteúdo técnico" implanta-se também seu "contexto" de produção. Conteúdo e contexto são reconstruídos localmente nos esforços de implantação de modelos "universais", de sorte que, no caso, implantar a análise estruturada ou o CMMI é tentar implantar também o contexto, por exemplo, de medições, levantamentos, estatística e de formalização norte-americanos. Para termos uma dimensão disso, poderíamos refletir sobre como brasileiros e norte-americanos reagem à formalização. Nos EUA seria admissível uma pergunta, por vezes usual entre nós brasileiros, como: "*será que a lei 'tal' vai pegar*"? Logo, mesmo se fosse o caso de concordar com a justificativa dos "fatores não-técnicos" para o malogro de um projeto, eles próprios teriam que ser reconstruídos localmente também. Patrocínio para um norte-americano não é o mesmo que para um brasileiro. Buscar/obter patrocínio em uma corporação norte-americana não é o mesmo que buscar/obter patrocínio numa estatal brasileira.

Tomar o esforço subjacente às duas histórias como um desafio aparentemente de implantar um "conteúdo técnico" no "contexto adequado" é o que leva à sensação de ruptura entre "teoria" e "prática". Diferente dos anos 1970, nos anos 1990 e 2000 a informática era menos passível de operacionalização e controle através de representações modernas. Foi visto que a aplicação de uma abordagem assim, com sua prevalente visão de mundo mecanicista, tenderá a ser bem sucedida somente se

puderem ser reproduzidas certas premissas, como, por exemplo, método e o pressuposto de ordem e estabilidade preexistentes (seção 3.4). Na rede sociotécnica da informática do BNDES nos anos 1970 foi possível reproduzir os efeitos e a validade dessas premissas. Aquele mundo parecia sujeitar-se bem à manipulação através de representações modernas, o que, sob uma visão retrospectiva, parece induzir à aceitação de que a "prática" e a "teoria" da engenharia de software estiveram mais próximas, curiosamente, em um caso no qual sequer mencionou-se a "teoria da engenharia de software", à época muito incipiente.

Entretanto, o mundo da primeira e o da segunda história eram bem diferentes daquele dos anos 1970. A complexidade — operacional, organizacional e do próprio negócio do BNDES — tornou-se muito maior. O controle central do CPD deixou de existir e surgiu a operação descentralizada da informática com GESIS e microcomputadores. A especialização analista/programador terminou, dando lugar a novos analistas-programadores que não podiam mais conceber o sistema isolados dos usuários. A situação de "terra virgem" e sistemas com interface em papel deixou de existir. A racionalidade do analista que podia impor a "melhor solução técnica" que projetara não era mais aceita na rede, quando até mesmo o "usuário" passou a ser o "analista" e o "desenvolvedor" do sistema que precisava. Os pontos de controle que garantiam a consecução das atividades, antes muito estendidos ao longo das tarefas de todos, deram lugar às respostas *online* dos microcomputadores. A estabilidade dos processos de negócio e a independência inicial entre as atividades automatizadas e os sistemas desapareceram, transparecendo a situação de interdependência entre sistemas e processos de negócio cada vez mais mutáveis.

Narrativas lineares não dariam conta de traduzir a complexa dinâmica existente na prática dos projetos da primeira e da segunda história. Cada vez mais, teriam que recorrer às explicações dos "fatores não-técnicos", supostamente além do enquadramento da engenharia de software. As duas histórias são rupturas com esse tipo de narrativa, pois provocam no sentido de que implantar um processo de software ou a análise estruturada no BNDES é agir sobretudo no que estaria "fora" da engenharia de software, pouco reportando-se a esses modelos "universais". São rupturas porque trouxeram mais os "contaminantes sociais", o "ruído" — um conjunto de coisas que não podem ser bem alinhavadas nas narrativas lineares (LAW, 1999) — do que o "sinal" — no caso a análise estruturada e o CMMI —.

> Na era da alta modernidade e da globalização, no entanto, mais conhecimento pode ser a mesma coisa que levar a *mais imprevisibilidade*, *mais incerteza* e *menos controle*. [Isso] contradiz o otimismo [da racionalidade] instrumental com respeito à predeterminação da controlabilidade de coisas incontroláveis [...]. Modernização significa integração. Ao mesmo tempo, toda mudança — novas tecnologias

> introduzidas, implementação de estruturas organizacionais e procedimentos de trabalho, etc. — traz efeitos colaterais [o que produz o "ruído"] não previstos. Qualquer mudança pode afetar aqueles que interagem com os processos que estão envolvidos na mudança. [...] Nós estamos vivendo a era dos efeitos colaterais... O efeito colateral, [ou seja, o "ruído"] não a racionalidade instrumental [o "sinal"], é o motor das mudanças [...].[lvii] (HANSETH; BRAA, 2001, p. 48, grifos do autor).

As duas histórias buscaram trazer à tona a situação paradoxal de uma engenharia de software que se deseja "puramente técnica", pois, em algumas situações, o seu "fora" tem impedido sua "existência" na prática. Na verdade a postura tecnocêntrica e difusionista, não só da engenharia de software, mas de toda a tecnociência moderna, abriga-se na conveniência das divisões nítidas entre "questões técnicas" e "questões não-técnicas", ou "sociais", um discurso que possibilita a manutenção da falácia das escolhas puramente técnicas, de uma visão neutra e determinista da tecnologia, do suposto divórcio entre as questões tecnológicas e éticas e da visão dos artefatos de software apenas como objetos de engenharia. Coyne, embora não pontuando especificamente a engenharia de software, mas sim a tecnologia de informação como um todo, corrobora essas afirmativas.

> Como a tecnologia da informação marginaliza a ética? Este divórcio é mais evidente na distinção comum entre questões técnicas e questões relacionadas aos fatores humanos. [...] Muito longe de admitir uma tensão dinâmica, dialética, indeterminada, a oposição entre fatores técnicos e humanos está claramente entrincheirada e institucionalizada no modo de organização e prática da indústria da tecnologia da informação. Por exemplo, cursos na maioria das universidades distinguem a tecnologia e a sua aplicação, bem como sua avaliação crítica. A primeira pertence à engenharia, os últimos à área de humanidades. [...] As disciplinas de engenharia e de tecnologia são vistas como contribuintes claras e genuínas para a competitividade internacional e o bem estar social. [...] Atrelada a essa bifurcação entre o técnico e o ético está a premissa comum acerca da neutralidade da tecnologia. Tendo estabelecido a distinção entre técnica e valor, a racionalidade tecnológica agora nos causa a impressão de que os objetos não são nem bons nem ruins, mas o uso humano que os faz uma coisa ou outra.[lviii] (COYNE, 1995, p. 76).

Capítulo 6
Conclusão

Nesta dissertação, buscou-se compreender a engenharia de software como uma disciplina historicamente construída. Como qualquer outra disciplina, seria impossível que ela estivesse isenta de suas condições de produção. Tratou-se de um esforço de "desnaturalizar" a idéia de uma engenharia de software pré-discursiva, algo que já existisse antes mesmo de ser construída, e cuja evolução se processaria através de descobertas/invenções sucessivas no curso de um progresso inevitável. Foi visto que, assim como a própria tecnologia da computação, a engenharia de software surgiu sob o forte impulso decorrente do empreendimento norte americano da Guerra Fria.

Diferente da idéia de progresso inevitável da ciência e tecnologia, que evoluiria com a sucessão de descobertas e inventos "objetivos" e "tecnicamente" otimizados, realizados por cientistas e engenheiros geniais, tratou-se de enquadrar a engenharia de software como uma construção sociotécnica. Como uma rede que foi estabelecendo-se em um processo de resolução de controvérsias, um campo de *traduções*, no qual o resultado não pode ser determinado *a priori*, a exemplo do ocorrido no caso do *go to* (seção 2.3.1), cuja abolição não pode ser atribuída à uma inspiração genial, ou à inventividade "técnica" de Dijkstra. Foram *traduções* que imbricaram, indissociavelmente, tecnologia, ciência e a sociedade ao mesmo tempo que se foi resolvendo a controvérsia acerca da impropriedade do uso do *go to*.

