Relatório Técnico

Revisão *Quasi*-Sistemática da Literatura: Conciliação de processos de desenvolvimento de software

Andréa Magalhães Magdaleno
(andrea@cos.ufrj.br)

Cláudia Maria Lima Werner
(werner@cos.ufrj.br)

Renata Mendes de Araujo
(renata.araujo@uniriotec.br)

PESC
Programa de Engenharia de Sistemas e Computação

COPPE/UFRJ

Rio de Janeiro, Dezembro de 2009

Revisão *Quasi*-Sistemática da Literatura: Conciliação de processos de desenvolvimento de software

Andréa Magalhães Magdaleno[1]
Cláudia Maria Lima Werner[1]
Renata Mendes de Araujo[2]

[1]Programa de Engenharia de Sistemas e Computação (PESC) – COPPE/UFRJ
Caixa Postal 68.511 – 21945-970 – Rio de Janeiro – RJ – Brasil

[2]Programa de Pós Graduação em Informática (PPGI) – Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO) – 22290-240 – Rio de Janeiro – RJ – Brasil

{andrea, werner}@cos.ufrj.br, renata.araujo@uniriotec.br

RESUMO

Este relatório técnico tem por objetivo descrever uma pesquisa relacionada à conciliação de processos de desenvolvimento de software tradicional, ágil e livre. Um protocolo de pesquisa foi utilizado para conduzir uma revisão *quasi*-sistemática da literatura. As buscas foram realizadas no período de abril-maio de 2009. Os dados obtidos da literatura foram analisados e permitiram observar a escassez de trabalhos que tratem da conciliação dos três processos de desenvolvimento e a existência de uma quantidade significativa de trabalhos abordando a conciliação do desenvolvimento tradicional com o ágil. Diante da imaturidade desta área de pesquisa, cujos trabalhos iniciais datam de 2001, é razoável supor que esta combinação de apenas dois processos é uma estratégia inicial, pois neste cenário as distâncias a serem percorridas, para superar as diferenças entre os processos, são menores. Ao mesmo tempo, também nota-se uma quantidade significativa de relatos de organizações que estão tentando combinar estes processos por entenderem que eles se complementam. Entretanto, a maioria das propostas já existentes ainda caminha no sentido da combinação de práticas de modelos diferentes ou da adaptação de modelos tradicionais. Neste trabalho de pesquisa defende-se que os objetivos de flexibilidade, qualidade e produtividade só serão alcançados pelas organizações através do balanceamento da colaboração e disciplina. Ambas, a colaboração e a disciplina, estão presentes, ainda que com diferentes ênfases, no desenvolvimento de software tradicional, ágil e livre e são estes os fatores determinantes para adaptar um processo de desenvolvimento de software às necessidades e particularidades dos projetos e das organizações.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

Nas últimas décadas assistimos à globalização dos mercados. O cenário mundial oferece novas oportunidades de negócio, mas também apresenta grandes desafios. Neste ambiente competitivo, as organizações precisam de flexibilidade e velocidade para responder às demandas dos clientes, oferecendo rapidamente produtos e serviços de qualidade.

Em particular, as organizações de desenvolvimento de software são continuamente desafiadas pela necessidade de melhorar a qualidade dos produtos de software gerados. As tecnologias evoluem rapidamente e os sistemas de software estão se tornando cada vez maiores e mais complexos. Somam-se a isso, os novos desafios trazidos pelo desenvolvimento distribuído e em larga escala.

Neste contexto, a premissa de que o processo de desenvolvimento de software adotado influencia diretamente na qualidade do produto gerado (PRESSMAN, 2001), levou muitas organizações a adotarem os modelos de qualidade, tais como o CMMI (CHRISSIS ET AL., 2006), a ISO 12207 (ISO/IEC, 2007) e o MPS-BR (SOFTEX, 2009), para tornar o seu processo de desenvolvimento menos caótico, mais previsível e mais disciplinado (CONBOY AND FITZGERALD, 2004, PATEL ET AL., 2006).

Por outro lado, o sucesso de alguns projetos de software livre como Linux, Apache[1] e Mozilla[2], chamou a atenção da academia, da indústria e dos usuários por produzir software de alta qualidade, rápida e gratuitamente (CUBRANIC AND BOOTH, 1999, FELLER AND FITZGERALD, 2001, HAEFLIGER ET AL., 2007). O software livre não traz em si algum tipo de ruptura tecnológica, mas traz uma nova alternativa viável de desenvolvimento de produtos de software baseada em processos de desenvolvimento com características próprias, que tem quebrado alguns princípios dos modelos tradicionais (RAYMOND, 2001, REIS, 2003).

Os métodos ágeis também se apresentaram como uma alternativa ao desenvolvimento de software tradicional ao lidar com as mudanças que surgem durante um projeto de desenvolvimento através de ciclos de desenvolvimento mais curtos e com alto nível de envolvimento e participação do cliente. Assim, produzem software com um mínimo de documentação, utilizando equipes que se auto-organizam para lidar com o trabalho, contando com o conhecimento tácito adquirido ao longo do projeto (BECK ET AL., 2001, COCKBURN, 2001, HIGHSMITH AND COCKBURN, 2001).

Cada um com as suas peculiaridades, seus casos de sucesso e seus desafios, estes três modelos de desenvolvimento seguiram caminhos distintos. Devido a problemas de diferenças de vocabulário, más interpretações e mau uso das abordagens, os três modelos costumam ser percebidos como opositores, mas visando ganhos de qualidade e produtividade, alguns autores propõem a conciliação entre eles (BOEHM, 2002, BOEHM AND TURNER, 2003, GLASS, 2001, GLAZER ET AL., 2008, WARSTA AND ABRAHAMSSON, 2003).

Em geral, as propostas de conciliação existentes caminham no sentido da comparação e combinação das práticas sugeridas pelos diferentes modelos, visando à obtenção de um novo modelo de processos híbrido. Por exemplo, Santana et al (2006), sugerem o mapeamento das práticas entre o XP e o MPS-BR, verificando a sua aderência e tentando chegar a um modelo de processos híbrido que permita o uso conjunto dos modelos. Na mesma linha de pesquisa, mas substituindo o MPS-BR pelo CMM, Paulk (2001) compara o XP com o CMM para identificar similaridades e possibilidades de combinação entre eles. Paulk (2001) analisa também como o XP pode ajudar as organizações a atingirem os objetivos do CMM.

[1] Site Projeto Apache: http://www.apache.org
[2] Site Projeto Mozilla: http://www.mozilla.org

Contudo, a natureza complexa da atividade de desenvolvimento de software e a grande variedade de métodos existentes tornam a tarefa de comparação dos modelos de desenvolvimento, uma tarefa árdua e imprecisa (BOEHM AND TURNER, 2003). Além disso, neste tipo de proposta não se consegue garantir que o processo resultante realmente tenha os níveis de disciplina ou agilidade desejados.

Boehm e Turner (2003) sugerem a análise de riscos das características do projeto como o caminho para alcançar o equilíbrio entre agilidade e disciplina. Esta análise considera três categorias de riscos: os riscos do ambiente, os riscos de se adotar um modelo ágil e os riscos de se adotar um modelo de desenvolvimento tradicional. Entretanto, esta proposta foca apenas no desenvolvimento ágil e no tradicional, sem levar em consideração o processo de desenvolvimento de software livre.

Neste trabalho de pesquisa, defende-se que a conciliação significa mais do que a combinação de práticas dos diferentes modelos de processos. A hipótese investigada é que é possível equilibrar os aspectos de colaboração e disciplina, presentes em cada um dos modelos de desenvolvimento. A colaboração entre as pessoas envolvidas no projeto de desenvolvimento pode ser compreendida através da análise de redes sociais e a disciplina pode ser calibrada através do formalismo adotado na definição dos processos.

Neste contexto, o primeiro passo é estudar a viabilidade dessa conciliação e quais abordagens têm sido propostas por outros pesquisadores ou adotadas por organizações que implantaram práticas de processos de desenvolvimentos distintos. Assim, foi realizada uma revisão sistemática da literatura (BIOLCHINI ET AL., 2005, KITCHENHAM, 2004), cujo planejamento, execução e resultados são apresentados neste relatório.

Este trabalho está organizado da seguinte forma: na seção 2, é apresentado o processo da revisão sistemática da literatura. Na seção 3, é descrito o planejamento elaborado para esta revisão *quasi*-sistemática. Na seção 4, descreve-se a execução do protocolo apresentado na seção 3. Na seção 5, são analisados os documentos recuperados, extraídas as informações relevantes e resumidos os resultados obtidos com a revisão *quasi*-sistemática. Na seção 6, são respondidas as questões de pesquisa, caracterizando a conciliação de processos de desenvolvimento de software. Por fim, na seção 7, são apresentadas as conclusões deste trabalho e as oportunidades de trabalhos futuros.

2.Processo da Revisão Sistemática da Literatura

Kitchenham (2004) define uma revisão sistemática da literatura como um “meio de identificar, avaliar e interpretar toda pesquisa disponível relevante a uma questão, ou área, ou fenômeno de interesse de uma pesquisa particular”. Revisões sistemáticas são baseadas em uma estratégia de pesquisa definida, que visa detectar o máximo possível de literatura relevante.

A principal razão para a realização de uma revisão sistemática é aumentar a qualidade do material sobre o assunto de interesse. A revisão sistemática pode ajudar a orientar o processo de investigação, evitando a duplicação desnecessária de esforços e erros. Em contraste com uma revisão convencional da literatura, realizada em uma forma *ad-hoc*, sempre que se começa uma investigação particular, uma revisão sistemática segue uma sequência bem definida e rigorosa de passos metodológicos (Figura 1), que oferecem um elevado valor científico para os resultados obtidos. No entanto, uma revisão sistemática requer um esforço considerável em comparação à uma revisão convencional da literatura, mas esse é o preço a ser pago por uma forma aprofundada e completa de investigação em uma área de interesse.

O processo de revisão sistemática da literatura (BRERETON ET AL., 2007) é constituído por dez passos, que podem ser agrupados em três fases principais. Durante a fase de *planejamento da revisão*, o pesquisador identifica as necessidades de revisão, especifica a questão de pesquisa, e desenvolve o protocolo da revisão (ver seção 3). No final desta fase, o protocolo deve ser validado para garantir que o planejamento é possível, antes de executar a revisão.

Na fase de *execução da revisão*, as buscas nas fontes definidas são executadas. Os estudos obtidos são avaliados de acordo com os critérios estabelecidos no protocolo. Em seguida, os dados relevantes encontrados nos trabalhos selecionados são extraídos e sintetizados.

Finalmente, os resultados da revisão são publicados na terceira fase. Na prática, esta fase de *documentação da revisão* é realizada durante todo o processo, para armazenar os resultados de todas as fases anteriores. A Figura 1 ilustra a visão geral do processo de revisão sistemática da literatura que foi utilizado no presente trabalho.

Figura 1 – Processo da Revisão Sistemática da Literatura (Fonte: Adaptado de: (BRERETON ET AL., 2007))

Tendo em vista que o objetivo deste estudo é realizar um estudo exploratório de caracterização da área, não haverá comparação. Assim, como um dos conjuntos da busca estará vazio, a presente revisão caracteriza-se como uma revisão *quasi*-sistemática, apesar de preservar o mesmo formalismo e seguir o mesmo processo da revisão sistemática (TRAVASSOS ET AL., 2008).

3.Planejamento da Revisão *Quasi*-Sistemática

Esta revisão *quasi*-sistemática inicia pela definição de um protocolo de revisão que especifica a questão principal e as questões secundárias da pesquisa e os métodos que serão utilizados para executar a revisão. O protocolo deve explicitar os critérios de inclusão e exclusão para acessar cada estudo potencial e documentar a estratégia de busca utilizada, de forma a permitir que leitores (e outros pesquisadores) possam conhecer seu grau de rigor e completeza.

As próximas subseções descrevem os componentes do protocolo utilizado na presente revisão. Este protocolo foi definido com base no *template* proposto por Biolchini *et al*. (2005) e no exemplo apresentado por Abrantes e Travassos (2007).

Assim, será adotada uma abordagem PICO que estrutura a questão de pesquisa em quatro elementos básicos: população, intervenção, comparação e resultado (PAI ET AL., 2004).

3.1. Objetivo

O objetivo desta pesquisa é caracterizar a conciliação de processos de desenvolvimento de software tradicional, livre e ágil.

Em particular, almeja-se identificar as oportunidades e desafios e as propostas para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil. Além disso, busca-se por relatos ou resultados de organizações ou comunidades que implantaram em seus processos de desenvolvimento de software, práticas oriundas de pelo menos duas abordagens de desenvolvimento de software tradicional, livre ou ágil. Por último, deseja-se verificar a existência de modelos de processos de desenvolvimento de software já definidos e que combinem pelo menos duas dessas abordagens.

Contudo, vale ressaltar que não se pretende investigar características individuais de cada um dos processos de desenvolvimento de software ou relatos sobre o uso individual dessas abordagens em projetos de desenvolvimento de software.

3.2. Questões de pesquisa

Questão principal (QP):

Como caracterizar a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?

Questões secundárias (QS):

Uma vez que os trabalhos existentes sobre a conciliação de processos de desenvolvimento de software sejam identificados, propõe-se uma caracterização através das seguintes questões:

- *QS1 - Quais são as oportunidades e os desafios para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?*
- *QS2 - Quais são as propostas, de outros pesquisadores ou especialistas, para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?*
- *QS3 - Quais são as estratégias adotadas, pelas organizações ou pelas comunidades, para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?*
- *QS4 - Quais foram os resultados obtidos pelas organizações ou comunidades que conciliaram processos de desenvolvimento de software tradicional, livre e ágil?*
- *QS5 - Quais são os modelos de processos de desenvolvimento de software híbridos existentes que combinem as abordagens tradicional, livre e ágil?*

3.3. Escopo

Aplicação:

Servir de base ou apoiar pesquisas sobre a conciliação de processos de desenvolvimento de software tradicional, livre e ágil.

População:

Organizações de desenvolvimento de software e Projetos de desenvolvimento de software.

Intervenção:

Processo de desenvolvimento de software (tradicional, livre e ágil).

Comparação:

Não há.

Resultados:

- Caracterização da conciliação dos processos de desenvolvimento de software tradicional, livre e ágil;
- Identificação das oportunidades e desafios da conciliação dos processos de desenvolvimento de software tradicional, livre e ágil;
- Identificação das propostas de conciliação dos processos de desenvolvimento de software tradicional, livre e ágil;
- Identificação das estratégias adotadas pelas organizações ou comunidades para a conciliação dos processos de desenvolvimento de software tradicional, livre e ágil;
- Identificação dos resultados obtidos pelas organizações ou comunidades que conciliaram processos de desenvolvimento de software tradicional, livre e ágil;
- Identificação dos modelos de processos de desenvolvimento de software híbridos que conciliem práticas do desenvolvimento tradicional, livre e ágil.

Controle:

- Tradicional, livre e ágil: nenhum

Durante a revisão inicial da literatura não foram identificados trabalhos que tratem da conciliação entre os três processos de desenvolvimento de software.

- Livre e ágil: nenhum
- Tradicional e livre: nenhum
- Tradicional e ágil: 3

BOEHM, B.; TURNER, R., 2003, "Using risk to balance agile and plan-driven methods", *IEEE Computer*, v. 36, n. 6, pp. 57-66.

BOEHM, B.; TURNER, R., 2004, " Balancing agility and discipline: evaluating and integrating agile and plan-driven methods", *International Conference on Software Engineering (ICSE)*, pp. 718-719.

VINEKAR, V.; SLINKMAN, C. W.; NERUR , S., 2006, "Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View", *Information Systems Management*, v. 23, n. 3, pp. 31-42.

Idioma:

Para a realização desta pesquisa foi selecionado o idioma inglês, devido à sua adoção pela grande maioria das conferências e periódicos internacionais relacionados com o tema de pesquisa e por ser o idioma utilizado pela maioria das

bases eletrônicas relacionadas com o tema listadas no Portal de Periódicos da CAPES. Em uma busca preliminar, não foram encontrados artigos escritos em português que trouxessem contribuição para o tema da pesquisa. Além disso, textos em português, embora se reconheça a sua importância, muitas vezes não se encontram indexados, o que aumenta o esforço ou impede sua busca.

Tipos de Documentos:

Qualquer tipo de trabalho ou artigo que trate da conciliação dos processos de desenvolvimento de software tradicional, livre ou ágil.

3.4. Seleção de fontes

Para as bibliotecas digitais é desejado:

- Possuir engenho de busca que permita o uso de expressões lógicas ou mecanismo equivalente;
- Possuir engenho de busca que permita a busca no texto completo ou em campos específicos das publicações;
- As publicações devem pertencer a uma das editoras listadas no Portal de Periódicos da CAPES;
- Os mecanismos de busca utilizados devem garantir resultados únicos através da busca de um mesmo conjunto de palavras-chave.

Assim, as bibliotecas digitais selecionadas foram:

Compendex (em modo *Expert Search*): <http://www.engineeringvillage2.org>

IEEE (em modo *Advanced Search*): <http://ieeexplore.ieee.org>

Scopus (em modo *Advanced Search*): <http://www.scopus.com>

Web of Science (em modo *Advanced Search*): <http://apps.isiknowledge.com>

Estas bibliotecas foram escolhidas porque são as que se tem acesso para recuperação de referências, bem como maior facilidade para recuperação do texto completo do artigo. Além disso, estas fontes foram consideradas significativas no sentido de oferecerem publicações pertinentes e que podem contribuir para o resultado da pesquisa.

Vale ressaltar ainda que a biblioteca da ACM (*Association for Computing Machinery*), apesar da sua importância dentro da área de computação, não atende ao último critério estabelecido, visto que a sua máquina de busca não garante a acurácia dos resultados retornados e diferentes execuções de uma mesma consulta nem sempre trazem os mesmos resultados (SOUZA, 2008). Além disso, esta biblioteca possui diversas redundâncias com a biblioteca do IEEE e o seu conteúdo também é indexado pela biblioteca Scopus. Logo, apesar desta biblioteca não ter sido incluída entre as fontes selecionadas, entende-se que o seu conteúdo estará contemplado.

3.5. Palavras-chave

- **Questão principal (QP):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM, open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI*

Resultado: *combination, adaptation, conciliation, reconciliation, balance, aggregation, tailoring, integration, customization, coexist, juxtaposition, compatibility*

- **Questão secundária 1 (QS1):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM, open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI*

Resultado: *opportunity, advantage, benefit, profit, reward, strength, challenge, problem, difficult, drawback, obstacle, trouble, weakness*

- **Questão secundária 2 (QS2):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM, open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI*

Resultado: *proposal, suggestion, proposition, idea*

- **Questão secundária 3 (QS3):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM,*

open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI

Resultado: *strategy, design, plan*

- **Questão secundária 4 (QS4):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM, open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI*

Resultado: *result, experience, history, consequence, outcome, knowledge, conclusion, effect, solution, findings, lesson, success, failure, case study*

- **Questão secundária 5 (QS5):**

População: *organization, organisation, enterprise, corporation, business, company, industry, firm, community, project, development, engineering, software, system, application, product*

Intervenção: *process, activity, method, approach, practice, methodology, technique, paradigm, procedure, principle, agile, lightweight, lean, XP, SCRUM, open source, free source, libre, OSS, FOSS, FLOSS, conventional, traditional, plan-driven, plan driven, closed, rigorous, proprietary, CMM, CMMI*

Resultado: *model, framework*

3.6. Critérios de inclusão e exclusão

Na revisão serão incluídos todos os trabalhos encontrados com a utilização do método descrito, desde que o documento esteja disponível na web e satisfaça pelo menos um dos seguintes critérios de inclusão (CI):

- CI1 - Os documentos devem abordar a conciliação de dois ou mais processos de desenvolvimento de software tradicional, livre e ágil;
- CI2 - Os documentos devem discutir oportunidades e desafios para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil;
- CI3 - Os documentos devem apresentar propostas para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil;

- CI4 - Os documentos devem relatar experiências das organizações ou comunidades que implantaram práticas de processos de desenvolvimento de software tradicional, livre e ágil;
- CI5 - Os documentos devem descrever modelos de processos de desenvolvimento de software híbridos que combinem práticas do desenvolvimento de software tradicional, livre e ágil.

Devem ser excluídas as publicações contidas no conjunto preliminar que satisfaçam a pelo menos um dos seguintes critérios de exclusão (CE):

- CE1 – Os documentos que não tenham sido escritos em inglês serão excluídos;
- CE2 – Os documentos que não tenham o texto completo do trabalho disponível através da internet e que também não se consiga através de contato com os autores serão excluídos;
- CE3 – Os documentos que nitidamente tratem de outros assuntos não pertinentes ao objetivo desta revisão *quasi*-sistemática serão excluídos;
- CE4 - Os documentos que se limitem a relatar o uso individual dos processos de desenvolvimento de software tradicional, livre e ágil em projetos de desenvolvimento de software serão excluídos;
- CE5 – Os documentos que não abordem o processo de desenvolvimento de software, mas sim ferramentas ou ambientes computacionais específicos de apoio ao desenvolvimento de software, serão excluídos;
- CE6 – Os documentos que tratem da conciliação entre os modelos de desenvolvimento, mas com foco em aspectos legais, no modelo de negócio ou nos métodos de avaliação, e não no processo de desenvolvimento em si, serão excluídos;
- CE7 – Caso o mesmo estudo tenha sido publicado mais de uma vez, a versão mais relevante, ou seja, aquela que explica o estudo em mais detalhes, será utilizada e a outra será excluída;
- CE8 – Caso o mesmo estudo já tenha sido selecionado por outra questão de pesquisa mais abrangente, este estudo será excluído da lista de seleções da questão de pesquisa mais restrita.

3.7. Processo de seleção dos trabalhos

A seleção dos estudos dar-se-á em quatro etapas:

i) Seleção e catalogação preliminar dos documentos coletados: A seleção preliminar das publicações será feita a partir da aplicação da expressão de busca às fontes selecionadas. Cada publicação será catalogada no repositório de dados do estudo para análise posterior.

ii) Seleção dos documentos relevantes: A seleção preliminar com o uso da expressão de busca não garante que todo o material coletado seja útil no contexto da pesquisa, pois a aplicação das expressões de busca é restrita ao aspecto sintático. Dessa forma, após a identificação das publicações através dos mecanismos de buscas, serão lidos os resumos dos trabalhos para que sejam analisados seguindo os critérios de inclusão e exclusão estabelecidos.

iii) Avaliação dos documentos relevantes: a lista de documentos incluídos e excluídos será avaliada pelos demais pesquisadores. Os pesquisadores deverão entrar em consenso sobre a seleção das publicações cujas avaliações se mostrem conflitantes. Em caso de impasse entre os pesquisadores, a publicação deverá ser incluída na lista de selecionadas. Para diminuir o risco de que uma publicação seja excluída prematuramente em uma das etapas do estudo, sempre que existir dúvida a publicação não será excluída.

iv) Extração de informações dos documentos relevantes: após a definição da lista final de documentos relevantes, os documentos serão lidos pelos pesquisadores para extração das informações sobre conciliação de processos de desenvolvimento de software. Esta etapa também será avaliada pelos outros pesquisadores.

3.8. Avaliação da qualidade dos trabalhos

Neste trabalho de pesquisa, será considerado que as fontes dos documentos são confiáveis, e que os textos tenham passado por revisões externas que serviram de filtragem para que tenham qualidade suficiente para contribuir com a revisão *quasi*-sistemática.

Em caso de informações conflitantes entre os documentos encontrados, será progressivamente atribuída maior relevância aos trabalhos que: defendam a ideia da conciliação de processos; apresentem uma proposta teórica para a conciliação de processos; contenham resultados de experiências reais das organizações que adotaram a conciliação de processos de desenvolvimento de software.

3.9. Estratégia de extração de informações

A estratégia de extração de informações é projetada para coletar as informações necessárias para responder as questões de pesquisa e avaliar a qualidade do estudo. Assim, de cada artigo aprovado pelo processo de seleção, o pesquisador extrai, e os outros revisam, as seguintes informações:

- Informações para referência do trabalho:
    - Título do documento;
    - Autor(es);
    - Data de publicação;
    - Fonte;
    - Processos de desenvolvimento de software envolvidos.
- Questão principal (QP):
    - Práticas conciliadas entre os processos de desenvolvimento de software;
    - Critérios de comparação.
- Questão secundária 1 (QS1):
    - Oportunidades para a conciliação;
    - Desafios para a conciliação.
- Questão secundária 2 (QS2):
    - Descrição da proposta para conciliação.

- Questão secundária 3 (QS3):
    - Estratégias adotadas pelas organizações.
- Questão secundária 4 (QS4):
    - Resultados obtidos pelas experiências das organizações.
- Questão secundária 5 (QS5):
    - Modelo de processos com conciliação.

3.10. Sumarização dos resultados

Os resultados serão tabulados. Com base nestes resultados serão definidas as estratégias de análise quantitativa e qualitativa. A análise qualitativa deve tecer considerações com o intuito de discutir os achados com relação às questões de pesquisa propostas. Nenhuma meta-análise será realizada.

3.11. *String* de busca

Na medida do possível, a *string* de busca será a mesma para todas as máquinas de busca. Contudo, poderá haver adaptações para se adequar a restrições de máquinas de busca específicas, observando-se as seguintes diretrizes:

- A *string* derivada deverá ser logicamente equivalente à *string* original, ou
- Na impossibilidade de se manter equivalência exata, deverá a *string* derivada ser mais abrangente para evitar perda de documentos potencialmente relevantes.

De acordo com Pai et al. (2004) os quatro elementos básicos que estruturam a questão de pesquisa podem ser relacionados com o operador lógico AND. Como o elemento básico “comparação” não é aplicável a este estudo, isto foi feito para o conjunto de palavras-chave escolhidas para representar cada um dos elementos “população” (P), “intervenção” (I) e “resultado” (R), resultando na seguinte estrutura: (P) AND (I) AND (R). Para cada um destes três elementos da estrutura, as respectivas palavras-chave foram combinadas com o operador lógico OR.

Devido ao grande número de termos, optou-se por dividir a busca em várias *strings,* de acordo com as combinações possíveis entre os processos de desenvolvimento de software tradicional, ágil e livre. Com isso a complexidade da *string* de busca foi reduzida, o que garante melhor legibilidade e diminui a probabilidade de erro (humano e das máquinas de busca das editoras).

3.11.1 Questão Principal (QP)

A partir das questões de pesquisa foram derivadas as strings de busca. As *strings* utilizadas para a questão principal são listadas a seguir:

- **QPa:**

String combinando os três processos de desenvolvimento de software:

("software development" OR "software engineering" OR "software project" OR "system development" OR "system engineering" OR "system project" OR "application development" OR "application engineering" OR "application project" OR "product development" OR "product engineering" OR "product project" OR "software organization" OR "software organisation" OR "software enterprise" OR

"software corporation" OR "software company" OR "software industry" OR "software firm" OR "software community")

AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)

AND (combination OR adaptation OR conciliation OR reconciliation OR balance OR aggregation OR tailoring OR integration OR customization OR coexist OR juxtaposition OR compatibility)

- **QPb:**

String combinando os processos livre e ágil:

("software development" OR "software engineering" OR "software project" OR "system development" OR "system engineering" OR "system project" OR "application development" OR "application engineering" OR "application project" OR "product development" OR "product engineering" OR "product project" OR "software organization" OR "software organisation" OR "software enterprise" OR "software corporation" OR "software company" OR "software industry" OR "software firm" OR "software community")

AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)

AND (combination OR adaptation OR conciliation OR reconciliation OR balance OR aggregation OR tailoring OR integration OR customization OR coexist OR juxtaposition OR compatibility)

- **QPc:**

String combinando os processos livre e tradicional:

("software development" OR "software engineering" OR "software project" OR "system development" OR "system engineering" OR "system project" OR "application development" OR "application engineering" OR "application project" OR "product development" OR "product engineering" OR "product project" OR "software organization" OR "software organisation" OR "software enterprise" OR "software corporation" OR "software company" OR "software industry" OR "software firm" OR "software community")

AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)

AND (combination OR adaptation OR conciliation OR reconciliation OR balance OR aggregation OR tailoring OR integration OR customization OR coexist OR juxtaposition OR compatibility)

- **QPd:**

String combinando os processos ágil e tradicional:

("software development" OR "software engineering" OR "software project" OR "system development" OR "system engineering" OR "system project" OR "application development" OR "application engineering" OR "application project" OR "product development" OR "product engineering" OR "product project" OR "software organization" OR "software organisation" OR "software enterprise" OR "software corporation" OR "software company" OR "software industry" OR "software firm" OR "software community")

AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR activity OR method OR approach OR practice OR methodology OR technique OR paradigm OR procedure OR principle)

AND (combination OR adaptation OR conciliation OR reconciliation OR balance OR aggregation OR tailoring OR integration OR customization OR coexist OR juxtaposition OR compatibility)

3.11.2 Questão Secundária 1 (QS1)

As *strings* utilizadas para a questão secundária 1 são listadas pela Tabela 1:

QS1	Processos	*String*
QS1a	Tradicional Ágil e Livre	**QPa AND** (opportunity OR advantage OR benefit OR profit OR reward OR strength OR challenge OR problem OR difficult OR drawback OR obstacle OR trouble OR weakness)
QS1b	Livre e Ágil	**QPb AND** (opportunity OR advantage OR benefit OR profit OR reward OR strength OR challenge OR problem OR difficult OR drawback OR obstacle OR trouble OR weakness)
QS1c	Livre e Tradicional	**QPc AND** (opportunity OR advantage OR benefit OR profit OR reward OR strength OR challenge OR problem OR difficult OR drawback OR obstacle OR trouble OR weakness)
QS1d	Ágil e Tradicional	**QPd AND** (opportunity OR advantage OR benefit OR profit OR reward OR strength OR challenge OR problem OR difficult OR drawback OR obstacle OR trouble OR weakness)