Aquilo que depreendemos por técnicas, tecnologias, modelos, ferramentas, teorias e padrões da engenharia de software são produtos e efeitos resultantes de sua rede sociotécnica. Instituir-se como disciplina, o que só pode ocorrer quando se estabiliza como rede sociotécnica, equivale a associar-se a diversos atores, a participar em outras redes. Em seu processo de construção, ainda em curso, a engenharia de software buscou sua chancela científica, o que inevitavelmente vinculou-a à rede hegemônica da modernidade, e, portanto, ao estilo de pensamento dominante no desenvolvimento científico, tecnológico e social.

Antes mesmo da engenharia de software existir como disciplina autônoma, a performance da programação em larga escala surgiu associada a um discurso militarista beligerante de eminente praxe moderna que muito contribuiu para o viés tecnocêntrico da engenharia de software. Alinhada ao estilo de pensamento hegemônico, ela valoriza e busca teorias, modelos, processos, padrões e ferramentas "universais", supostamente dotados de qualidades intrínsecas que podem ser

replicadas nos locais onde sejam implantados. No entanto, existem inevitáveis rupturas e descontinuidades entre as teorias e modelos "universais" e a prática. A engenharia de software, na tentativa de explicar tais rupturas e descontinuidades, incorre na típica assimetria difusionista de imputar aos "fatores não-técnicos", a algo essencialmente "fora" de seu escopo, muitas dificuldades que surgem na prática.

Não se buscou nesta dissertação repisar a atualmente batida declaração de que os ditos "fatores não-técnicos", ou "fatores sociais", ou "fatores humanos", são determinantes nos projetos de software. O que se buscou foi a compreensão do porquê aquilo que é reconhecido como determinante para o bom êxito dos projetos é considerado "fora" da engenharia de software. Um nível inicial de compreensão pode ser delineado através da percepção de qual é o estilo de pensamento subjacente à construção da disciplina da engenharia de software. Algo que a esmagadora maioria dos engenheiros de software não consegue perceber e criticar, exatamente por conta de sua formação se dar segundo esse mesmo estilo de pensamento. Os engenheiros de software muitas vezes duvidam da validade de suas experiências práticas em favor da suposta validade inerente aos modelos e padrões "universais". Quando não os conseguem implantar e/ou aplicar, dificilmente aventam a possibilidade de inadequação dos próprios modelos e padrões.

O enfoque sociotécnico faculta uma percepção alternativa à difusão dos modelos e padrões. Pensar no engenheiro de software como um mero implantador desses modelos cristaliza a separação entre os países avançados e os periféricos, os primeiros, formadores de recursos humanos para concebê-los e, os segundos, formadores de recursos humanos para implantá-los / operá-los. Uma percepção alternativa é ter o engenheiro de software como um agente de (re)projetação local do modelo em sua realidade prática, em sua realidade situada. Seu trabalho deixa de ser o de implantar um "conteúdo técnico" no "contexto organizacional", para ser um trabalho de projetar as relações, papéis e atuações dos mais variados atores, humanos e não-humanos, incluindo como atores os próprios modelos e seus componentes envolvidos. Trata-se do trabalho de estabilizar uma rede sociotécnica, um trabalho de *tradução* que, quando bem sucedido, resulta em um ordenamento que permite a afirmação de que "o modelo foi implantado com sucesso".

Exemplos em que redes sociotécnicas locais são estabilizadas, somados ao jugo do estilo de pensamento hegemônico, induzem à interpretação de que o esforço do engenheiro de software pareça ser o de implantação de modelos e padrões "universais". Serve de exemplo o caso do BNDES nos anos 1970 que facilmente poderia ser explicado como bem sucedido através dessa aproximação difusionista. Já os outros dois casos do BNDES, nos anos 1990 e 2000, sob uma ótica difusionista,

seriam considerados malogros decorrentes da influência de "fatores não-técnicos". No entanto, a explicação exercitada com o enfoque sociotécnico foi a da atuação reciprocamente favorável dos diversos atores envolvidos, que culminou na estabilização da rede sociotécnica da informática do BNDES nos anos 1970, cujo efeito de ordem observado foi resultado que não pode ser atribuído a algum modelo ou padrão existente *a priori*. De forma simétrica, a explicação para o "insucesso" havido nos anos 1990 e 2000, também decorreu da atuação recíproca dos atores. Uma atuação que não favoreceu a estabilização da rede de modo que a ordem resultante permitisse a um "observador difusionista" julgar como bem sucedida a implantação da análise estruturada e do CMMI. Logo, diferente de retratar idealizadamente a experiência do BNDES nos anos 1970, cabe reconhecer que sucesso e fracasso também estão imbricados, variando de acordo com o enquadramento escolhido. Nos anos 1970, a estabilização da rede da informática do BNDES favorece a idéia de metodologia e processos formalizados terem existido. Não é adequado generalizar, extrapolando o enquadramento escolhido, e defender que naquele período houve maior sucesso do que nos anos 1990 e 2000. Apenas é possível dizer que a estabilização da rede nos anos 1990 e 2000 não favorece o reconhecimento de que metodologias e processos formalizados existiram nesses momentos.

A partir da problematização que aqui se estabeleceu podem ser vislumbradas algumas linhas para futuras investigações. O fundamental, todavia, é manter uma percepção crítica sobre o estilo de pensamento hegemônico que tutela a engenharia de software para permitir *linhas de fuga* (DELEUZE; GUATTARI, 1995), isto é, abordagens alternativas para as questões que não se dobram a soluções modernistas. Manter uma percepção crítica que viabilize abordagens que não sofram da miopia decorrente das grandes divisões, responsáveis pelo estabelecimento "teórico", *a priori*, de um "dentro" e de um "fora" da engenharia de software. Parafraseando Bruno Latour (2000), por onde poderíamos começar o estudo dos "fatores não-técnicos"? Pela engenharia de software já estabelecida, cujo enquadramento fixo delimita seu "dentro" e o seu "fora"? Ou seguindo engenheiros de software em ação, que lidam em sua prática indistintamente com diversos tipos de problemas desacompanhados de rótulos que os indiquem como sendo "técnicos" ou "não-técnicos"?

A segunda opção implica aceitar um limite empírico e variável entre o "dentro" e o "fora" da engenharia de software. Implica em deixar de formar os engenheiros de software exclusivamente sob o conforto das grandes narrativas simplificadoras para instrumentá-los, também, a mergulhar na complexidade situada e heterogênea da

prática que enfrentarão. Como complementar a formação dos engenheiros de software brasileiros? Como instrumentá-los para serem, mais do que difusores de modelos "universais", *tradutores* capazes de (re)projetar tais modelos na prática? Ou então de serem capazes de formular seus próprios modelos? Como torná-los aptos a reconhecer e lidar com realidades onde questões humanas, sociais, econômicas, culturais, políticas e ecológicas não são externas aos seus problemas, mas, ao contrário, compõem com ferramentas, métodos, modelos, teorias e técnicas um mesmo tecido sem costura?

A complexidade do mundo do engenheiro de software tem resistido a se deixar reduzir em partes constituintes mais simples e nitidamente identificáveis:

> [...] Com o Macintosh (e portanto, por extensão, o ubíquo estilo Macintosh de interface disseminado pela Microsoft) as pessoas têm saído da visão reducionista e mecanicista de como se relacionar com um computador [...]. Nos anos 1970 e início dos anos 1980, computadores traziam um ethos modernista: analise e você concordará; a partir de meados dos anos 1990, os complexos mundos simulados dos obscuros computadores ofereceram uma experiência que colocaram essas premissas em xeque. Culturalmente, o Macintosh trouxe a idéia de que é mais proveitoso explorar o mundo cujas superfícies são movediças do que embarcar numa busca por mecanismos, origens e estruturas. O modo de entendimento provido pelo Macintosh colocou-se em contraste com a perspectiva modernista que deu vida a textos de pensadores como Freud, Marx e Darwin, que sugeriram que o entendimento provinha da redução de coisas complexas a elementos mais simples, através da descoberta dos mecanismos ocultos por trás do comportamento.[lix] (TURKLE, 2005, p. 24).