Tabela 1 – *Strings* da Questão Secundária 1

3.11.3 Questão Secundária 2 (QS2)

As *strings* utilizadas para a questão secundária 2 são listadas pela Tabela 2:

QS2	Processos	*String*
QS2a	Tradicional Ágil e Livre	**QPa AND** (proposal OR suggestion OR proposition OR idea)
QS2b	Livre e Ágil	**QPb AND** (proposal OR suggestion OR proposition OR idea)
QS2c	Livre e Tradicional	**QPc AND** (proposal OR suggestion OR proposition OR idea)
QS2d	Ágil e Tradicional	**QPd AND** (proposal OR suggestion OR proposition OR idea)

Tabela 2 – *Strings* da Questão Secundária 2

3.11.4 Questão Secundária 3 (QS3)

As *strings* utilizadas para a questão secundária 3 são listadas pela Tabela 3:

QS3	Processos	*String*
QS3a	Tradicional Ágil e Livre	**QPa AND** (strategy OR design OR plan)
QS3b	Livre e Ágil	**QPb AND** (strategy OR design OR plan)
QS3c	Livre e Tradicional	**QPc AND** (strategy OR design OR plan)
QS3d	Ágil e Tradicional	**QPd AND** (strategy OR design OR plan)

Tabela 3 – *Strings* da Questão Secundária 3

3.11.5 Questão Secundária 4 (QS4)

As *strings* utilizadas para a questão secundária 4 são listadas pela Tabela 4:

QS4	Processos	*String*
QS4a	Tradicional Ágil e Livre	**QPa AND** (result OR experience OR history OR consequence OR outcome OR knowledge OR conclusion OR effect OR solution OR findings OR lesson OR success OR failure OR case study)
QS4b	Livre e Ágil	**QPb AND** (result OR experience OR history OR consequence OR outcome OR knowledge OR conclusion OR effect OR solution OR findings OR lesson OR success OR failure OR case study)
QS4c	Livre e Tradicional	**QPc AND** (result OR experience OR history OR consequence OR outcome OR knowledge OR conclusion OR effect OR solution OR findings OR lesson OR success OR failure OR case study)
QS4d	Ágil e Tradicional	**QPd AND** (result OR experience OR history OR consequence OR outcome OR knowledge OR conclusion OR effect OR solution OR findings OR lesson OR success OR failure OR case study)

Tabela 4 – *Strings* da Questão Secundária 4

3.11.6 Questão Secundária 5 (QS5)

As *strings* utilizadas para a questão secundária 5 são listadas pela Tabela 5:

QS5	Processos	*String*
QS5a	Tradicional Ágil e Livre	**QPa AND** (model OR framework)
QS5b	Livre e Ágil	**QPb AND** (model OR framework)
QS5c	Livre e Tradicional	**QPc AND** (model OR framework)
QS5d	Ágil e Tradicional	**QPd AND** (model OR framework)

Tabela 5 – *Strings* da Questão Secundária 5

4. Execução da Revisão *Quasi*-Sistemática

Após o término da fase de planejamento da revisão *quasi*-sistemática, o protocolo foi estabelecido e a execução da revisão foi iniciada. Esta seção descreve em detalhes como a revisão *quasi*-sistemática foi conduzida. As buscas foram realizadas utilizando máquinas de busca de editoras ou bibliotecas digitais disponíveis no portal CAPES.

4.1. Execução das buscas das questões principais

A primeira rodada das buscas nas bibliotecas digitais se concentrou nas questões principais e foi realizada em cada uma das fontes selecionadas no período de 26/04/2009 a 27/04/2009.

4.1.1. Compendex

A busca foi efetuada com a opção *autostemming* ligada e utilizando como campo de busca o resumo. Devido à pouca quantidade de documentos obtidos com a *string* original, foram consideradas para busca apenas a população e intervenção, desconsiderando-se o resultado, e absorvendo-se o ônus de selecionar documentos manualmente. Assim, os resultados obtidos pela busca são resumidos pela Tabela 6. Como a máquina retornou artigos duplicados para a mesma *string*, estas duplicatas foram excluídas. Dentre os três artigos de controle definidos para a QPd, dois deles são indexados pela biblioteca Compendex e foram recuperados pela busca.

Questão	Quantidade Recuperada	Artigos de Controle
QPa	17	Não definidos
QPb	55	Não definidos
QPc	171	Não definidos
QPd	193	2 indexados 2 recuperados
TOTAL	**436**	

Tabela 6 – Resultados da busca na Compendex

4.1.2. IEEE

Devido à pouca quantidade de documentos obtidos com a *string* original, a *string* foi reformulada para ficar mais abrangente. Então foram consideradas para busca apenas a população e intervenção, desconsiderando-se o resultado. Para melhorar a qualidade dos resultados obtidos, resumidos pela Tabela 7, a busca foi feita utilizando-se o campo resumo. Dos dois artigos de controle indexados pela biblioteca apenas um foi recuperado.

Questão	Quantidade Recuperada	Artigos de Controle
QPa	4	Não definidos
QPb	10	Não definidos
QPc	36	Não definidos
QPd	100	2 indexados 1 recuperado
TOTAL	**150**	

Tabela 7 – Resultados da busca na IEEE

4.1.3. Scopus

No caso da questão QPa e QPb a *string* utilizada não incluiu os resultados, pois foram obtidos poucos artigos. Os resultados da busca realizada para as questões principais são resumidos pela Tabela 8. A busca foi feita nos campos título, resumo e palavra-chave. Os três artigos de controle indexados foram recuperados.

Questão	Quantidade Recuperada	Artigos de Controle
QPa	14	Não definidos
QPb	56	Não definidos
QPc	19	Não definidos
QPd	73	3 indexados 3 recuperados
TOTAL	**162**	

Tabela 8 – Resultados da busca na Scopus

4.1.4. Web of Science

A máquina de busca não conseguiu processar a *string* completa. Por limitações de implementação, esta máquina ficou restrita a 50 termos de busca e 49 operadores. Assim, a *string* utilizada não incluiu os resultados, limitando-se a população e intervenção. Nesta máquina, o campo escolhido para a busca foi o tópico (TS), pois ele inclui o título, resumo e palavras-chave. Utilizando os artigos de controle, percebeu-se que os resultados aumentavam caso os termos fossem colocados também no plural. Assim, as *strings* foram modificadas, principalmente em relação a uma simplificação dos termos da população, para contemplar o plural e, ao mesmo, atender a limitação de número de termos, conforme apresentado a seguir. Os resultados obtidos pela busca realizada para as questões principais são resumidos pela Tabela 9.

- **QPa:**

TS=((software OR development OR engineering OR project OR system OR organization OR enterprise OR corporation OR company OR industry OR firm OR community) AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR processes OR activity OR activities

OR method OR methods OR approach OR practice OR practices OR methodology OR methodologies OR technique OR techniques OR paradigm OR paradigms OR procedure OR procedures OR principle OR principles))

- **QPb:**

TS=((software OR development OR engineering OR project OR system OR organization OR enterprise OR corporation OR company OR industry OR firm OR community) AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (process OR processes OR activity OR activities OR method OR methods OR approach OR approaches OR practice OR practices OR methodology OR methodologies OR technique OR techniques OR paradigm OR paradigms OR procedure OR procedures OR principle OR principles))

- **QPc:**

TS=((software OR development OR engineering OR project OR system OR organization OR enterprise OR corporation OR company OR industry OR firm OR community) AND ("open source" OR "free source" OR libre OR OSS OR FOSS OR FLOSS) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR processes OR activity OR activities OR method OR methods OR approach OR approaches OR practice OR practices OR methodology OR methodologies OR technique OR techniques OR paradigm OR paradigms OR procedure OR procedures OR principle OR principles))

- **QPd:**

TS=((software OR development OR engineering OR project OR system OR organization OR enterprise OR corporation OR company OR industry OR firm OR community) AND (agile OR lightweight OR lean OR XP OR SCRUM) AND (conventional OR traditional OR "plan-driven" OR "plan driven" OR closed OR rigorous OR proprietary OR CMM OR CMMI) AND (process OR processes OR activity OR activities OR method OR methods OR approach OR approaches OR practice OR practices OR methodology OR methodologies OR technique OR techniques OR paradigm OR paradigms OR procedure OR procedures OR principle OR principles))

AND

TS=(combination OR adaptation OR conciliation OR reconciliation OR balance OR aggregation OR tailoring OR integration OR customization OR coexist OR juxtaposition OR compatibility)

Questão	Quantidade Recuperada	Artigos de Controle
QPa	4	Não definidos
QPb	30	Não definidos
QPc	184	Não definidos
QPd	112	2 indexados 2 recuperados
TOTAL	**330**	

Tabela 9 – Resultados da busca na Web Of Science

4.1.5. Consolidação dos resultados das buscas

Resumindo, a Tabela 10 apresenta a consolidação dos resultados das buscas realizadas para as questões principais nas quatro máquinas de buscas.

Biblioteca	QPa	QPb	QPc	QPd
Compendex	17	55	171	193
IEEE	4	10	36	100
Scopus	14	56	19	73
Web of Science	4	30	184	112
TOTAL	**39**	**151**	**410**	**478**

Tabela 10 – Resultados gerais da busca das questões principais

A partir dos resultados sumarizados pela Tabela 10, foi calculada a distribuição dos percentuais de trabalhos encontrados em relação às máquinas de busca. Através da Figura 2 é possível perceber que existe uma predominância de resultados vindos da biblioteca Compendex.

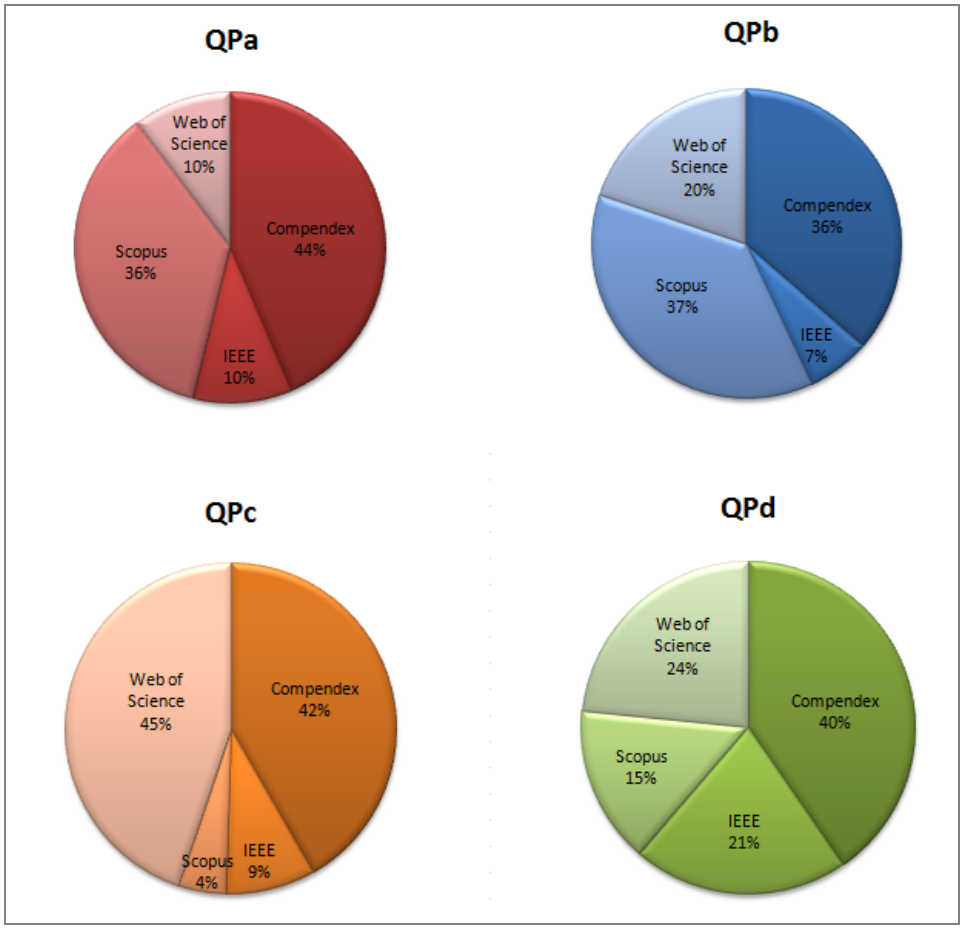


Figura 2 – Distribuição dos resultados pelas máquinas de buscas

4.2. Execução das buscas das questões secundárias

As buscas das questões secundárias foram realizadas a partir dos resultados encontrados para as questões principais, após a retirada das duplicações,

utilizando-se os campos título, resumo e palavras-chave. Os resultados das buscas das questões secundárias são sumarizados pela Tabela 11.

	QS1	QS2	QS3	QS4	QS5
Tradicional, Ágil e Livre (a)	10	2	7	14	10
Livre e Ágil (b)	27	6	31	48	42
Livre e Tradicional (c)	170	27	129	252	140
Ágil e Tradicional (d)	178	26	197	276	160
TOTAL	**385**	**61**	**364**	**590**	**352**

Tabela 11 – Resultados das buscas das questões secundárias

5. Resultados da Revisão *Quasi*-Sistemática

Nesta seção são detalhados os resultados obtidos com a revisão *quasi*-sistemática realizada. Os documentos recuperados foram analisados em relação as duplicatas existentes, a disponibilidade do texto completo do artigo e em relação aos critérios de inclusão e exclusão estabelecidos no protocolo.

5.1.Análise dos Documentos Recuperados

5.1.1. Eliminação de duplicatas

Como primeiro passo para a análise dos documentos recuperados, as repetições encontradas nas buscas das questões principais foram eliminadas, mantendo-se o artigo remanescente contabilizado para a biblioteca digital com maior quantidade de itens recuperados. A nova situação quantitativa, das questões principais, após eliminação de repetições, é apresentada pela Tabela 12:

	QPa			QPb			QPc			QPd		
Biblioteca	**AVA**	**%D**	**SEL**	**AVA**	**%D**	**SEL**	**AVA**	**%D**	**SEL**	**AVA**	**%D**	**SEL**
Compendex	17	0%	17	55	83%	9	171	19%	137	193	0%	193
IEEE	4	100%	0	10	60%	4	36	80%	7	100	36%	64
Scopus	14	78%	3	56	0%	56	19	63%	7	73	73%	19
Web of Science	4	50%	2	30	43%	17	184	0%	184	112	11%	99
TOTAL	**39**	**43%**	**22**	**151**	**43%**	**86**	**410**	**18%**	**335**	**478**	**21%**	**375**

Tabela 12 – Resultados das buscas das questões principais sem duplicatas

Legenda: AVA – Número de artigos avaliados %D – % de duplicatas excluídas SEL – Número de artigos selecionados inicialmente

Do total geral de 1.078 artigos analisados, foram excluídas 260 repetições, que correspondem a 24% do total, e foram selecionados, para serem analisados na próxima fase, 818 artigos (Figura 3). Este percentual de repetições é reflexo das superposições de trabalhos indexados pelas diferentes bibliotecas digitais. Em especial, observou-se que a base de dados da IEEE possui muitas duplicações em relação a Scopus e Compendex. No entanto, para garantir uma maior cobertura dos

resultados, todas as máquinas de busca foram mantidas, pois havia artigos indexados por apenas uma delas.

Figura 3 – Gráfico de análise do percentual geral de duplicações encontradas

5.1.2. Primeiro Filtro – Título e Resumo

Após a eliminação das duplicatas, foi feita uma primeira avaliação, a partir da leitura de todos os títulos e resumos, aplicando-se os critérios de exclusão, foram excluídas as referências que nitidamente tratavam de outros assuntos não pertinentes à pesquisa. Após eliminação de tais itens, a nova situação quantitativa é resumida pela Tabela 13, onde pode-se observar que, proporcionalmente, a máquina da Web of Science foi a que retornou resultados menos relevantes:

	QPa			QPb			QPc			QPd		
Biblioteca	**AVA**	**%EF1**	**SEL**	**AVA**	**%EF1**	**SEL**	**AVA**	**%EF1**	**SEL**	**AVA**	**%EF1**	**SEL**
Compendex	17	64%	6	9	77%	2	137	69%	42	193	67%	63
IEEE	0	0%	0	4	75%	1	7	42%	4	64	65%	22
Scopus	3	33%	2	56	71%	16	7	57%	3	19	78%	4
Web of Science	2	100%	0	17	94%	1	184	79%	37	99	94%	5
TOTAL	**22**	**63%**	**8**	**86**	**76%**	**20**	**335**	**74%**	**86**	**375**	**74%**	**94**

Tabela 13 – Resultados das buscas após o primeiro filtro

Legenda: AVA - Número de artigos avaliados %EF1 - % de artigos excluídos pelo 1º. filtro SEL – Número de artigos selecionados para a próxima fase

Nesta fase, do total geral de 818 artigos analisados, foram excluídos 610 artigos e foram selecionados, para serem analisados na próxima fase, 208 artigos que correspondem a 25% dos trabalhos analisados nessa fase (Figura 4).

Figura 4 – Gráfico de Análise do Percentual Geral do Primeiro Filtro

Em relação às questões secundárias, os resultados após a aplicação do primeiro filtro são resumidos pela Tabela 14.

	QS1			QS2			QS3			QS4			QS5		
	AVA	%EF1	SEL	AVA	%EF1	SEL	AVA	%EF1	SEL	AVA	%EF1	SEL	AVA	%EF1	SEL
Tradicional, Ágil e Livre (a)	10	60 %	4	2	50 %	1	7	57 %	3	14	64 %	5	10	60 %	4
Livre e Ágil (b)	27	70 %	8	6	83 %	1	31	80 %	6	48	75 %	12	42	80 %	8
Livre e Tradicional (c)	170	70 %	51	27	70 %	8	129	79 %	27	252	76 %	60	140	70 %	42
Ágil e Tradicional (d)	178	70 %	53	26	30 %	18	197	76 %	47	276	75 %	69	160	74 %	41
TOTAL	**385**	**69 %**	**116**	**61**	**54 %**	**28**	**364**	**77 %**	**83**	**590**	**75 %**	**146**	**352**	**37 %**	**95**

Tabela 14 – Resultados das buscas das questões secundárias após o primeiro filtro
Legenda: AVA - Número de artigos avaliados %EF1 - % de artigos excluídos pelo 1º. filtro SEL – Número de artigos selecionados para a próxima fase

Após a execução do primeiro filtro dos artigos, a lista de artigos excluídos foi revista, por amostragem, por 4 outros pesquisadores. O objetivo desta validação é evitar que artigos sejam excluídos indevidamente e que assim se percam trabalhos relevantes para a pesquisa. Então, em caso de impasse entre os pesquisadores ou dúvida, a publicação foi incluída na lista de selecionadas. Os resultados apresentados acima já foram consolidados após esta validação.

5.1.3. Segundo Filtro - Conteúdo

Para que possa ser feita uma avaliação mais apurada e detalhada, é necessária a leitura do texto completo dos artigos. Assim, o primeiro passo para a realização deste segundo filtro é a recuperação dos arquivos com os textos completos dos artigos. Entretanto, não foi possível obter acesso a todas as referências, como demonstra a Tabela 15, mesmo entrando em contato com todos os autores por e-mail.

	QPa			QPb			QPc			QPd		
Biblioteca	**AVA**	**TCND**	**SEL**	**AVA**	**TCND**	**SEL**	**AVA**	**TCND**	**SEL**	**AVA**	**TCND**	**SEL**
Compendex	6	2	4	2	0	2	42	5	37	63	7	56
IEEE	0	0	0	1	0	1	4	0	4	22	0	22
Scopus	2	1	1	16	3	13	3	0	3	4	1	3
Web of Science	0	0	0	1	0	1	37	6	31	5	1	4
TOTAL	**8**	**3**	**5**	**20**	**3**	**17**	**86**	**11**	**75**	**94**	**9**	**85**

Tabela 15 – Disponibilidade dos textos completos dos artigos

Legenda: AVA - Número de artigos avaliados TCNC – Número de artigos com o texto completo não disponível SEL – Número de artigos selecionados

Do total de 208 artigos que deveriam ser avaliados nesta fase, 182 ou 88% tiveram o texto completo recuperado, através das máquinas de buscas ou de

contato por e-mail com os respectivos autores (Figura 5). Assim, no total não foi possível ter acesso ao texto completo de apenas 26 referências.

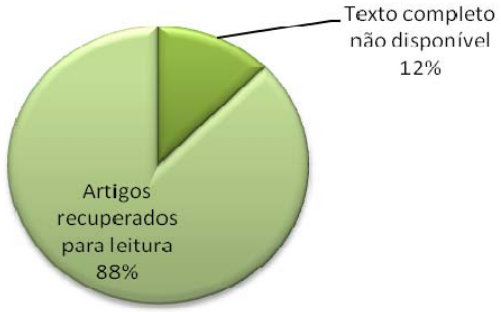


Figura 5 – Gráfico de análise do percentual geral de artigos com texto completo recuperado

Os artigos recuperados foram lidos pelo pesquisador, visando selecionar os candidatos a fazer parte da revisão sistemática (Tabela 16). Para todos os artigos lidos foi feita uma nova análise dos critérios de inclusão e exclusão.

	QPa			QPb			QPc			QPd		
Biblioteca	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**
Compendex	4	75%	1	2	100%	0	37	89%	4	56	69%	17
IEEE	0	0%	0	1	100%	0	4	75%	1	22	68%	7
Scopus	1	100%	0	13	76%	3	3	100%	0	3	66%	1
Web of Science	0	0%	0	1	0%	1	31	80%	6	4	25%	3
TOTAL	**5**	**80%**	**1**	**17**	**76%**	**4**	**75**	**85%**	**11**	**85**	**67%**	**28**

Tabela 16 – Resultados das buscas após o segundo filtro

Legenda: AVA - Número de artigos avaliados %EF2 - % de artigos excluídos pelo 2º. Filtro SEL – Número de artigos selecionados

Em relação às questões secundárias, os resultados após a aplicação do segundo filtro são resumidos pela Tabela 17.

	QS1			QS2			QS3			QS4			QS5		
	AVA	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**	**AVA**	**%EF2**	**SEL**
Tradicional, Ágil e Livre (a)	4	75 %	1	1	0%	1	3	100 %	0	5	100 %	0	4	100 %	0
Livre e Ágil (b)	8	87 %	1	1	100 %	0	6	100 %	0	12	915 %	1	8	100 %	0
Livre e Tradicional (c)	51	96 %	2	8	50 %	4	27	96 %	1	60	100 %	0	42	95 %	2
Ágil e Tradicional (d)	53	62 %	5	18	11 %	16	47	91 %	4	69	89 %	7	41	97 %	1
TOTAL	**116**	**93 %**	**9**	**28**	**25 %**	**21**	**83**	**93 %**	**5**	**146**	**94 %**	**8**	**95**	**96 %**	**3**

Tabela 17 – Resultados das buscas das questões secundárias após o segundo filtro

Legenda: AVA - Número de artigos avaliados %EF2 - % de artigos excluídos pelo 2º. filtro SEL – Número de artigos selecionados para a próxima fase

A lista completa dos artigos analisados neste segundo filtro, destacando os artigos efetivamente selecionados em cada questão de pesquisa, encontra-se detalhada nos Apêndices A, B, C e D respectivamente.

5.1.4. Instrumentos

Todas as referências recuperadas foram importadas para o JabRef[3] que foi o gerenciador de referências utilizado para manipular as referências recuperadas pelas máquinas de busca. A ferramenta JabRef foi útil para identificar repetições de referências. Além disso, permitiu a criação de campos customizados para registrar as informações extraídas, manter o controle dos dados de origem e acompanhar a decisão tomada em cada fase do processo de filtragem (Figura 6).

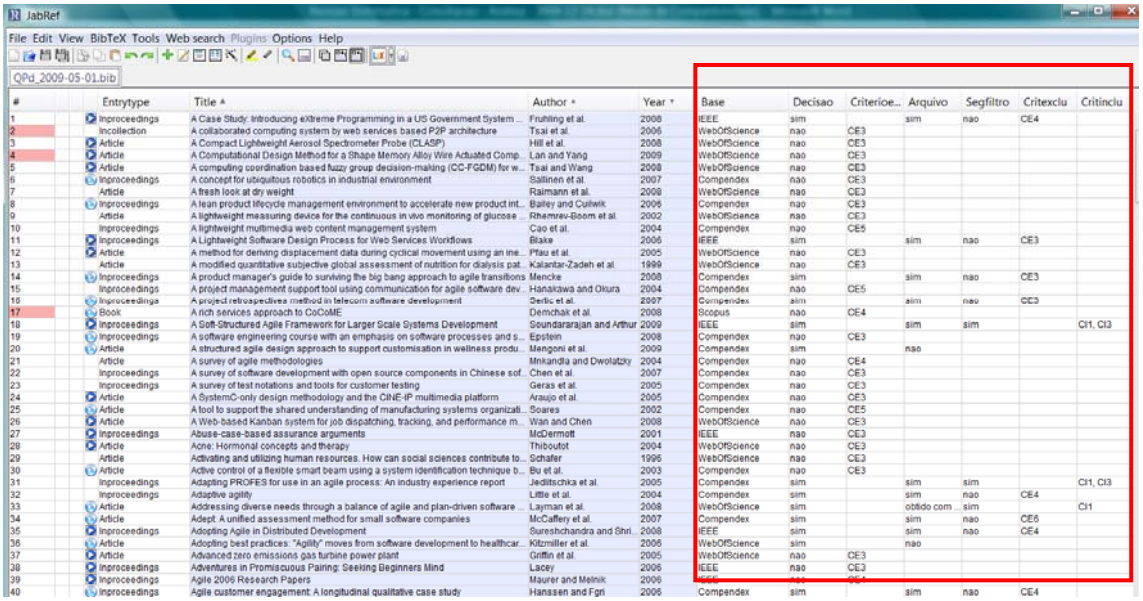


Figura 6 – Campos customizados no JabRef

O JabRef também foi utilizado para executar a busca das questões secundárias dentro dos resultados obtidos com as principais questões principais. Por último, a ferramenta também oferece a possibilidade de criar relatórios específicos para exportar todas essas informações em um formato pré-definido para facilitar o trabalho de manipulação das referências e extração e análise das informações (Figura 7).

QPd - Segundo Filtro							
#	Título	Base	Arquivo	2o. Filtro	Critério Exclusão	Critério Inclusão	Comentário
1.	A Case Study: Introducing eXtreme Programming in a US Government System Development Project	IEEE	sim	nao	CE4		O objetivo deste artigo é entender como o XP pode ser introduzido em uma organização governamental norte-americana que historicamente adotou o desenvolvimento tradicional.
2.	A Lightweight Software Design Process for Web Services Workflows	IEEE	sim	nao	CE3		Agilidade na composição de serviços web.
3.	A product manager's guide to surviving the big bang approach to agile transitions	Compendex	sim	nao	CE3		O artigo descreve o resultado da introdução de uma metodologia ágil na SalesForce.com. Trata-se de uma transição para uma metodologia ágil e não de uma conciliação.
4.	A project retrospectives method in telecom software development	Compendex	sim	nao	CE3		Nem o tradicional e nem o ágil atendem as necessidades do software de telecomunicações e os autores propõem uma metodologia baseada em retrospectiva de projeto.
5.	A Soft-Structured Agile Framework for Larger Scale Systems Development	IEEE	sim	sim		CI1, CI3	Propõe um framework híbrido que combina práticas ágeis e tradicionais para o desenvolvimento de sistemas em larga escala, com foco na mudança de requisitos.
6.	A structured agile design approach to support customisation in wellness product development	Compendex	nao				
7.	Adapting PROFES for use in an agile process: An industry experience report	Compendex	sim	sim		CI1, CI3	Adaptação da metodologia PROFES (orientada a sistemas embarcados e com foco na qualidade do produto) de melhoria para uso em um contexto industrial com um método ágil. Estudo de caso na BMW.
8.	Adaptive agility	Compendex	sim	sim		CI3	Na Landmark foi construída uma metodologia que classifica os projetos de acordo com a complexidade e incerteza para adaptar as práticas do processo ágil de acordo com o projeto.
9.	Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course	WebOfScience	obtido com autores	sim		CI1	O artigo descreve um curso de graduação onde os alunos aprendem tanto o processo tradicional quanto o ágil. Os alunos aprendem a aplicar as técnicas adequadas dependendo do projeto ou da equipe.
10.	Adept: A unified assessment method for small software companies	Compendex	sim	sim		CI1, CI3	Adept é um método de avaliação de processo que combina tradicional e ágil e favorece o seu uso em pequenas empresas.
11.	Adopting Agile in Distributed Development	IEEE	sim	nao	CE4		Uso de métodos ágeis em um cenário de desenvolvimento distribuído onde a equipe não está toda junta. Propõe um modelo para fazer esta transição e as lições aprendidas de uma organização que já passou por isso.

Figura 7 – Exemplo de relatório gerado para análise das informações do 2º. filtro

[3] Site JabRef: http://jabref.sourceforge.net/

5.2.Extração de Informações

A fase de extração de informações permite que sejam coletadas todas as informações necessárias para responder as questões de pesquisa. As informações capturadas dos trabalhos selecionados foram tabeladas, em ordem cronológica, e agrupadas de acordo com as questões de pesquisa que elas endereçam.

5.2.1. QPa

As informações extraídas do artigo selecionado da QPa são apresentadas pela Tabela 18.