Quais alternativas têm surgido para se atuar nessa complexidade? Tais alternativas[96], muitas gestadas com o enfoque mais amplo da disciplina de Sistemas de Informações, têm chegado à engenharia de software brasileira? Como têm sido ensinadas / aplicadas e quais resultados têm sido verificados? Particularizando para as instituições públicas brasileiras, tão sujeitas a descontinuidades (o que por si só já ensejaria estudos sobre seu funcionamento), como e quais alternativas podem ser utilizadas?

Respostas a perguntas desse tipo seriam favorecidas por um refinamento da agenda da engenharia de software brasileira, em seu desafio de participar da viabilização do Brasil como produtor de software relevante no mercado internacional. Uma agenda que continue valorizando tudo que se resolve bem com o enquadramento mecanicista e positivista, o que é legítimo, mas que seja sensível à dissonância cada vez maior desse enquadramento e que fomente outras

---

[96] Arnold (2003, p.228-230) traz um resumo de abordagens alternativas aos chamados *hard methods*. Destaca abordagens como *Computer Supported Cooperative Work* (CSCW), *soft systems design*, *partipatory design*, *user-centered design*, *end-user computing*, *democratic design*, *socio-technical design*. São abordagens consideradas marginais, posto que, diferente das abordagens positivistas consideradas centrais, têm respaldo filosófico em fontes outras, como fenomenologia, antropologia, etnometodologia, construtivismo social, determinismo social, perspectivas estruturalistas e teoria ator-rede.

possibilidades. Para isso é necessário um posicionamento alternativo ante o estilo de pensamento hegemônico no desenvolvimento científico e tecnológico ocidental, o que favoreceria um refinamento que melhor sintonizasse a agenda da engenharia de software brasileira com a transformação concomitante do mundo e da computação. Uma computação que deixou de lado a representação moderna, analítica, da centralização e hierarquização do CPD, da interface texto em favor da descentralização e distribuição, da simulação, das interfaces gráficas, *fac-simile* de nossos movimentos e ações. Uma computação que deixou de informatizar processos definidos analiticamente em favor da simulação de possibilidades e instâncias de processos de negócio.

As investigações sobre a prática histórica da engenharia de software brasileira também teriam grande valor. Elas permitiriam destacar empiricamente as complexas redes de relações técnicas, sociais, políticas e organizacionais que ajudariam a descortinar a complexidade de nosso presente, favorecendo sínteses mais adequadas às nossas necessidades locais. Como exemplos desse tipo de investigação, citamos dois dos nossos próprios experimentos: o primeiro é um artigo relacionado à empresa COBRA — Computadores Brasileiros S. A., intitulado "A Cobra Teve uma Partitura" (TEIXEIRA; CUKIERMAN, 2005b); o segundo, são os capítulos 4 e 5 desta dissertação que retrataram um pouco da informática do BNDES.

No primeiro experimento, quando se investigou o desenvolvimento de software ocorrido na COBRA no final dos anos 1970, constatou-se uma clara contraposição à idéia "naturalizada" de que um desenvolvimento de software, para ser bem sucedido, deve respaldar-se numa racionalidade moderna, numa racionalidade "puramente de engenharia". Naquela época, quando existiu no Brasil um *discurso* de *autonomia tecnológica nacional em informática*, a COBRA experimentou uma prática de desenvolvimento de software bem sucedida. Naquela experiência a estruturação do trabalho pôde se dar em torno de metáforas alternativas às modernas (representação, formalização, sistema, objeto, ordem, mecanismos, automatização). Trata-se de um exemplo local para nós, os engenheiros de software brasileiros, que delineia um enquadramento para o desenvolvimento de software baseado em valores como oralidade, individualidade, espírito, persuasão, o que resulta em interpretações diferentes da busca de controle hierárquico e centralizado dos projetos de software guiados por especificações. Um aprofundamento futuro deste experimento poderia

traçar sua proximidade com as atuais *metodologias ágeis*[97] de desenvolvimento de software.

Já o segundo experimento, como narrado no capítulo 4, também um caso dos anos 1970 e bem sucedido, mostra a informática do BNDES com uma típica inspiração moderna. No entanto, quando se prolonga a história considerando também o caminho trilhado pela informática do BNDES nas décadas subseqüentes (capítulo 5), chama a atenção a enorme coincidência existente com a própria história da engenharia de software norte-americana, como apresentada no artigo "*A View of 20th and 21st Century Software Engineering*", de Barry Boehm (2006). Todavia, ainda que coincidentes, as histórias ocorrem com vinte anos de defasagem. Vejamos.

Nos anos 1950 a engenharia do hardware é a inspiração primária para os processos de desenvolvimento de software. A palavra chave é o uso eficiente do caro e limitado hardware. Programadores são treinados na prática (*on the job*) em projetos como o SAGE. Tem-se uma ambiência com grande respaldo metodológico, tecnologia favorecendo a padronização e um forte controle hierárquico e centralizado dos projetos, guiados por especificações formalizadas, em um cenário de "terra virgem". Quadro semelhante o BNDES viveu nos anos 1970, quando a otimização do uso do hardware tinha peso preponderante, os profissionais eram formados nas empresas, a instalação de informática era centralizada no CPD, os processos eram formalizados e controlados hierarquicamente, num ambiente também de "terra virgem", posto que se iniciava a construção dos primeiros sistemas.

Com o aumento do poder do hardware, explode a demanda e a diversidade do software nos EUA dos anos 1960. A falta de pessoal habilitado enseja as linguagens de alto nível e muitos novos programadores puderam ingressar no mundo da computação. A percepção das diferenças inerentes aos processos de produção de hardware e de software leva, a partir de meados dos anos 1960, a uma prática de desenvolvimento de software insubordinada ao controle rígido do processo. Tem-se uma era do "*software crafting*", da popularização da programação por tentativa e erro (*code-and-fix*). Surge a figura dos programadores *cowboys*, os profundos conhecedores dos sistemas e os únicos capazes de, misteriosamente, nos momentos decisivos, reparar os defeitos e sanar as falhas. Como conseqüência, "código espaguete" em profusão começou a ser gerado e os elevados custos com retrabalho começaram a preocupar. Em resposta, nos anos 1970, são concebidos os métodos estruturados e o ciclo de desenvolvimento em cascata, numa tentativa de se depender menos das habilidades dos profissionais e garantir planejamento e controle mais

[97] Sobre metodologias ágeis vide: BOEHM, B.; TURNER, R., 2004, *Balancing Agility and Discipline: a*

efetivo dos projetos. E o que se viu no BNDES vinte anos depois, nos anos 1980 e 1990? Exatamente a explosão da demanda por software disseminada por todas as áreas de negócio e a falta de pessoal suficiente para atendê-la. Com a introdução da microinformática surgem novos analistas-programadores nas GESIS (as gerências de sistemas distribuídas pelas áreas de negócio). Devido à formação dos novos programadores e ao novo ambiente tecnológico, os novos sistemas passam a ser construídos à revelia de qualquer método e controle formalizados. Somente os "donos dos sistemas", aqueles que os construíram, sabiam tudo a seu respeito e eram os únicos capazes de prover manutenções neles tempestivamente, a exemplo dos "programadores *cowboys*". A falta de controle preocupa e nos anos 1990 buscou-se implantar a análise estruturada e maior formalização do esforço de gerenciamento dos projetos de desenvolvimento e manutenção dos sistemas.

Nos anos 1980, nos EUA, decorrente da não conformidade freqüente dos sistemas desenvolvidos com o ciclo cascata e devido à sobrecarga imposta pelos processos formalizados de desenvolvimento, surgem os padrões de qualidade e os modelos de maturidade, bem como a estruturação do trabalho em fábricas de software. Exatamente 20 anos depois, conforme narrado no capítulo 5, o BNDES nos anos 2000 deliberou o uso de fábricas de software e a implementação do CMMI.

Nestas últimas palavras, peço novamente licença para o uso da primeira pessoa e revelar a surpresa que me assaltou ao constatar a coincidência, com vinte anos de defasagem, entre a história da informática do BNDES e a história da engenharia de software norte-americana. Trata-se de uma questão instigadora para trabalhos futuros que resolvessem abordar por que somos consumidores, com décadas de atraso, e não produtores de muito daquilo que precisamos em nossa sociedade. Seríamos formados para difundir modelos e padrões "universais" para cumprir o ciclo capitalista necessário à amortização dos investimentos incorridos por seus produtores[98]? Novamente tomando o CMMI por exemplo, como um modelo que surgiu por conta da demanda de contratação de produtores de software pelo Departamento de Defesa dos EUA (DoD), algo claramente alinhado e coerente com suas condições de produção, torna-se um "em si universal"? Por que usualmente acreditamos que ao importar o

---

guide for the perplexed. Addison Wesley.

[98] Wagner do Carmo discute a NGN (*Next Generation Network*), uma tecnologia que busca a convergência dos meios de comunicação — dado e voz — em uma mesma plataforma. Um dos pontos de sua análise é que "*antes de tudo,* [é necessário] *estar atento às contingências locais e mostrar que nada é transferido sem se transformar. A rede NGN surge em um momento* ***propício para os países do Primeiro Mundo*** *e em um momento* ***ingrato para os do Terceiro Mundo****. Este lado ingrato representa uma rede não convergente que ainda não foi amortizada* [diferente da situação do Primeiro Mundo]" (do CARMO, W. R., 2005, *NGN – Uma análise sociotécnica da convergência das telecomunicações no Brasil.* Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, p.111, grifos nossos).

CMMI estamos importando apenas um "conteúdo técnico", e não importando conteúdo e contexto imbricados? O CMMI é "conteúdo de engenharia de software", mas também é, indissociavelmente, o "contexto norte-americano" de, por exemplo, medições, levantamentos, estatísticas e formalização de projetos. Implantar o CMMI em uma empresa teria de ser reprojetar, localmente, tal contexto e conteúdo.

Enfim, foi surpreendente perceber-me, retrospectivamente, tantas vezes difusionista e tutelado pelo estilo de pensamento dominante. Talvez agora, após o esforço de pesquisa que tornou possível a presente dissertação, esteja entendendo com mais propriedade a epígrafe utilizada: "*o mundo não é, o mundo está sendo*". Entendendo que também a ordem da engenharia de software não é dada (*o mundo não é*), mas que é construída (*o mundo está sendo*) e, assim sendo, que temos opções que não somente a de submeter-nos ao estilo hegemônico. Esse entendimento é fundamental para uma engenharia de software brasileira engajada e historicamente responsável em seu *contexto-conteúdo*.

Fontes e Referências Bibliográficas

1 – REFERÊNCIAS

ARNOLD, M., 2003, "Systems Design Meets Habermans, Foucault and Latour". In: CLARKE, S., et. al (eds.), *Socio-Technical and Human Cognition Elements of Information Systems*. London, Information Science Publishing.

BAKER, F.T., 1972, "Chief programming team management of production programming". In: IBM Systems Journal, v.11, n.01, p.56-73.

BAUM, C., 1981, *The System Builders: the history of SDC*. Santa Monica, SDC.

BENINGTON, H.D., 1956, Production of Large Computer Programs. In: Proceedings of The ONR Symposium on Advanced Program Methods for Digital Computers, jun., p.15-27. Também disponível em: Annals of the History of Computing, v. 5, n. 4, oct. 1983, p. 299-310.

BIO, S.R., 1985, *Sistemas de Informação*: *um enfoque gerencial*. São Paulo, Ed. Atlas.

BLOOR, D., 1991, *Knowledge and Social Imagery*. Chicago, University of Chicago Press.

BOEHM, B., 2006, "A View of 20th and 21st Century Software Engineering". In: 28th International Conference on Software Engineering (ICSE), Shangai, China, p.12-29.

BOOCH, G., 1991, *Object Oriented Design: with applications*. Benjamin/Cummings Publishing Company Inc.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., 1998, *The Unified Modeling Language User Guide*. Addison-Wesley Longman Inc.

BROOKS, F.P., 1986, *No Silver Bullet*: *essence and accidents of software engineering*. In: Proceedings of the IFIP Tenth World Computing Conference, p. 1069-1076.

BROOKS, F.P., 1995, *The Mythical Man-Month*: *essays on software engineering*. 20th Anniversary ed. Addison Wesley.

BUXTON, J.N.; RANDELL, B. (eds.), 1970, *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. Brussels, Scientific Affairs Division. Disponível em: <http:// //homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF>. Acesso em: 20 jul. 2007.

CALLON, M., 1999, "Some Elements of a Sociology of Translation: domestication of the scallops and the fishermen of St. Brieuc bay". In: BIAGIOLI, M., *The science studies reader*, New York, Routledge, p.67-83.

CALLON, M., 2001, "A Agonia de um Laboratório". Tradução de Ivan da Costa Marques. Reprodução livre em português, do texto original de Michel Callon para fins de estudo. <http://www.necso.ufrj.br/Trads/A%20agonia%20de%20um%20laboratorio.rtf>. Acesso em: 20 jul. 2007.

CAMPBELL-KELLY, M., 2004, *From Airline Reservations to Sonic the Hedghog: A History of the Software Industry*, MIT Press.

CARTER, L. et al., 2002, *The Road to CMMI: results of the first technology transition workshop*. Technical Report, CMU/SEI-2002-TR-007, Software Engineering Institute, Pittsburgh – PA.

CHRISSIS, M., et al., 2003, *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley.

COAD, P., YOURDON, E., 1990, *Object Oriented Analysis*. Englewood Cliffs, New Jersey: Yourdon Press.

COYNE, R., 1995, *Designing Information Technology in the Postmodern Age: from method to metaphor*. Cambridge, Massachusetts, MIT Press.

COX, B., 1990, "There Is a Silver Bullet", In: Byte Magazine, 15/10.

DAHL, O.-J., 2002, "The Roots of Object Orientation: the SIMULA language". In: BROY, M., DENERT, E. (eds), *Software Pioneers: contributions to software engineering*, Berlin, Germany, Springer-Verlag, p.79-90.

DAHLBOM, B.; MATHIASSEN, L., 1993, *Computer in Context: The Philosophy and Practice of Systems Design*. Cambridge, NCC Blackwell Publishers.

DAGNINO, R.; NOVAES, H.T., 2006, O papel do engenheiro na sociedade. In: III Encontro Nacional de Engenharia e Desenvolvimento Social — ENEDS, Rio de Janeiro.

DELEUZE, G.; GUATTARI, F., 1995, *Mil Platôs: Capitalismo e Esquizofrenia*. Vol. 01, Editora 34.

DeMARCO, T., LISTER, T., 1999, *Peopleware: productive projects and teams*. 2nd ed., New York, Dorset House Publishing.

DeMARCO, T., 2002, "Structured Analysis: Beginnings of a New Discipline". In: BROY, M., DENERT, E. (eds), *Software Pioneers: contributions to software engineering*, Berlin, Germany, Springer-Verlag, p.521-527.

DIJKSTRA, E.W., 1968, "Go To Statement Considered Harmful". In: Communications of the ACM, v. 11, n. 2, p.147-148.

_____, 2002, "From 'Goto Considered Harmful' to Structured Programming". In: BROY, M., DENERT, E. (eds), *Software Pioneers: contributions to software engineering*, Berlin, Germany, Springer-Verlag, p.340-346.

EDWARDS, P.N., 1997, *The Closed World: computers and the politics of discourse in Cold War America*. Massachusetts, MIT Press.

ENDRES, A., 2002, "Commentary on James E. Tomayko 'Software as Engineering'". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.77-81.

ENSMENGER, N.; ASPRAY, W., 2002, "Software as Labor Process". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.139-165.

FAROOQ, O., 2006, "India Firms Warn on IT Skills Gap", In: BBC News, 07 nov. 2006. Disponível em: <http://newsvote.bbc.co.uk/go/pr/fr/-/2/hi/south_asia/6124872.stm>. Acesso em: 09 nov. 2006.

FLECK, L., 1986, *La Génesis y el Desarrollo de un Hecho Científico*. Madrid, Alianza.

FUGGETTA, A., 2000, "Software Process: A Roadmap". In: FINKELSTEIN, A. (ed.), *The Future of Software Engineering*.