Informações da QPa		
1.	**Título do documento:**	Corporate-, Agile- and Open Source Software development: A witch's brew or an elixir of life?
	Autor(es):	M. Theunissen; D. Kourie; A. Boake
	Data de publicação:	2008
	Fonte:	Balancing Agility and Formalism in Software Engineering – LNCS – Springer
	Processos de desenvolvimento de software:	Ágil e Livre
	Práticas:	Não se aplica
	Critérios de comparação:	Comparação pela aderência aos princípios do Manifesto Ágil: • **Similaridades:** entrega cedo do software. • **Diferenças:** equipes ágeis ficam localizadas fisicamente próximas, enquanto no software livre as equipes são geograficamente distribuídas; no desenvolvimento ágil, o papel do cliente é encarnado por usuários do negócio, mas no software livre o papel do cliente é encarnado pelos próprios programadores que são usuários do software. O software livre é compatível com alguns princípios do Manifesto Ágil, mas incompatível com outros. Então, existem diferenças significativas entre as duas abordagens e o software livre não pode ser considerado apenas mais uma instância dos métodos ágeis. Como resultado destas diferenças, o escopo de sinergia entre o desenvolvimento ágil e o desenvolvimento livre é limitado e tendem a aparecer tensões entre as equipes que seguem princípios distintos.
	Informações da QS1	
	Oportunidades para a conciliação:	Não se aplica
	Desafios para a conciliação:	Dificuldades que podem ser encontradas diante da tentativa de uso do software livre dentro de uma cultura corporativa: • **Monitoramento dos desenvolvedores:** em um ambiente onde as pessoas são remuneradas pelo seu trabalho, existe a necessidade de monitorar os funcionários. Entretanto, esta prática não funciona diante do caráter voluntário do desenvolvimento de software livre. Além disso, fica difícil monitorar como os desenvolvedores estão contribuindo para os diferentes projetos de software livre e avaliar a importância destas contribuições para os interesses da organização. • **Definição de cronograma:** os projetos de software livre, em geral, adotam o princípio de lançamento frequente de versões, mas sem um planejamento formal, ou seja, as versões são liberadas no momento que os responsáveis do projeto acham adequado. Em um ambiente corporativo, existe a necessidade dos projetos trabalharem com prazos para atender determinados objetivos de negócio, aproveitar oportunidades de mercado e atender restrições financeiras. • **Processo de garantia de qualidade:** o desenvolvimento de software livre encoraja a revisão por pares, mas não existe um processo formal de revisão de código, o que contrasta com o paradigma ágil e outras técnicas de garantia de qualidade tradicionais da engenharia de software. • **Aspectos legais das licenças de software livre:** os desenvolvedores corporativos geralmente não estão familiarizados com a diversidade de licenças de software livre e seus inter-relacionamentos. Assim, se existir, dentro da mesma organização, desenvolvimento de software

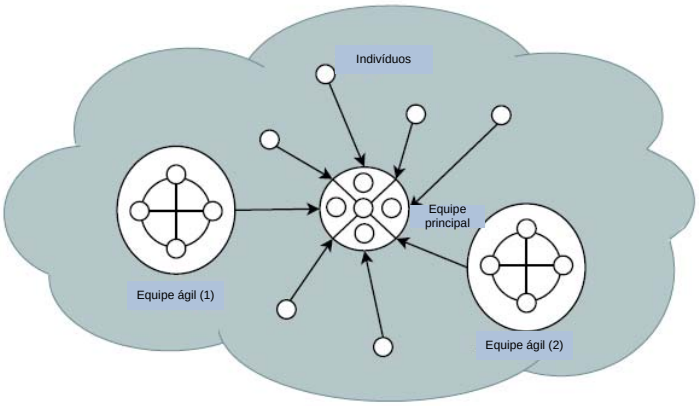
Informações da QPa	
	livre e proprietário, então eles devem ser isolados para evitar possíveis contaminações do código que poderiam trazer riscos legais à empresa. Desafios para a conciliação do desenvolvimento ágil e livre, onde uma equipe ágil dentro da organização colabora com um projeto de software livre externo (Figura 8): • **Comunicação:** do ponto de vista ágil, talvez o maior desafio seja a adaptação a uma forma diferente de comunicação entre os desenvolvedores, visto que o desenvolvimento ágil se baseia na comunicação verbal face-a-face. Já o desenvolvimento de software livre se baseia na comunicação através da Internet, o que permite um fluxo de informação contínuo que se torna extremamente perturbador ao desenvolvimento ágil. Além disso, a comunicação verbal da equipe centralizada precisa ser transmitida aos outros desenvolvedores que estão distantes. • **Controle:** os desenvolvedores ágeis estão acostumados a participar da tomada de decisões do projeto. Entretanto, quando a equipe é parte de uma comunidade de desenvolvedores maior, este controle sobre o projeto é perdido. Por exemplo, a decisão sobre o lançamento de versões passa a não ser uma decisão apenas da equipe ágil. • **Participação na comunidade:** como parte de uma comunidade com uma cultura diferente, os desenvolvedores ágeis precisam estender esta cultura e conhecer as suas regras implícitas de participação.  Figura 8 – Equipe ágil colaborando com um projeto de software livre
Informações da QS2	
Descrição da proposta de conciliação:	Para resolver alguns dos desafios citados, os autores endossam a noção da definição de um processo para o projeto. Este processo do projeto não depende apenas dos subprocessos internos aos projetos, mas também precisa levar em consideração os processos dos projetos de software livre externo. Propostas para a conciliação: • **Pontos de interação:** em geral, vão existir pontos de interação entre os subprocessos internos ao projeto e os subprocessos dos projetos de software livre externos. Os desafios citados anteriormente vão estar presentes nestes pontos de interação. Para começar a lidar com estas tensões é importante articular (formal ou informalmente) um subprocesso em cada ponto de interação. Com essa abordagem é possível especificar processos de desenvolvimento corporativos usando os projetos externos apenas como *plug-ins* encapsulando os processos. • **Papel adicional:** este novo papel funciona como um canal de comunicação entre os projetos. Este papel é o responsável por manter o entendimento sobre o processo e a evolução dos artefatos do processo externo e por disseminar este conhecimento pelo resto da equipe.

Tabela 18 – Informações extraídas do artigo da QPa

5.2.2. QPb

As informações extraídas dos artigos selecionados da QPb são apresentadas pela Tabela 19.

Informações da QPb		
2.	**Título do documento:**	Continuous integration in open source software development
	Autor(es):	A. Deshpande; D. Riehle
	Data de publicação:	2008
	Fonte:	Open Source Development, Communities and Quality – Springer
	Processos de desenvolvimento de software:	Ágil e Livre
	Práticas:	Integração contínua de código (os desenvolvedores contribuem com código para o projeto frequentemente e em pequenos incrementos, sempre garantindo que o projeto continua compilando e passando nos testes). A prática de integração contínua de código é geralmente associada ao desenvolvimento ágil. Para verificar a hipótese de que o processo de desenvolvimento de software livre é similar ao processo de desenvolvimento ágil, os autores analisaram dados de 500 projetos de software livre ativos para verificar o funcionamento da integração de código. Os resultados indicaram que o desenvolvimento de software livre não mudou a sua forma de integração de código, ou seja, não foi influenciado significativamente pelo advento desta nova prática ágil. Uma explicação para isso é que talvez a integração contínua de código já seja adotada nos projetos de software livre a mais tempo.
	Critérios de comparação:	Não se aplica
3.	**Título do documento:**	Sprinting toward open source development
	Autor(es):	G. Goth
	Data de publicação:	2007
	Fonte:	IEEE Software
	Processos de desenvolvimento de software:	Ágil e Livre
	Práticas:	Desenvolvimento orientado a *sprint* (um grupo de desenvolvedores se reúne em um local e passa dois ou três dias trabalhando no desenvolvimento de uma parte do software).
	Critérios de comparação:	As práticas do desenvolvimento ágil estão tão alinhadas às práticas do desenvolvimento livre que é possível afirmar que o desenvolvimento ágil se situa no meio do caminho entre o desenvolvimento de software tradicional e o desenvolvimento livre.
	Informações da QS1	
	Oportunidades para a conciliação:	O desenvolvimento face-a-face e o desenvolvimento de software livre não são mutuamente exclusivos e a combinação dos dois pode trazer resultados muito positivos. A adoção da prática de desenvolvimento orientado a *sprint* se mostrou benéfica para: • Envolver as pessoas no processo de desenvolvimento; • Construir a comunidade, permitindo que as pessoas se conheçam, se encontrem e colaborem pessoalmente para facilitar a futura colaboração remota; • Completar um trabalho importante.
	Desafios para a conciliação:	Não se aplica
	Informações da QS4	
	Experiência das organizações:	O projeto de software livre PyPy estabeleceu uma abordagem híbrida de desenvolvimento de software, inspirada no modelo da Zope Corporation, onde adota a prática de desenvolvimento orientado a *sprint*, geralmente

Informações da QPb		
		associada ao desenvolvimento ágil, para resolver o problema do desenvolvimento distribuído.
4.	**Título do documento:**	Modeling and simulation of open source development using an agile practice
	Autor(es):	I. Turnu; M. Melis; A. Cau; A. Setzu; G. Concas; K. Mannaro
	Data de publicação:	2006
	Fonte:	Journal of Systems Architecture
	Processos de desenvolvimento de software:	Ágil e Livre
	Práticas:	Desenvolvimento orientado a testes (*Test Driven Development* – TDD) Este trabalho estuda os efeitos do desenvolvimento orientado a testes no desenvolvimento de software livre, pois esta é considerada uma prática ágil mais fácil de ser aplicada em um ambiente geograficamente distribuído. Este estudo foi feito através de modelos de simulação que foram calibrados com dados do projeto Apache. Os resultados indicaram que projetos de software livre que adotam esta prática ágil conseguem melhores resultados em relação a qualidade do código.
	Critérios de comparação:	Não se aplica
5.	**Título do documento:**	Agile principles and open source software development: A theoretical and empirical discussion
	Autor(es):	S. Koch
	Data de publicação:	2004
	Fonte:	Extreme Programming and Agile Processes in Software Engineering – LNCS – Springer
	Processos de desenvolvimento de software:	Ágil e Livre
	Práticas:	Não se aplica
	Critérios de comparação:	A comparação entre o desenvolvimento ágil e livre encontrou similaridades impressionantes entre os dois modelos de desenvolvimento. Os dados empíricos confirmaram, pelo menos parcialmente, a existência de princípios ágeis no desenvolvimento de software livre. Por outro lado, uma diferença significativa encontrada é a necessidade de contato pessoal e proximidade física da equipe de desenvolvimento, demandada pelo desenvolvimento ágil, e que não se observa no desenvolvimento livre. • **Processo de desenvolvimento de software:** nem o ágil e nem o software livre possuem uma descrição do processo de desenvolvimento de software. No máximo, eles possuem um conjunto de princípios para nortear um projeto. • **Foco nos indivíduos**: tanto o desenvolvimento ágil quanto o software livre focam nas competências e motivações individuais. • **Tamanho da equipe:** devido à necessidade de coordenar pessoalmente o trabalho, a maioria dos autores sugere que uma equipe ágil deva ser composta, no máximo, por 15 ou 20 pessoas. Mesmo não tendo uma restrição de tamanho para as equipes, a maioria dos projetos de software livre também trabalha com uma equipe de desenvolvedores principais de tamanho similar ou menor que o indicado para um projeto ágil. • **Auto-organização da equipe:** o desenvolvimento ágil enfatiza a importância das equipes se auto-organizarem para que possam se adaptar rapidamente às mudanças de requisitos. Isso requer um foco comum, confiança e respeito mútuos e uma intensa colaboração. No software livre, o foco comum é garantido pelo interesse dos desenvolvedores que decidiram participar do projeto. Além disso, como não existe uma coordenação centralizada e cada participante contribui espontaneamente, cada um acaba selecionando aquelas tarefas onde pode trabalhar com mais eficiência, o que leva a uma auto-organização natural da equipe. • **Proximidade da equipe de desenvolvimento:** o desenvolvimento

Informações da QPb		
		ágil se baseia no contato pessoal e na colaboração entre os membros da equipe de desenvolvimento que se encontram fisicamente reunidos. Porém, no software livre o desenvolvimento é realizado por uma equipe geograficamente distribuída e que se comunica através da Internet. • **Interação com cliente:** no desenvolvimento ágil existe uma interação e colaboração forte com o cliente que chega a fazer parte da equipe de desenvolvimento como um especialista do negócio. No desenvolvimento de software livre, o cliente também pode se juntar a equipe de desenvolvimento, mas com um papel de co-desenvolvedor para ajudar a melhorar o software mais rapidamente. • **Entrega cedo do software:** o desenvolvimento ágil enxerga o software funcionando com uma medida do progresso do andamento do projeto. Para usar essa medida na colaboração com o cliente e para garantir ciclos de *feedback* curtos, são lançadas versões frequentes. Esta prática aumenta a motivação de todos os participantes, permitindo a discussão do status do projeto e aumentando as chances de descobrir as mudanças necessárias. O desenvolvimento de software livre também trabalha com o lançamento frequente de versões para que os usuários e desenvolvedores possam testar e depurar o código para encontrar e corrigir os erros mais rapidamente. Essa prática também ajuda a manter os desenvolvedores estimulados, pois eles se sentem recompensados ao ver o resultado do seu trabalho ganhar forma no software. • **Mudança de requisitos:** o desenvolvimento ágil se preocupa em acomodar as mudanças de requisitos que surgem durante o desenvolvimento do software. Por isso, adota uma arquitetura simples que facilite as mudanças e permita a refatoração do código. O mesmo acontece no desenvolvimento de software livre.

Tabela 19 – Informações extraídas dos artigos da QPb

5.2.3. QPc

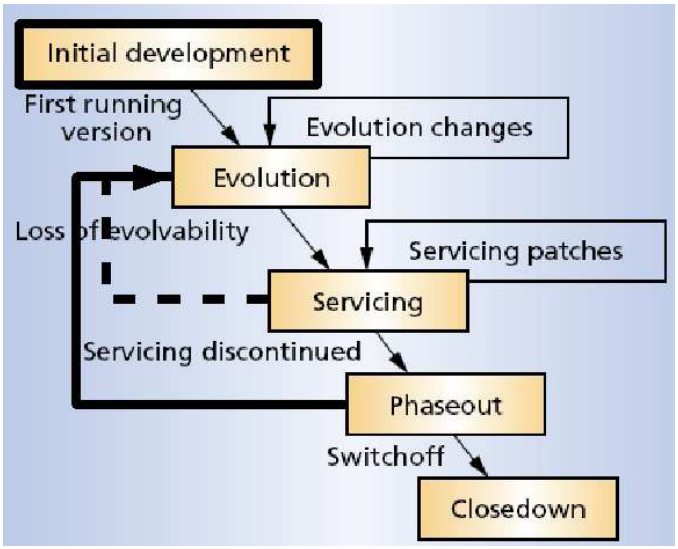
As informações extraídas dos artigos da QPc são apresentadas pela Tabela 20.