HAIGH, T., 2002, "Discussion following 'Software as Science". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.61-62.

HANSETH, O.; MONTEIRO, E., 1998, *Understanding Information Infrastructure*. Manuscript. Disponível em <http://heim.ifi.uio.no/~oleha/Publications/book.pdf>. Acesso em: 01 abr. 2005.

HANSETH, O.; BRAA, K., 2001, "Globalization and 'Risk Society'". In: CIBORRA, C., et al., *From Control to Drift: The Dynamics of Corporate Information Infrastructures*, Oxford, Oxford Univ. Press, p.41-55.

HARVEY, D., 2006, *Condição Pós-Moderna: Uma Pesquisa sobre as Origens da Mudança Cultural*. 15ª ed., São Paulo, Edições Loyola.

HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds.), 2002, *History of Computing: software issues*. Berlin, Germany, Springer-Verlag.

HATTON, L., 1998, "Does OO Sync with How We Think?". In: IEEE Software, may.

HERBSLEB, J.D., 2005, "Beyond Computer Science". In: Proceedings of 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, EUA, p. 23-27.

HIRSCHHEIM, R.; HEINZ, K.K.; LYYTINEN, K., 1995, *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge University Press.

HOHN, F., 1955, "The First Conference on Training Personnel for the Computing Machine Field". In: The American Mathematical Monthly, v.62, n.1, p.8-15.

HOUNSHELL, D.A., 2002, "Are Programmer Oppressed by Monopoly Capital, or Shall the Geeks Inherit the Earth? Commentary on Nathan Ensmenger & Willian Aspray, 'Software as Labor Process'". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.167-175.

HUMPHREY, W.S., 2000, "The Personal Software Process: status and trends". In: IEEE Software, nov/dec, p.71-75.

JACKSON, M.A., 1988, *Princípios em Projetos de Programas*. Rio de Janeiro, Editora Campus.

LATOUR, B., 2000, *Ciência em Ação: como seguir cientistas e engenheiros sociedade afora*. São Paulo, Editora UNESP.

_____, 2001, *A Esperança de Pandora: ensaios sobre a realidade dos estudos científicos*. Bauru, SP, Editora EDUSC.

LAW, J., 1992, "Notes on the Theory of the Actor-Network: Ordering, Strategy, and Heterogeneity", Systems Practice, v.5, n.4, p.379-393.

_____, 1999, "Traduction/Trahison: Notes on ANT", Disponível em: <http://www.lancs.ac.uk/fss/sociology/papers/law-traduction-trahison.pdf >. Acesso em: 14 nov. 2006.

LEVESON, N.G., 1997, "Software Engineering: Stretching the Limits of Complexity". In: Communications of the ACM, v. 40, n. 2, p.129-131.

MAHONEY, M.S., 2002, "Software as Science — Science as Software". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.25-48.

_____, 2004, "Find a History for Software Engineering". In: IEEE Annals of the History of Computing, jan/march, p.8-19.

MARTIN, J., 1993, *Principles of Object-Oriented Analysis and Design*. Englewood Cliffs, New Jersey, PTR Prentice Hall.

McFEELEY, B., 1996, *IDEAL: A User's Guide for Software Process Improvement*. Handbook, CMU/SEI/96-HB-001, Software Engineering Institute, Pittsburgh — PA.

MÉSZÁROS, I., 2004, *O poder da ideologia*. São Paulo, Boitempo editorial.

NAUR, P.; RANDELL, B. (eds.), 1969, *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmish, Germany, 7th to 11th October 1968*. Brussels, Scientific Affairs Division. Disponível em: <http:// homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>. Acesso em: 20 jul. 2007.

PARNAS, D. L., 1997, "Software Engineering: An Unconsummated Marriage". In: Communications of the ACM, v. 40, n. 9, p.128.

PFLEEGER, S.L., 2001, *Software Engineering: theory and practice*. 2nd ed., Upper Saddle River, New Jersey, PTR Prentice Hall.

PLOOY, N.F. du, 2003, "The Social Responsibility of Information Systems Developers". In: CLARKE, S., et. al (eds.), *Socio-Technical and Human Cognition Elements of Information Systems*. London, Information Science Publishing, p.41-59.

PRESSMAN, R.S., 2001, *Software engineering: a practitioner's approach*. 5th ed., McGraw-Hill.

QUIVY, R., VAN CAMPENHOUDT, L., 1998, *Manual de metodologia em Ciências Sociais*. 2ªed. Lisboa, Gradiva.

ROYCE, W.W., 1970, "Managing the Development of Large Software Systems". In: Proceedings of IEEE WESCON, aug., p.1-9.

RUBIN, F., 1987, "'GOTO considered harmful' considered harmful", In: Communications of the ACM, v. 30, n.3, p.195-196.

RUMBAUGH, J., et al., 1994, *Modelagem e Projeto Baseados em Objetos*. Rio de Janeiro, Editora Campus.

SCACCHI, W., 2004, "Socio-Technical Design". In BAINBRIDGE, W.S. (ed.), *The Encyclopedia of Human-Computer Interaction*, Berkshire Publishing Group.

SEELY, B., 2002, "Commentary on James E. Tomayko, 'Software as Engineering'". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.83-91.

SOMMERVILLE, I., 2004, *Software engineering*. 7th ed., Addson-Wesley.

SUCHMAN, L., 1987, *Plans and Situated Actions: the problem of human machine communication*. Cambridge University Press.

TEIXEIRA, C.A.N.; CUKIERMAN, H., 2005a, Apontamentos para Enriquecer o Perfil do Engenheiro de Software. In Congresso da Sociedade Brasileira de Computação, 25., 2005, São Leopoldo/RS. Anais Eletrônicos... São Leopoldo/RS: Unisinos. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0026.pdf >. Acesso em: 08 jan. 2007.

TEIXEIRA, C.A.N.; CUKIERMAN, H., 2005b, A Cobra Teve uma Partitura. In Workshop um Olhar Sociotécnico sobre a Engenharia de Software, 1., 2005, Rio de Janeiro/RJ. Disponível em: <http://www.cos.ufrj.br /~handrade/woses/woses-2005/anais.htm>. Acesso em: 27 jul. 2007.

TEIXEIRA, C.A.N., 2006, Algumas observações sobre os vínculos entre a Engenharia de Software e o pensamento moderno. In Workshop um Olhar Sociotécnico sobre a Engenharia de Software, 2., 2006, Vila Velha/ES. Disponível em: <http://www.cos.ufrj.br /~handrade/woses/woses2006/anais.htm>. Acesso em: 17 abr. 2007.

TEIXEIRA, C.A.N.; CUKIERMAN, H., 2007, Por que Falham os Projetos de Implantação de Processos de Software?. In Workshop um Olhar Sociotécnico sobre a Engenharia de Software, 3., 2007, Porto de Galinhas/PE. Disponível em: <http://www.cos.ufrj.br /~handrade/woses/woses2007/anais.htm>. Acesso em: 01 jul. 2007.

TOMAYKO, J.E., 2002, "Software as Engineering". In: HASHAGEN, U., KEIL-SLAWIK, R., NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.65-76.

TOMAYKO, J.E.; HAZZAN, O., 2004, *Human Aspects of Software engineering*. Hingham, Charles River Media, Inc.

TURKLE, S., 2005, *The Second Self: computer and the human spirit*. 20th Anniversary ed. MIT Press.

YOURDON, E., 1992, *Análise Estruturada Moderna*. Rio de Janeiro, Ed. Campus.

2 – FONTES MANUSCRITAS E ORAIS

Entrevistas com:

Almeida, Cláudio Cezar Carvalho, A informática do BNDES. Rio de Janeiro, 26/06/2006.

Calheiros, Felipe Carneiro, A informática do BNDES. Rio de Janeiro, 07/08/2006.

Calvano, João Alberto, A informática do BNDES. Rio de Janeiro, 21/06/2006.

Carvalho, Vânia Cerqueira de, A informática do BNDES. Rio de Janeiro, 10/07/2006.

Cohen, Melvyn Afonso, A informática do BNDES. Rio de Janeiro, 20/06/2006.