Informações da QPc		
6.	**Título do documento:**	Open Collaboration within Corporations Using Software Forges
	Autor(es):	D. Riehle; J. Ellenberger; T. Menahem; B. Mikhailovski; Y. Natchetoi; B. Naveh; T. Odenwald
	Data de publicação:	2009
	Fonte:	Software, IEEE
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	Os seguintes princípios da colaboração estão presentes no desenvolvimento de software livre: • **Igualitarismo:** os projetos de software livre estão disponíveis através da Internet e tipicamente incluem todos que desejem contribuir e ajudar. • **Meritocracia:** todas as contribuições são avaliadas de forma transparente com base no seu mérito e as discussões são conduzidas publicamente nas listas de discussões. • **Auto-organização:** nenhum processo definido é imposto e a própria comunidade determina como irá se organizar para conduzir o trabalho. Estes princípios contrastam com a gestão dos processos de desenvolvimento de software internos das organizações: • **Tarefas designadas:** alocação dos recursos nos projetos e nas tarefas designada pelos níveis superiores. • **Posição:** a hierarquia entre os papéis presentes no projeto implica em status e determina quem toma a decisão final em relação ao projeto e implementação do software. • **Processos impostos:** o processo é definido pela organização e imposto a todos os projetos de desenvolvimento de software.

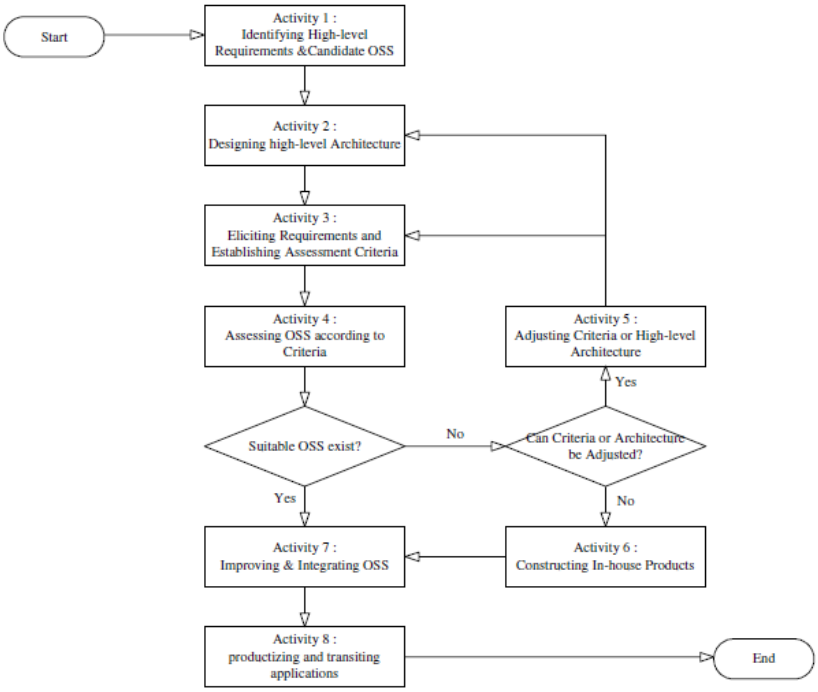
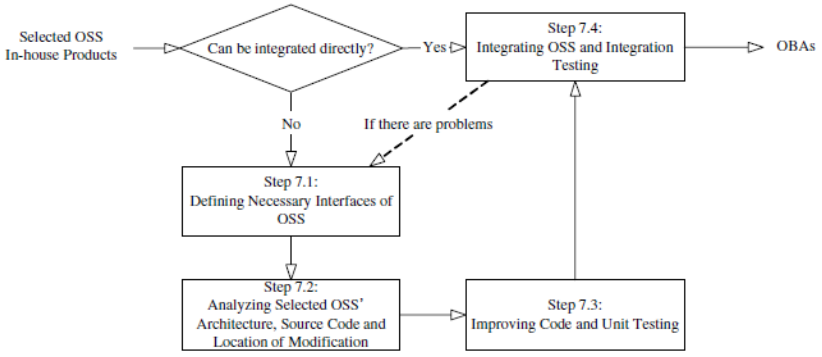
Informações da QPc		
	Informações da QS3	
	Estratégias adotadas pelas organizações:	Alguns grandes vendedores de software, como HP, IBM, Microsoft e SAP, já tomaram ações para incorporar as boas práticas do desenvolvimento de software livre no seu processo de desenvolvimento de software corporativo. Todavia, um dos principais problemas encontrados é que as equipes internas de projetos diferentes usam ferramentas diferentes e até incompatíveis dentro da mesma organização. Isso dificulta as contribuições espontâneas e voluntárias dos desenvolvedores nos projetos. Assim, este artigo foi motivado pelo desejo de entender como o desenvolvimento de software comercial pode se beneficiar das melhores práticas existentes no software livre, se estas práticas também funcionariam nas organizações e como elas podem ser transferidas do contexto das comunidades de software livre para o contexto das organizações. Utilizando práticas colaborativas de desenvolvimento de software livre na SAP, os autores observaram que elas podem complementar o desenvolvimento de software tradicional. Uma forma de facilitar a adoção das práticas de software livre nas organizações é a adoção de software *forge*. O software *forge* emergiu da Internet e os desenvolvedores são os seus usuários principais. Um software *forge* é baseado em uma plataforma web extensível que integra as ferramentas colaborativas de desenvolvimento de software. O SourceForge é o exemplo mais conhecido de software *forge* na Internet, pois hospeda uma grande coleção de projetos de software livre. Outros exemplos são BerliOS, Codehaus e Tigris. O software *forge* tem duas visões principais: • Uma **visão de portfólio de projetos** que permite aos desenvolvedores navegar e encontrar um projeto. • Uma **visão do projeto** que oferece ao desenvolvedor um conjunto de ferramentas para trabalhar em um projeto específico. Um software *forge* geralmente inclui aplicações de gestão de tarefas, gestão de defeitos, lista de discussão, gerência de configuração de software e de documentação. Um software *forge* apóia todo o processo de desenvolvimento de software, unifica as ferramentas utilizadas em todas as atividades do processo e integra todas elas em uma única interface facilitando a navegação. Como todos os projetos utilizam as mesmas ferramentas, os desenvolvedores conseguem mais facilmente colaborar em mais de um projeto ou mudar de projeto. A principal diferença entre um software *forge* e uma ferramenta CASE é o seu design centrado em colaboração que facilita a localização de um projeto, o seu entendimento e as contribuições dos voluntários. Entre outros benefícios, um software *forge* centraliza e armazena informações importantes e diminui os gastos com tarefas de administração e manutenção dos ambientes. Com o estímulo aos princípios da colaboração e o apoio de um software *forge*, desenvolvedores entusiasmados e motivados podem dedicar o seu esforço aos projetos internos da organização ao invés de projetos de software livre externos.
7.	**Título do documento:**	Towards a process maturity model for open source software
	Autor(es):	M. Ciolkowski; M. Soto
	Data de publicação:	2008
	Fonte:	Annual IEEE International Computer Software and Applications Conference (COMPSAC)
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS5	
	Modelo de processos com conciliação:	No desenvolvimento tradicional de software, modelos de maturidade de processo, como o CMMI, vêm sendo usados para determinar se uma organização tem condições de entregar o software no prazo e com um nível de qualidade aceitável.

Informações da QPc		
		Contudo, estes modelos são percebidos como inadequados ao desenvolvimento de software livre, pois se acredita que as comunidades de software livre operam de modo caótico e, por isso, nenhum processo de desenvolvimento sistemático poderia ser utilizado. Na prática, as comunidades de software livre são bem organizadas, aplicam algumas boas práticas e existem evidências de maturidade de processos nos projetos de software livre, tendo em vista a qualidade dos produtos gerados. Por outro lado, os modelos de qualidade atualmente existentes realmente não podem ser aplicados diretamente ao software livre porque eles possuem muitos elementos específicos das organizações tradicionais. Assim, os autores acreditam que é possível reutilizar e adaptar os modelos de maturidade existentes (CMMI e SPICE) para as características específicas do software livre. A partir desta ideia, é apresentada uma proposta de modelo de maturidade para software livre, ainda em desenvolvimento, onde se destacam algumas práticas comuns aos modelos de maturidade e ao software livre: • Gerência de configuração de software; • Gerência de *releases* de software; • Análise de requisitos. Os benefícios de um modelo de avaliação de processos para software livre são: • Melhor fundamentação para avaliar a capacidade de uma comunidade de software livre de produzir software de qualidade e a sua sustentabilidade a longo prazo; • Facilitar a adoção do software livre em algumas indústrias que têm necessidade de avaliar os seus fornecedores. Outros modelos de avaliação de projetos de software livre citados: • *Qualification and Selection of Open Source* (QSOS); • *Cap Gemini`s Open Source Maturity Model* (OSMM); • *Navica`s Open Source Maturity Model* (OSMM); • *Business Readiness Rating* (OpenBRR). Estes modelos levam em consideração o produto e a comunidade, mas têm uma perspectiva de processo rudimentar.
8.	**Título do documento:**	Achieving quality in open-source software
	Autor(es):	M. Aberdour
	Data de publicação:	2007
	Fonte:	IEEE Software
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Garantia da qualidade de software e Controle da qualidade de software Através da revisão do corpo de conhecimento existente na literatura, o autor investiga as diferentes abordagens relativas a qualidade do software adotadas no desenvolvimento de software livre e que podem ser incorporadas ao desenvolvimento tradicional.
	Critérios de comparação:	Desenvolvimento tradicional: • Metodologia de desenvolvimento bem definida; • Documentação extensiva; • Metodologia de garantia da qualidade e testes formais e estruturados; • Requisitos definidos por analistas; • Processo de avaliação formal de riscos; • Objetivos mensuráveis utilizados ao longo do projeto; • Descoberta de defeitos através de testes caixa-preta assim que possível no processo; • Evidência empírica referente a qualidade utilizada rotineiramente para apoiar a tomada de decisão; • Os membros da equipe são designados para as tarefas; • Fase formal de design é realizada e aprovada antes de iniciar a programação; • Muito esforço colocado no planejamento do projeto. Desenvolvimento de software livre:

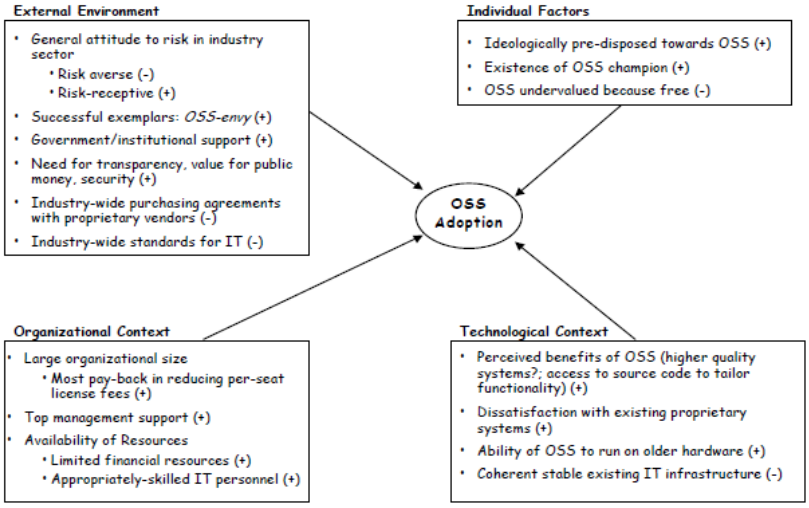
Informações da QPc		
		• Metodologia de desenvolvimento geralmente não definida; • Pouca documentação; • Testes e metodologia de garantia da qualidade informais e não estruturados; • Requisitos definidos por programadores; • Sem um processo de avaliação formal de riscos; • Poucos objetivos mensuráveis; • Descoberta de defeitos através de testes caixa-preta no final do processo; • Evidência empírica referente a qualidade não é coletada; • Os membros da equipe escolhem as tarefas; • O projeto segue diretamente para a fase de programação; • Pouco esforço colocado no planejamento do projeto. Esta pesquisa concluiu que o desenvolvimento de software livre utiliza algumas boas práticas do desenvolvimento tradicional, mas que estas práticas são realizadas de forma diferente. Fatores que contribuem para a alta qualidade do software livre: • Existência de uma **comunidade sustentável** que possa desenvolver e depurar o código rapidamente para construir novas funcionalidades. Nesta comunidade, os membros assumem diferentes papéis, definidos de acordo com o modelo cebola. A sustentabilidade da comunidade é garantida através da meritocracia que reconhece o crescimento do envolvimento de determinados membros e exerce forte atração sobre os programadores voluntários. • A **modularidade do código** permite que muitos programadores trabalhem no código do software, em módulos separados, sem interferir no trabalho dos demais. • A **revisão por pares** avalia uma contribuição antes que ela seja incorporada ao repositório de código fonte. Uma diferença significativa é que no desenvolvimento de software livre a revisão por pares é uma tarefa contínua e rotineira, enquanto no desenvolvimento tradicional, apesar de usada durante décadas, a revisão por pares corresponde no máximo a um ou dois passos do processo de desenvolvimento. • O desenvolvimento de software livre estabelece um **ambiente e cultura** que encorajam a inovação e a criatividade, criando um processo mais focado nas pessoas e não nos métodos e ferramentas como no desenvolvimento tradicional. • Enquanto no desenvolvimento tradicional, o rigor nos **testes** é um fator importante para garantir a qualidade, no desenvolvimento de software livre este rigor é substituído pelo lançamento frequente de versões que permite que toda a comunidade de usuários e desenvolvedores valide o software e, em alguns casos, pelos testes de regressão automatizados.
	Informações da QS5	
	Modelo de processos com conciliação:	O artigo cita os seguintes modelos de avaliação de qualidade: • **Cap Gemini`s Open Source Maturity Model (OSMM)**: modelo comercial que avalia o software em relação a um conjunto de indicadores do produto. • **Navica`s Open Source Maturity Model (OSMM)**: processo formal, publicado sob uma licença livre, que avalia a maturidade do software, suporte, documentação, treinamento e serviços. • **Business Readiness Rating (OpenBRR)**: modelo de avaliação que está em desenvolvimento por um conjunto de organizações que procura expandir os dois modelos anteriores para criar um padrão aberto.
9.	**Título do documento:**	Adapting the "staged model for software evolution" to free/libre/open source software
	Autor(es):	A. Capiluppi; J. M. Gonzalez-Barahona; I. Herraiz; G. Robles
	Data de publicação:	2007
	Fonte:	International Workshop on Principles of Software Evolution (IWPSE)
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica

Informações da QPc		
	Critérios de comparação:	O artigo foca nas similaridades e diferenças entre a evolução e os padrões de ciclo de vida dos sistemas comerciais tradicionais e do software livre. O modelo em estágios (ou fases) existente para a evolução do software é revisado para se compatibilizar também com os projetos de software livre (Figura 9). Cada uma das fases do modelo é analisada para verificar sua adequação ao software livre: • **Fase de desenvolvimento inicial:** os sistemas comerciais tradicionais são construídos por uma quantidade fixa de projetistas, desenvolvedores e testadores. O executável do software é publicado somente quando a primeira versão está executando corretamente. Esta versão será então utilizada pelos usuários. Já no software livre, os projetos tipicamente iniciam com poucos desenvolvedores e, posteriormente, novos desenvolvedores podem ir se juntando ao grupo. O código fonte do sistema é publicado bem antes de estar completo ou funcionando. Os receptores desta versão não são apenas os usuários, mas também outros desenvolvedores. Todos recebem acesso de leitura ao repositório de gerência de configuração do projeto, a partir de onde podem acessar o código fonte.  Figura 9 – Modelo de evolução em estágios adaptado ao software livre • **Fase de mudanças evolutivas:** o lançamento de várias versões é observado tanto nos sistemas comerciais tradicionais quanto no software livre. Porém, nos sistemas comerciais tradicionais uma alta frequência de mudanças é vista como um fator de instabilidade. O *feedback* é fornecido pelos usuários através de solicitações de mudanças e coletadas como requisitos para versões futuras. No software livre as versões são lançadas com mais frequência e isso é percebido como um fator de vitalidade. As versões podem ser lançadas assim que estão prontas ou, mais recentemente em alguns projetos, é possível que haja um cronograma planejado para o lançamento de versões para oferecer uma versão estável periodicamente aos usuários. O *feedback* é fornecido pelos usuários da mesma forma do desenvolvimento tradicional, mas também sob a forma de *patches* de correção de código que os próprios usuários escrevem e que, possivelmente, será incorporada em versões futuras. Nos dois tipos de projetos, esta fase pode durar anos já que ambos possuem uma característica de longa vida do software, mas alguns projetos de software livre apresentam uma taxa de crescimento superlinear que parece contrariar as Leis de Lehman para Evolução de Software que aponta um padrão de desaceleração do tamanho ao longo do tempo. • **Fase de serviços:** o sistema é considerado maduro e novas melhorias não são mais incorporadas. No desenvolvimento tradicional, quando os benefícios de um sistema não estão mais balanceados com o seu custo de manutenção, o sistema deixa de sofrer evolução e se torna um sistema legado. No software livre, a fase de serviços também pode ser detectada, mas depois ela é seguida por um novo período de evolução. • **Fase de retirada gradual:** no desenvolvimento tradicional esta fase é

Informações da QPc		
		atingida quando a empresa responsável declara que não fará mais manutenções, nem mesmo corretivas, no sistema. O mesmo comportamento pode ser observado no software livre quando uma equipe de desenvolvimento declara que não tem mais a intenção de manter o sistema. Entretanto, no software livre, como o código fonte está disponível, uma nova equipe de desenvolvedores pode assumir este sistema e levá-lo a uma nova fase de mudanças evolutivas.
10.	**Título do documento:**	A development process for building OSS-based applications
	Autor(es):	M. Huang; L. Yang; Y. Yang
	Data de publicação:	2006
	Fonte:	International Software Process Workshop (SPW)
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	Construir aplicações baseadas em software livre (*Open Source Software Based Application* - OBA) já existente é uma tarefa necessária para muitas organizações. Entretanto, sem um processo de desenvolvimento bem definido, que acomode as particularidades do software livre, esta tarefa pode representar um alto risco para as organizações. Enquanto o uso de software livre pode acelerar e reduzir os custos de desenvolvimento, as consequências de selecionar o produto inadequado podem prejudicar todos estes possíveis benefícios. Construir uma OBA requer modificações, melhorias e integrações do software livre já existente e o processo de desenvolvimento tradicional pode não funcionar bem diante deste cenário. Os autores consideram que construir uma OBA é similar a desenvolver um software de prateleira. As tarefas principais nestes dois cenários são: selecionar o produto mais adequado (levando em consideração que a qualidade dos produtos de software livre pode ser irregular e que eles variam bastante em relação as funcionalidades), ajustá-lo de acordo com as necessidades e integrá-lo. A principal diferença reside no fato que o componente ou software de prateleira geralmente não pode ser modificado, pois não se tem acesso ao seu código fonte, enquanto no software livre esta possibilidade existe, já que o código fonte está disponível.
	Informações da QS2	
	Descrição da proposta de conciliação:	A partir destas similaridades e diferenças, os autores propõem um processo de desenvolvimento para OBA análogo ao processo de desenvolvimento de software de prateleira (Figura 10). Este processo foi aplicado em um estudo de caso de desenvolvimento de um sistema de e-mail utilizando software livre. Este processo análogo leva em consideração as seguintes especificidades do desenvolvimento de OBA: • No desenvolvimento de software de prateleira é difícil definir o projeto de alto nível da arquitetura do software, pois algumas interfaces entre os produtos selecionados não podem ser construídas. Assim, a arquitetura é definida em paralelo com a seleção dos produtos. Porém, no desenvolvimento de OBA é possível definir esta arquitetura logo no início já que as interfaces entre os produtos de software livre podem ser ajustadas para se encaixar na arquitetura definida para o software. Além disso, esta definição da arquitetura favorece a modularidade e a manutenibilidade e permite que outros produtos de software livre possam ser adicionados com maior eficiência. • Alguns atributos específicos do software livre devem ser levados em consideração na avaliação dos produtos. Os critérios de avaliação de software de prateleira geralmente incluem atributos de sistemas como funcionalidade, desempenho. Estes atributos também são importantes de serem levados em consideração na avaliação de software livre, mas adicionalmente outros atributos também devem ser considerados, tais como: estabilidade da arquitetura, modularidade, manutenibilidade, uso das ferramentas de desenvolvimento adotadas no projeto de software livre, qualidade e facilidade de compreensão do código fonte do software e licença. Uma lista completa dos critérios de avaliação do

Informações da QPc		
		software livre é oferecida pelos autores no artigo. • O processo de integração de software de prateleira é uma reutilização em caixa-preta onde é desenvolvido um "glue code" para que os componentes possam funcionar em conjunto. Este desenvolvimento é apoiado pela documentação existente e pelo suporte do fornecedor do produto. No caso do software livre, nem sempre existe essa documentação e suporte. Assim, é necessário estudar e entender o código fonte do software para analisar quais alterações serão necessárias, quais partes devem ser modificadas e como elas devem ser modificadas para evitar possíveis problemas. Além disso, deve-se levar em consideração a integração com os produtos já existentes na organização (Figura 11).  Figura 10 – Processo de desenvolvimento para construção de OBA  Figura 11 – Atividades para melhoria e integração de software livre
11.	**Título do documento:**	Collaboration, peer review and open source software
	Autor(es):	J.P. Johnson
	Data de publicação:	2006
	Fonte:	Information Economics and Policy
	Processos de desenvolvimento	Tradicional e livre

Informações da QPc		
	de software:	
	Práticas:	Colaboração e revisão por pares A estrutura organizacional influencia a eficiência e a produtividade. A estrutura de uma comunidade de software livre minimiza os custos associados com a distribuição de informações. Tal fato se manifesta em duas práticas principais: colaboração (compartilhamento de ideias) e revisão por pares. Como o sucesso destas práticas está ligado a estrutura organizacional mais flexível das comunidades de software livre, as organizações podem encontrar dificuldades em adotar completamente estas práticas.
	Critérios de comparação:	**1) Revisão por pares:** • **Preocupação com a carreira:** nas organizações, a preocupação com a carreira, que é determinante para o valor e o pagamento dos salários, faz com que os participantes evitem ser muito críticos nas revisões por pares com medo de serem alvo dela no futuro. Já no software livre, como não existe esta mesma preocupação e os desenvolvedores são motivados pelo desejo de desenvolver um software de qualidade, eles são mais críticos e cuidadosos nas revisões. O autor assume que em ambos os casos existe a preocupação com a reputação. **2) Colaboração** • **Autoridade formal:** enquanto a autoridade formal, presente nas organizações, pode fazer com que os funcionários trabalhem mais arduamente para implementar potenciais melhorias no sistema, esta mesma autoridade também pode fazer com que os funcionários omitam informações para reduzir a sua carga de trabalho. Já no desenvolvimento de software livre, onde não existe esta autoridade formal, os participantes são mais propensos a compartilhar ideias e informações já que eles não podem ser forçados a implementá-las.
12.	**Título do documento:**	Commercial adoption of open source software: An empirical study
	Autor(es):	E. Glynn; B. Fitzgerald; C. Exton
	Data de publicação:	2005
	Fonte:	International Symposium on Empirical Software Engineering (ISESE)
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS1	
	Desafios e oportunidades para a conciliação:	No artigo os autores investigam a adoção do software livre nas organizações para identificar os fatores que levam estas organizações a se dispor a assumir os riscos dessa iniciativa e para identificar os fatores que podem ajudar a mitigar estes riscos. Para realizar este estudo foi desenvolvido um *framework*, baseado na teoria da adoção da inovação, para refletir o processo de adoção de software livre. Este *framework* é composto por quatro categorias (Figura 12): ambiente externo, contexto organizacional, fatores individuais e contexto tecnológico. Cada categoria possui fatores facilitadores ou inibidores à adoção do software livre nas organizações. A relevância dos fatores deste *framework* foi avaliada inicialmente no contexto de um estudo de caso em um hospital. Em seguida, estes fatores foram utilizados em um questionário para medir o nível de assimilação do software livre nas organizações. Os resultados mostraram que em relação ao ambiente externo, as organizações são muito influenciadas pela possibilidade de colaboração com uma grande comunidade de software livre. Além disso, a percepção de outras organizações que também estejam usando o software livre é outro aspecto importante. Em relação ao contexto organizacional, destaca-se a atratividade do custo do software livre e a disponibilidade da equipe de desenvolvedores já existentes na comunidade de software livre. Em termos dos fatores tecnológicos, a possibilidade de modificar o código-

Informações da QPc		
		fonte e a transparência no acesso ao código foram elencadas como importantes. Nesta categoria, a ausência de um fornecedor, que se responsabilize pela manutenção e suporte, é apontada como um fator inibidor significativo. Na categoria dos fatores individuais, a importância do apoio a ideologia do software livre e a existência de um líder da comunidade de software foram destacados.  Figura 12 – *Framework* de investigação da adoção do software livre
13.	**Título do documento:**	Open knowledge management: Lessons from the open source revolution
	Autor(es):	Y. Awazu; K. C. Desouza
	Data de publicação:	2004
	Fonte:	Journal of the American Society for Information Science and Technology
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Gestão de conhecimento Os autores examinaram algumas comunidades de software livre para gerar ideias sobre como melhorar as práticas atuais de gestão de conhecimento das organizações aumentando a sua transparência. Estas ideias podem ser consolidadas em um novo modelo de gestão de conhecimento (*Open Knowledge Management*) que poderia ser particularmente útil em ambientes onde especialistas precisam compartilhar conhecimento.
	Critérios de comparação:	• No desenvolvimento tradicional, o conhecimento é considerado um recurso raro que precisa ser protegido e uma fonte de poder. Assim, dificilmente ele é livremente compartilhado, ficando restrito aos participantes da tarefa. Já as comunidades de software livre se baseiam na troca de conhecimento através de tecnologia de informação e comunicação. • A disponibilidade do código para todos os usuários facilita a inovação, pois permite que os usuários se envolvam na correção e melhoria do software. Além disso, evita que o software seja usado como uma fonte de poder por parte daqueles que o conhecem, o que geralmente prejudica o fluxo de conhecimento nas organizações. • A aquisição de conhecimento nas organizações é uma tarefa tediosa, pois as fontes de conhecimento são difíceis de identificar e o conhecimento necessário geralmente não está explicitado, pois ele é restrito as pessoas realmente envolvidas na prática. Já o desenvolvimento de software livre não discrimina os participantes, o que amplia as fontes de conhecimento e potencializa as fontes de ideias para inovação. • Em muitas organizações, os indivíduos relutam em arcar com os custos de explicitar o conhecimento. Assim, prevalece a regra do 80-20 onde 80% das contribuições são feitas por 20% dos funcionários. Essa característica também é observada nos projetos de software livre, onde

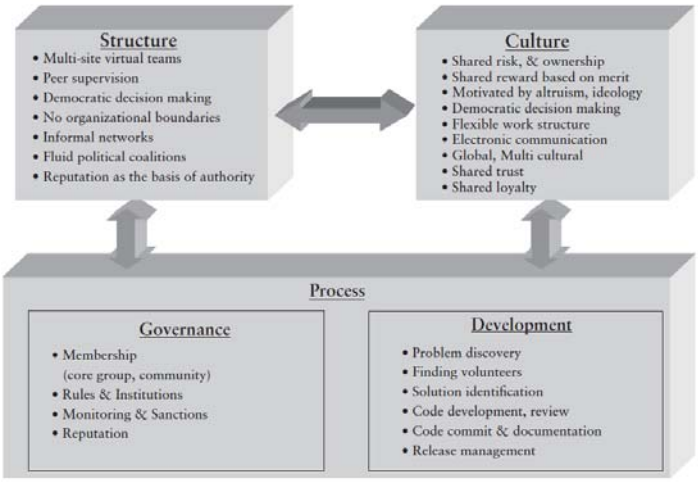
		também existe um mecanismo de recompensa pelas contribuições dos desenvolvedores através do crescimento da reputação deste indivíduo e da meritocracia que promove estes indivíduos a posições de destaque dentro da comunidade. • A preocupação com a reputação diante dos seus pares (monitoramento horizontal) também ajuda a garantir a qualidade das contribuições realizadas pelos membros das comunidades de software livre. No desenvolvimento tradicional predomina o monitoramento vertical, ou seja, um controle do chefe sobre o subordinado e uma tomada de decisão *top-down*. • Na maioria das organizações, os sistemas de gestão de conhecimento são criados com artefatos pobres e incompletos, sem nenhum mecanismo de garantia de qualidade que verifique se esse conhecimento é válido e útil. Assim, as buscas se tornam mais difíceis e os usuários acabam abandonando o uso do sistema. As comunidades de software livre são imunes a este problema, pois os coordenadores do projeto garantem que só serão incorporadas contribuições realmente relevantes e mantêm a qualidade do conhecimento armazenado.
	Informações da QS1	
	Desafios para a conciliação:	• **Gestão de conhecimento:** conhecimento visto como uma fonte de poder nas organizações, o que desencoraja a inovação, a reutilização e o compartilhamento de conhecimento.
	Oportunidades para a conciliação:	• **Disponibilidade do código:** a disponibilidade do código para todos os usuários facilita a inovação, pois permite que os usuários se envolvam na correção e melhoria do software
	Informações da QS2	
	Descrição da proposta de conciliação:	Para que as organizações possam incorporar essas práticas nos seus ambientes é importante: • O acesso ao conhecimento necessário para solucionar um problema seja rápido e fácil. • Implementar mecanismos de incentivo que beneficiem os contribuidores. • Atrair, além dos especialistas, os participantes passivos, pois é o reconhecimento deste público que incentiva os especialistas a contribuir. Além disso, a reunião desta variedade de pessoas ajuda a diversificar o conhecimento. • Ainda que estejam geograficamente distribuídos, é importante que os colaboradores tenham interesses e experiências comuns que os aproximem. Essa proximidade lógica ajuda a criar o sendo de comunidade e encoraja uma transferência de conhecimento mais rápida entre os especialistas e os novatos. Assim, uma organização pode tentar segmentar as iniciativas de gestão de conhecimento por áreas de domínio com um ambiente dedicado a cada comunidade criada. • A motivação para a gestão de conhecimento não pode partir da alta gerência, pois para ser efetiva é importante que essa motivação seja intrínseca aos participantes e que os grupos se formem naturalmente e não por uma determinação da empresa.
14.	**Título do documento:**	Open source software - An evaluation
	Autor(es):	A. Fuggetta
	Data de publicação:	2003
	Fonte:	Journal of Systems and Software
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	O software livre é um importante fenômeno. Assim, é essencial aprofundar o entendimento da natureza do software livre e dos fatores que motivaram o seu sucesso. Entretanto, muitas observações sobre o software livre parecem ser fracamente motivadas ou baseadas em argumentos falhos devido à falta de entendimento. Assim, se torna difícil identificar os aspectos novos do desenvolvimento de software livre em relação às práticas do desenvolvimento de software tradicional.