Esteves Filho, Mário José Soares, A informática do BNDES. Rio de Janeiro, 22/08/2006.

Fischler, Alan Adolfo, A informática do BNDES. Rio de Janeiro, 11/07/2006.

Fonseca, Oswaldo Luiz Humbert, A informática do BNDES. Rio de Janeiro, 21/06/2006.

Franco, Sidney, A informática do BNDES. Rio de Janeiro, 01/08/2006.

Frota, Osmar Nelson, A informática do BNDES. Rio de Janeiro, 31/07/2006.

Gonçalves, José Marcos Silveira, A informática do BNDES. Rio de Janeiro, 02/08/2006.

Gouvea, Renato Luiz Proença de, A informática do BNDES. Rio de Janeiro, 26/06/2006.

Oliveira, Ângela Curvello Araújo d', A informática do BNDES. Rio de Janeiro, 07/07/2006.

Matsushima, Ricardo Massao, A informática do BNDES. Rio de Janeiro, 14/07/2006.

Nardin, Marcelo, A informática do BNDES. Rio de Janeiro, 10/07/2006.

Viveiros, Sérgio Marques de, A informática do BNDES. Rio de Janeiro, 10/08/2006.

3 – FONTES IMPRESSAS

3.1 – Periódicos

AFBNDE INFORMA. Rio de Janeiro: Associação dos Funcionários do Banco Nacional do Desenvolvimento Econômico, ano I, no. 2, jun. 1975.

3.2 – Relatórios

ANDERSEN CONSULTING Arthur Andersen, Plano Estratégico de Informações: Relatório Final, Banco Nacional do Desenvolvimento Econômico, Rio de Janeiro, Brasil, outubro de 1991.

BOOZ, ALLEN and Hamilton International Inc. Management Consultants, Relatório sobre o Projeto de Melhoria de Organização e Gerência, Banco Nacional do Desenvolvimento Econômico, Rio de Janeiro (GB), Brasil, dezembro de 1972. 488 páginas, XXXVI anexos.

DESIS, 1989, Relatório de atividades, Departamento de Sistemas, Banco Nacional de Desenvolvimento Econômico e Social.

GEPES, 2003a, Relatório de análise das entrevistas realizadas com os analistas e coordenadores do AAI/DESIS, Departamento de Integração de Processos, Banco Nacional de Desenvolvimento Econômico e Social.

3.3 – Outros Documentos

FARIAS, R., 1994, Resumo do Plano Estratégico de Informações da Andersen Consulting – 1991, Auditoria Interna, Banco Nacional de Desenvolvimento Econômico e Social.

GEPES, 2003b, NOTA TÉCNICA AAI/DEINP Nº 03/2003, Departamento de Integração de Processos, Banco Nacional de Desenvolvimento Econômico e Social.

GEPES, 2004, Plano de Ação: projeto de implantação de processo de software (IPS–BNDES), Departamento de Integração de Processos, Banco Nacional de Desenvolvimento Econômico e Social.

INSTRUÇÃO PADRONIZADA (IP) AA/DESIS 003/90.

INSTRUÇÃO PADRONIZADA (IP) AA/DESIS 005/91.

INSTRUÇÃO DE SERVIÇO (IS) ACO/SUP Nº 02/2005.

MEDES, 1990, Metodologia de Desenvolvimento de Sistemas do BNDES, Departamento de Sistemas, Banco Nacional de Desenvolvimento Econômico e Social.

MEDES, 1993, Metodologia de Desenvolvimento de Sistemas do BNDES (revisão), Departamento de Sistemas, Banco Nacional de Desenvolvimento Econômico e Social.

MIT, 198?, Manual de Instruções Técnicas, Departamento de Sistemas, Banco Nacional de Desenvolvimento Econômico e Social.

NOTA do Diretor VP 001/93.

PDI-1985, AP/DESIS, março 1985, Plano Diretor de Informática, Banco Nacional de Desenvolvimento Econômico e Social. 140 microfichas.

PDI-2003, AAI, janeiro 2004, Plano Diretor de Informática 2004–2006, Banco Nacional de Desenvolvimento Econômico e Social.

Resoluções do Conselho de Administração do BNDES, Nos. 245/66; 283/67; 402/71.

Resoluções da Diretoria do BNDES, Nos. 425/72; 475/75.

ANEXO I
Notas de Tradução

[i] The high rate of so-called information systems "failures" (e.g. systems abandoned before completion, completed but never used, very large budged and time overruns) has been a concern since the earliest days of computer-based information systems. It's generally agreed that most, if not all, of these failures could be attributed not to a lack of tools and techniques, but to the neglect of "human factors" in the dominant systems analysis and design practices. (Página 1)

[ii] [...] constitutes a prime example of how modern technoscience confounds traditional categories of theory and practice. (Página 5)

[iii] Do not lump all software together and threat it as a single phenomenon. Software for a space flight, a text editor or a student's first trials in Java are quite different things. To do so, would be like treating an aircraft maintenance manual, a best selling novel and graffiti in the subway as comparable works in writing. Too often, people who have written one type of program declare themselves experts on all types. You not accept that for any other types of engineering or literary work. Nor does it make the teaching of software engineering any easier. (Página 11)

[iv] [...] the RAND Corporation, [...] where the natural scientists, social scientists, and mathematicians worked syde by side to anticipate and prepare the future of the war, [...] used systems analysis techniques, born from wartime operations research, to investigate both the mundane problems of weapons procurements and the unknown realm of nuclear strategy. These techniques both benefited from and helped promote RAND's extensive work on computers; RAND's computation center, among the largest in the world, significantly affected the nature and direction of software development in the 1950s. In addition, by legitimating systems analysis, computers helped advance Rand's theory-and simulation-based approach to strategy. (Página 13)

[v] A commonly used figure for the total development is $8 billion, including construction of twenty-six SAGE sites, purchase of fifty-six computers at $ 30 million apiece, installation of twenty-five thousand telephone lines, plus the associated equipment and services. SDC's programming for eight versions of SAGE would amount to $ 150 million, about 2 percent of the total cost of the system. (Página 14)

[vi] "RAND is far too busy to take on the problems of industry. Besides, its charter forbids it to do so. But it does offer this hopeful thought. Men trained in System Development Division become a source of supply for directors of future automation projects..." These words appearing in Business Week some eighteen months before SDC started operations, accurately prophesied the role the company was to play in training thousands of programmers, system analysts, and system training experts who would shortly occupy the ground floor of the emerging data processing industry. Questioned on this topic, veteran SDCers are quick to recal: "We trained this country's first three thousands programmers". "There were no systems programmers before SDC". "Whatever company I visit, I meet two or three alumni". "We trained the industry!". (Página 16)

[vii] Competition for programmers has driven the salaries up so fast that programming has become probably the country's highest paid technological occupation. ... Even so, some companies can't find experienced programmers at any price. (Página 21)

[viii] Their experience is that graduates in our programs seem to be mainly interested in playing games, making fancy programs that really do not work, writing trick programs, etc., and are unable to discipline their own efforts so that what they say will do gets done o time and in practical form. (Página 22)

[ix] By the middle of the 1960s, a perceptible shift in the relative costs of hardware and software had occurred. The falling cost of hardware allowed computers to be used for more and larger applications, which in turn required larger and more complex software. As the scale of software projects expanded, they became increasingly difficult to supervise and control. The pressing problems for software developers were now more managerial than technical. [...] In a presentation to the Fall Eastern Joint Computer

Conference in 1965, J. Presper Eckert argued that programming would become manageable only when it could be referred to as "software engineering". (Página 23)

[x] It is easy for me to single out the one factor that led to our relative success: we were all engineers and had been trained to organize our efforts along engineering lines. (Página 24)

[xi] 'Software engineering' was deliberately chosen as being provocative. (Página 26)

[xii] In 20 years [...], software engineering has evolve from a obscure idea practiced by a relatively small number of zealots to a legitimate engineering discipline. (Página 28)

[xiii] [...] unleashed a veritable storm of often acrimonious debate on the viability of software engineering. The inconsequence of this debate of more than thirty years is evident from a announcement of Pittsburgh's Carnegie Science Center's activities for this year's National Engineer's Week. The note said that, "engineers from all disciplines — chemical, electrical, mechanical, civil" would participate. (Página 30)

[xiv] The most-noted item ever published in Communications was a letter from Edsger W. Dijkstra entitled "Go To Statement Considered Harmful" which attempted to give a reason why the GOTO statement might be harmful. Although the argument was academic and unconvincing, its title seems to have become fixed in the mind of every programming manager and methodologist. Consequently, the notion that the GOTO is harmful is accepted almost universally, without question or doubt. To many people, "structured programming" and "GOTOless programming" have become synonymous. This has caused incalculable harm to the field of programming, which has lost an efficacious tool. It is like butchers banning knives because workers sometimes cut themselves. [...] In short, the belief that GOTOs are harmful appears to have become a religious doctrine, unassailable by evidence. I do not know if I can do anything that will dislodge such deeply entrenched dogma. At least I can attempt to reopen the discussion by showing a clearcut instance where GOTOs significantly reduce program complexity. [...] All of my experiences compel me to conclude that it is time to part from the dogma of GOTO-less programming. It has failed to prove its merit. (Página 31)

[xv] By today's standards we did not know what we are doing; we did not dream of giving any guarantee that our implementation would be correct because we knew full well that we lacked the theoretical knowledge that would be need for that. (Página 31)

[xvi] For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. [...] Am I to judge by whom my thinking has been influenced? It's fairly obvious that I am not uninfluenced by Peter Lundin and Christopher Strachey. [...] The remark about the undesirability of the **go to** statement is far from new [...], but I have not been able to trace it, presumably, it has been made by C. A. R. Hoare. (Página 32)

[xvii] In 1968, the *Communications of the ACM* published a text of mine under the title "The goto statement considered harmful", which in later years would be the most frequently referenced, regrettably, however, often by authors who had seen no more of it than its title. This text became the cornerstone of my fame by become a template: we would see all sorts of articles under the title "X considered harmful" for almost any X, include one titled "Dijkstra considered harmful". But what had happened? I had submitted a paper under the title "A case against goto statement", which, in order to speed up its publication, the editor had changed into a "Letter to the Editor", and in the process he had given it a new title of his own invention! The editor was Niklaus Wirth. (Página 33)

[xviii] (...) the need for program flowcharts was an article of faith. In the 1950s and the 1960s, computer flowcharting created its own literature and culture. The American Standards Institute produced standard charting conventions, stationers produced stencils and drawing aids, and college programming instructors required their students to use flowcharts. By the second half of the 1960s, software packages such as Autoflow had been developed to produce flowcharts automatically. In the early 1970s, a fad for "structured programming" swept aside 20 years of flowcharting history in about 3 years. In a similar way, the hierarchy of labor was challenged, and the arbitrary distinctions between senior and junior programmers and between programmers and coders were lost. Indeed the term "coder" fell out of use. (Página 33)

[xix] The silver bullet is a cultural change rather than a technological change. It is a paradigm shift — a software industrial revolution based on reusable and interchangeable parts that will alter the software universe as surely as the industrial revolution changed manufacturing. [...] To get a grip on object-oriented means coming to the realization that is an end, not a means — an objective rather than technologies for achieving it. [...] Object-oriented means abandoning the process-centric view of the software universe [...] in favor of a product-centered [...]. (Página 36)

[xx] An order-of-magnitude gain can be made by object-oriented programming only if the unnecessary underbrush of type specification remaining today in our programming languages is itself responsible for nine-tenth of the work involved in designing a program product. I doubt it. (Página 36)

[xxi] An excellent 1990 paper by Brad Cox, "There Is a Silver Bullet", argues eloquently for the reusable, interchangeable component approach as an attack on the conceptual essence of the problem. I enthusiastically agree. Cox however misunderstands "NSB" [...], he reads it as asserting that software difficulties arise "from some deficiency in how programmer build software today." My argument was that the essential difficulties are inherent in the conceptual complexity of the software functions to be designed and built at any time, by any method. (Página 36)

[xxii] Replacing the idiosyncratic 'artistic' ethos that has long governed software writing with a more efficient, cost-effective engineering mind-set. (Página 39)

[xxiii] [The COMPUTER metaphor ...] returns to the Cartesian metaphor of the mind as a mathematical engine, but with a massively elaborated concrete structure that vastly enriches the Cartesian concept. (Página 43)

[xxiv] Programming in the 1950's was a black art, a private arcane matter ... each problem required a unique beginning at square one, and the success of a program depended primarily on the programmer's private techniques and inventions. (Página 46)

[xxv] If it should ever turn out that the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence that I have ever encountered. (Página 47)

[xxvi] That such a conference was opportune hardly needs emphasis, for the shortage of competent candidates for positions in the computer field is becoming increasingly critical as the demand for computers grows. Not too widely recognized is the fact that this problem is of importance to the entire mathematical fraternity. (Página 47)

[xxvii] Our principles may be summarized under four headings. (1) *Computers are mathematical machines.* Every aspect of their behavior can be defined with mathematical precision, and every detail can be deduced from this definition with mathematical certainty by the laws of pure logic. (2) *Computer programs are mathematical expression.* They describe with unprecedented precision and in every minutest detail the behavior, intended or unintended, of the computer on which they are executed. (3) *A programming language is a mathematical theory.* It includes concepts, notations, definitions, axioms and theorems, which help a programmer to develop a program which meets its specification, and prove that it does so. (4) *Programming is a mathematical activity.* Like other branches of applied mathematics and engineering, its successful practice requires determined and meticulous application of traditional methods of mathematical understanding, calculation and proof. These are general philosophical and moral principles, and I hold them to be self-evident which is just as well, because all the actual evidence is against them. Nothing is really as I have described it, neither computers nor programs nor programming languages nor even programmers. (Página 49)

[xxviii] As a design tool, mathematical semantics was still far from the goal of correcting the anomalies that gave rise to errors in real programming languages. If computers and programs were "inherently mathematical objects" the mathematics of computers and programs of real practical concern had so far proved illusive. (Página 49)

[xxix] It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and mathematical elegance. (Página 50)

[xxx] Since the end of World War II, engineering educators had embraced an analytical and scientific approach [...]. (Página 51)

[xxxi] We can briefly mention some times where science was self-consciously applied to software engineering. Some have argued that the underlying science is still not complete enough to truly support software engineering in the same way as it does in older engineering disciplines. On this point we agree, but engineering knowledge is gained by practice first, then later by scientific explanation of successful practice. The fact that there are not scientific explanations of all of software engineering fits a nascent field. There are still examples where science has risen to the support of software engineering. These examples are uniformly from mathematics, which has turned out to be software engineering's language and also its foundation. (Página 51)

[xxxii] Engineering cannot be contrasted with art. The more good art there is in design, the better. The more daring the design is, the greater the success (or failure). [...] It may be that English-speaking engineers derive the designation of their profession from engines (e.g., war machines). By contrast, we Germans (still) prefer to use the French word "ingénieur" as imported more than 300 years ago. It is closely related to the word for ingenious in French (ingénieux) for with Petit Larousse gives de following definition: full of spirit, inventiveness and skill. (Página 52)

[xxxiii] Most (but not necessarily all) modeling techniques focus on functions, data or objects as elementary building blocks. The implicit and/or explicit underlying assumptions are that: (1) these building blocks exist in the world (realism) and (2) there is an objectively definable set of things whose definition is independent of the perceptions of the developer (objectivism). The implication of the first assumption is that it is the developer's job to "find" those objects as though they were the treasures of a sunken ship washed up on shore just waiting to be picked up by the first one to come along. The implication of the second assumption is that any two developers should come up with the same model (because they will find the same treasures) and if there are differences they are resolvable. If two developers do see things differently, assumption (2) suggests that one developer is not seeing the application as clearly as the other, or that one developer is simply not as good as the other. (Página 53)

[xxxiv] This is an important truth: when you're attacking complexity by partitioning, the thinner the interface, the better the partitioning — if the interfaces are still thick, go back and partition again, searching for the natural seams of the domain. (Página 54)

[xxxv] They were stuck on the method because it gave a comforting sense of completeness; it appeared to them to be The Answer to all of their problems. When it didn't solve their problems they blamed themselves and tried harder. I now believe that my 1975 book was overly persuasive and that many in our industry were simply seduced by it. This is party the result of my unconstrained enthusiasm for a method that had worked superbly for me (in a limited domain) (...). (Página 54)

[xxxvi] Programmers, systems analysts, and other software workers are experiencing efforts to break down, simplify, routinize, and standardize their own work so that it, too, can be done by machines rather than people. ... Elaborate efforts are being made to develop ways of gradually eliminating programmers, or at least reduce their average skill levels, required training, experience, and so on. ... Most of the people that we call programmers, in short, have been relegated largely to subsidiary and subordinate roles in the production process. ... While a few of them sit at the side of managers, counseling and providing expert's advice, most simply carry out what someone else has assigned them. (Página 55)

[xxxvii] Even though manual and intellectual tasks are significantly different, we can measure, analyze, and optimize both and thus apply Taylor's principles equally well. (Página 55)

[xxxviii] Any intellectual process that can be carried out mechanically can be performed by a general purpose digital computer. (Página 56)

[xxxix] You have to involve the team all the way through the change process, understand their doubts and involving them in planning the new process. By making them stakeholders in the process change, it is much more likely that they will wish to make it work. (Página 58)

[xl] Project managers have to solve technical and nontechnical problems by using the people in their team in the most effective way possible. They have to motivate people [...] Poor management of people is one of the most significant contributors to project failure. (Página 58)

[xli] I called it 'The Romance of Programming', and it seemed to have been the best thing I've ever done. More people remember me for that speech than for more significant contributions I made to the program. (Página 59)

[xlii] I believe that a lot of what we construe as being theory and practice is in fact architecture and engineering. [...] I don't believe for instance that the majority of what Dijkstra does is theory — I believe that in time we will probably refer to the "Dijkstra School of Architecture". (Página 59)

[xliii] Software engineering, in a misguided attempt to gain legitimacy, deny the artistic aspects of their field. However, Fred Brooks found a truism that indicates that art, rather than handbook engineering, is clearly dominant. (Página 60)

[xliv] The only other projects of comparable size and complexity at this time were being undertaken at IBM. Most other commercial software developers were working on smaller, more self-contained efforts requiring far fewer programmers. Programmers at these installations worked on multiple projects involving a diverse range of business problems. They often participated in every aspect of system development, from requirements gathering to system design to implementation [...]. (Página 61)

[xlv] Project and team sociology may be a bit outside your field of expertise, but not beyond your capabilities. Whatever you name these people-related problems, they're more likely to cause you trouble [...] than all the design, implementation, and methodology issues you'll have to deal with. In fact, that idea is the underlying thesis of this whole book. (Página 62)

[xlvi] When we try to control the world with computer programs or methods for systems development, we should not forget that the mechanist world view is based on the assumption that the world we are trying to understand and control is itself an ordered, fundamentally unchanging system. (Página 63)

[xlvii] Relatively few errors now occur in the stages in the bottom half of the scheme. That is not surprising, given that they are the aspects of computing best understood mathematically, and that understanding has been translated into such practical tools as diagnostic compilers for high-level programming languages. (Página 63)

[xlviii] [...] is where software engineering has focused its attention since 1970s. But that is also where the science of software moves away from the computer into the wider world and interacts with the sciences (if they exist) pertinent to the systems to be modeled computationally. [...] But that is a question that software engineers share with scientist who have turned to the computer to take them into realms that are accessible neither to experiment nor to analytical mathematics. (Página 64)

[xlix] It is a truism to say that computers have become ubiquitous in today's organizations. Since their application in administrative data processing in the mid-1950s, they have become one of the key instruments for improving the formal information processing activities of organizations. In less than four decades, computer-based information systems have evolved from supporting back office, already formalized, systems such as payroll, to penetrating the entire organization. (Página 83)

[l] All phenomena are the effect or the product of heterogeneous networks. But in practice we do not cope with endless network ramification. Indeed, much of the time we are not even in a position to detect network complexities. So what is happening? The answer is that if a network acts as a single block, then it disappears, to be replaced by the action itself and the seemingly simple author of that action. At the same time, the way in which the effect is generated is also effaced: for the time being it is neither visible, nor

relevant. So it is that something much simpler — a working television, a well-managed bank or a healthy body — comes, for a time, to mask the networks that produce it. (Página 103)

[li] [...] the institution responsible for the war conceived the problem it was supposed to solve in mechanistic terms of physical science. (Página 104)

[lii] Cognitive theories, like computer technology, were first created to assist in mechanizing military tasks previously performed by human beings. (Página 104)

[liii] RCA wanted a software package that would enable programmers to produce flowcharts mechanically [...]. (Página 104)

[liv] Rationalism promotes a particular view of the clientele or end users of technological systems. Their participation in the process of inventing and designing computer systems is minimized. The experts have access to the theories and are best placed to deliver the appropriate designs. (Página 106)

[lv] David Parnas [...] comment[s] on the differences in the concerns of software developers and researchers and theorists. He observed what he thought was a substantial difference between software activities and work in other fields of engineering. "Whereas practicing engineers find things of value in research publications, most software developers do not." As a result, Parnas found that "the developers major problems were problems that I had never considered, problems that none of my professors or colleagues thought worthy of discussion." Parnas assumed — incorrectly — that this gap between research and the real world was unique to the software field. In fact, differences like this have emerged in many academic engineering fields since 1945, as researchers and theorists go separate ways. (Página 109)

[lvi] Standards and procedures are the basis of quality management, experienced quality managers recognize that there are intangible aspects of software quality [...] that cannot be embodied in standards. (Página 130)

[lvii] In the age of high modernity and globalization, however, more knowledge may just as well lead to *more unpredictability*, *more uncertainty*, and *less controllability*. [This ...] contradicts the instrumental optimism regarding the predetermined controllability of uncontrollable things [...]. Modernization means integration. At the same time, all change — new technologies introduced, organizational structures and work procedures implemented, and so on — has unintended side effects. Any change may affect those interacting with processes that are involved in the change. [...] We are living in the age of side effects... The side effect, not instrumental rationality, is becoming the motor of social change. (Página 144)

[lviii] How does information technology marginalize the ethical? This divorce is most evident in the common distinction made between technical issues and human-factor issues. [...] Far from sustaining a dynamic, indeterminate, dialectical tension, the opposition between the technical and human factors is thoroughly entrenched and institutionalized in the way information technology industries and practices are organized. For example, courses within most universities and colleges distinguish between technology and its application and critical evaluation. The former commonly belongs to engineering, the latter to the humanities. [...] The engineering and technology disciplines are thought to genuinely and unambiguously contribute to international competitiveness and societal well-being. [...] Coupled with this bifurcation of the technical and ethical is the common assumption of the neutrality of technology. Having established the distinction between technique and value, technological rationality now impresses on us that objects are neither good nor evil, but human use makes them so. (Página 145)

[lix] [...] with the Macintosh (and then, by extension, the ubiquitous Macintosh-style Windows interfaces introduced by Microsoft), people had moved away from a reductive and mechanistic view of how to relate to a computer and were "learning to take the machine at (inter)face value."In the 1970s and early 1980s, computers carried a modernist ethos: analyze and you shall know; by the mid-1990s, the complex simulation worlds of opaque computers offered an experience that called these assumptions into question. Culturally, the Macintosh carried the idea that it is more fruitful to explore the world of shifting surfaces than to embark on a search for mechanism, origins, and structure. The Macintosh way of understanding stood in contrast to the modernist perspective that animated the writings of such thinkers as Freud, Marx, and Darwin, who suggested that understanding proceeds by reducing complex things to simpler elements,

by discovering the hidden mechanisms behind behavior. Analyze and you shall know presented itself as a way of understanding the self and the social world. As a way of thinking, it animated the personal computer owners I write about in The Second Self. They were populist computer utopians who saw the computer as providing widespread access to information (previously available only to elites) that would encourage political engagement. Beyond this, they believed that a transparent relationship with computers would be empowering, that once people could own and understand something as complex as a computer, they would demand greater transparency in political decision-making processes. (Página 149)