Informações da QPc		
		O objetivo deste artigo é realizar uma avaliação qualitativa sobre a abordagem de software livre, discutindo criticamente as afirmações associadas ao desenvolvimento de software livre. **1) Aspectos técnicos:** • A flexibilidade defendida pelo modelo bazar não é original do software livre. Todas as críticas ao desenvolvimento em cascata são justamente baseadas na ideia de que não é possível desenvolver um software seguindo um processo tão rígido. Assim, várias abordagens buscam essa flexibilidade: prototipação rápida, desenvolvimento iterativo e incremental, ciclo de vida espiral, desenvolvimento rápido de aplicações e, mais recentemente, os métodos ágeis. Todas estas abordagens podem ser aplicadas tanto ao software proprietário quanto ao software livre. • O uso da Internet como canal de comunicação e colaboração também é comum tanto ao desenvolvimento de software tradicional quanto ao software livre. • A abertura do código não é a única forma de se motivar os desenvolvedores. • Sobre a necessidade de pessoal qualificado, hoje em dia qualquer domínio de negócio já reconhece a importância da qualificação do pessoal e as organizações levam este aspecto em consideração na sua seleção de pessoal. • O modelo de bazar pode não ser a melhor alternativa de desenvolvimento para todos os projetos de desenvolvimento de software, principalmente aqueles críticos de segurança. Por outro lado, o desenvolvimento de software livre não é a única forma de se realizar um desenvolvimento aos moldes do modelo bazar. Não existe nenhuma evidência de que o desenvolvimento de software livre implica em um processo de desenvolvimento no modelo bazar. • Com base nas pesquisas em engenharia de software e gerência de requisitos, e utilizando o Linux como exemplo, o autor conclui que a colaboração é possível se os desenvolvedores compartilham o conhecimento sobre os requisitos e a arquitetura do software. Isso explicaria porque a maioria dos projetos de software livre se concentra em desenvolvimento de infraestrutura. Assim, não é possível garantir que o desenvolvimento de software livre também vai funcionar tão bem para softwares de negócio ou para projetos de pesquisa e inovação. • Alguns produtos de software livre, como o Zope, amplamente adotados só se tornaram software livre depois que eles já estavam concluídos. Então, não seria correto atribuir a qualidade destes produtos ao processo de desenvolvimento de software livre. • O software livre não é essencial para a interoperabilidade entre os produtos. A interoperabilidade é atingida através de padrões abertos e não do software livre. Os padrões abertos normalmente definem interfaces. Estas interfaces podem ser implementadas através de software livre ou não. • Para muitos usuários finais a disponibilidade do código é absolutamente insignificante, pois eles não vão saber como utilizá-lo. Mesmo bons desenvolvedores vão encontrar dificuldades em entender o código de um grande sistema, sem nenhuma documentação. Então, é necessária informação de mais alto nível, além do código, para ajudar no seu entendimento e evolução. • Como é possível afirmar que os softwares comerciais existentes têm qualidade inferior ao software livre, com tantos exemplos em funcionamento, com sucesso, em sistemas críticos de aviação, finanças e etc e sem nenhuma evidência científica? • O princípio da inspeção de código também não é uma invenção do software livre. A única diferença é que dentro de uma comunidade de software livre o código pode ser inspecionado por um grande número de desenvolvedores. • Não existem também evidências de que o sucesso e a qualidade de softwares livres, como Linux e Apache, se deve ao fato deles terem sido desenvolvidos como projetos de software livre. **2) Aspectos econômicos e de negócio:** • O modelo de negócio baseado em venda de serviços não é novo e não se restringe ao software livre. • Abrir o código de um software já se mostrou uma estratégia comercial poderosa para proteger ou conquistar um mercado. Alguns exemplos de uso dessa estratégia por grandes empresas são: o caso da Netscape na sua disputa com a Microsoft em relação ao navegador web; o caso da

Informações da QPc		
		Sun que promove o Star Office como concorrente do Microsoft Office; e o caso da IBM e Sun que apóiam o desenvolvimento do Apache para evitar que o mercado de servidores de aplicação seja dominado de vez pela Microsoft o que reduziria o seu mercado de hardware para servidores. • Todo desenvolvimento de software tem um custo e o software livre não é exceção. No caso do software livre o que muda é como será pago o custo deste desenvolvimento. Este custo de desenvolvimento pode ser encarado como um investimento que será pago pela venda de serviços ou pode ser pago por uma empresa interessada em proteger ou conquistar um mercado. • Mesmo que os gastos atuais com desenvolvimento de software sejam excessivos devido aos esforços repetidos e aos custos das licenças, ainda assim não seria razoável imaginar que todas as necessidades de software poderiam ser supridas com software livre financiado com dinheiro público ou apenas com o voluntariado dos desenvolvedores e seria arriscado tentar impor essa abordagem de software livre a todos os domínios. Por outro lado, existem situações particulares de mercado onde a adoção do software livre é a única alternativa razoável, como no desenvolvimento cooperativo entre desenvolvedores de um projeto de pesquisa que precisa criar um produto específico. • O software livre também é considerado uma forma de proteger os usuários e as instituições governamentais, porque pode ser inspecionado para verificar se os requisitos foram atendidos e se está aderente aos requisitos de segurança. Todavia, no desenvolvimento de software tradicional, as organizações podem resolver este problema adquirindo os direitos sobre o código fonte do software desenvolvido de forma customizada pela contratada. Essa medida também aumenta a autonomia da organização para manutenções futuras por outras contratadas.
15.	**Título do documento:**	A study of configuration management in open source software projects
	Autor(es):	U. Asklund; L. Bendix
	Data de publicação:	2002
	Fonte:	IEE Proceedings: Software
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Gerência de configuração de software Os autores examinaram o processo de gerência de configuração de software de três projetos de software livre (KDE, Mozilla e Linux) para analisar como o processo, as ferramentas e os aspectos humanos da gerência de configuração contribuem para o seu sucesso. Os autores ainda discutem como as melhores práticas de gerência de configuração dos projetos de software livre podem ser podem ser transferidas para o desenvolvimento de software tradicional.
	Critérios de comparação:	Analisa o processo de desenvolvimento de software livre, através de um *framework* de gerência de configuração criado, e compara com o desenvolvimento tradicional. • **Perspectiva do desenvolvedor:** no desenvolvimento tradicional, geralmente, é adotada uma perspectiva gerencial, onde o objetivo é controlar o desenvolvimento do produto. Porém, no desenvolvimento de software livre, prevalece a perspectiva do desenvolvedor, onde o objetivo da gerência de configuração é oferecer um ambiente estável de trabalho e aumentar a eficiência. • **Controle de versões:** uso de ferramentas para armazenar as versões em um repositório, minimizando o consumo de espaço. As versões quase nunca são usadas para restaurar uma versão anterior e sim para entender, através do log, como um arquivo foi desenvolvido e para comparar versões usando a funcionalidade de *diff*. • **Gestão de *builds*:** devido à necessidade de trabalhar *offline*, a área de trabalho local de cada desenvolvedor contém todos os arquivos do código fonte necessários para a construção de *builds.* • **Seleção de configuração:** a última versão de cada um dos arquivos é usada para construir a configuração. Assim, a seleção é trivial.

Informações da QPc		
		• **Gestão da área de trabalho:** a ferramenta de gerência de configuração de software adotada apóia esta tarefa trabalhando com o conceito de projeto e permitindo operações atômicas para criar a área de trabalho e para sincronizar a área de trabalho com o repositório. Desta forma, os desenvolvedores não precisam estar continuamente conectados ao repositório. • **Controle de concorrência:** adota a política otimista, o que significa que os arquivos não são bloqueados para edição dos outros usuários quando são copiados do repositório por um dos usuários. Apesar do grande número de desenvolvedores e do desenvolvimento rápido, a ocorrência de conflitos é pequena e as alterações são mescladas pelos próprios desenvolvedores que se comunicam diretamente. As listas de discussão são utilizadas para oferecer percepção aos demais participantes sobre o que está sendo feito e assim o risco de um conflito complexo que demandaria maior esforço para ser resolvido. • **Gestão de mudanças:** muitos projetos de desenvolvimento tradicional usam um comitê para revisar as solicitações de mudanças e as tarefas são designadas aos desenvolvedores para implementação. No desenvolvimento de software livre a revisão das solicitações de mudanças não existe ou não é explícita e os próprios desenvolvedores escolhem as tarefas nas quais querem trabalhar com base nas suas habilidades e disponibilidade. Depois da implementação, o *patch* é avaliado (através de testes, revisões e discussões) e pode ser aprovado para ser atualizado no repositório ou rejeitado. O acesso para atualização dos *patches* no repositório, geralmente, é limitado aos desenvolvedores principais que aplicam as atualizações que forem aprovadas; • **Gestão de *releases*:** nenhum dos projetos de software livre estudados empacota o software do jeito tradicional, reunindo código, documentação, arquivos de help, scripts de instalação e etc. Além disso, nenhum deles trabalha com datas fixas para o lançamento de *releases* que ocorre de forma arbitrária. Fatores importantes da gerência de configuração em software livre: • **Ferramentas:** como em todo projeto de desenvolvimento de software, um conjunto de ferramentas é usado. No desenvolvimento de software livre prevalece o uso de ferramentas livre como CVS ou Subversion acompanhadas de ferramentas como e-mail, lista de discussão e web browsers. Além disso, adotam uma arquitetura cliente servidor com 1 único servidor atendendo múltiplos clientes para que não seja necessária replicação de servidores. Como todos os desenvolvedores do projeto vão precisar aprender sozinhos a usar a ferramenta é importante que ela seja simples e que ofereça percepção sobre o andamento do projeto. • **Processo:** pelos mesmos motivos citados anteriormente, o processo precisa ser simples e fácil de ser executado. • **Pessoas:** os moderadores e desenvolvedores protegem o repositório do código fonte do projeto. Assim, mesmo que os desenvolvedores ruins possam atrapalhar o andamento do projeto, pelo menos eles não conseguem destruir o projeto. Os moderadores não podem tentar melhorar as contribuições para evitar que eles se tornem gargalos. Por outro lado, como todos os desenvolvedores se preocupam com a sua reputação na comunidade, essa pressão social vira um fator motivador que ajuda a garantir a qualidade das contribuições enviadas. Todos os membros do projeto de software livre devem gostar de discutir o trabalho e compartilhar o conhecimento com os demais. O desenvolvimento de software livre é um trabalho de equipe e precisa de muita comunicação.
	Informações da QS2	
	Descrição da proposta de conciliação:	Alguns destes fatores, mas não todos, podem ser transferidos para o desenvolvimento de software tradicional. Por outro lado, algumas práticas de gerência de configuração do desenvolvimento de software tradicional também podem beneficiar os projetos de software livre. • Simplificar o processo de desenvolvimento de software tradicional evitando *branches* desnecessários. Para isso é possível dividir as tarefas em incrementos menores. Isso também facilita a criação de *baselines* frequentes, o que aumenta a percepção da equipe de desenvolvimento, e a integração de código; • Proteger o repositório de código fonte contra alterações indesejadas,

Informações da QPc		
		através da adoção de um papel similar ao do moderador; • Utilizar um modelo de gerência de configuração que funcione tanto *online* quanto *offline*, principalmente se o desenvolvimento é geograficamente distribuído; • Permitir a comunicação direta entre os desenvolvedores, pois quando a comunicação segue apenas através das camadas gerenciais ela é lenta e ineficiente; • A escolha de tarefas pelos desenvolvedores só é possível no software livre, pois como usuários do software estes desenvolvedores são altamente motivados, mas esta prática é difícil de transferir para o desenvolvimento tradicional; • Muitos projetos de software livre poderiam se beneficiar da existência de uma lista de alterações desejadas e de uma melhor rastreabilidade entre uma solicitação de mudança e a sua implementação no código; • A priorização dos requisitos, com base na severidade do problema, nos custos de implementação e etc também é uma necessidade nos projetos de software livre, mas é uma prática difícil de ser combinada com a escolha de tarefas pelos próprios desenvolvedores. • O processo de gestão de mudanças dos projetos de software livre precisaria ser revisto para evitar que solicitações que seriam rejeitadas por uma análise acabem sendo implementadas. • A capacidade de manter diferentes *releases* é, geralmente, uma necessidade no desenvolvimento tradicional, mas que não existe no desenvolvimento de software livre.
16.	**Título do documento:**	A framework for creating, hybrid-open source software communities
	Autor(es):	S. Sharma; V. Sugumaran; B. Rajagopalan
	Data de publicação:	2002
	Fonte:	Information Systems Journal
	Processos de desenvolvimento de software:	Tradicional e livre
	Práticas:	Não se aplica
	Critérios de comparação:	Apesar de décadas de pesquisa, o desenvolvimento de software continua sofrendo com problemas causados pela ineficiência da estrutura e dos processos organizacionais e pela forma como os projetos são gerenciados. Os proponentes do software livre sugerem que este modelo poderia contribuir para resolver alguns destes problemas, ainda que não seja uma panacéia que irá resolver todos os problemas. Apesar do sucesso do modelo de software livre, as organizações estão encontrando dificuldades em construir um modelo de negócio sob o paradigma do software livre e faltam estudos científicos sobre como as organizações podem implantar e se beneficiar do software livre. Assim, com base em uma teoria organizacional, os autores analisam as comunidades de software livre para entender o seu funcionamento (Figura 13). **1) Estrutura** • **Divisão do trabalho:** diferentemente das organizações tradicionais, as comunidades de software livre são compostas por um grande número de desenvolvedores voluntários que fazem suas contribuições nas tarefas que eles próprios escolhem, pois os projetos não são guiados por nenhum plano ou cronograma formal. • **Mecanismos de coordenação:** os desenvolvedores nas comunidades de software livre estão geograficamente distribuídos e raramente se encontram pessoalmente. Além disso, eles não conseguem dedicar uma grande quantidade de tempo de forma contínua ao projeto. Assim, uma estrutura formal de coordenação hierárquica não funcionaria. Por isso, são usados mecanismos de coordenação que enfatizam a descentralização do trabalho e a comunicação assíncrona. Geralmente, um grupo principal de desenvolvedores fornece uma visão geral do direcionamento estratégico do projeto. • **Tomada de decisão distribuída:** os indivíduos têm o poder de tomada de decisão e as decisões geralmente são por consenso, utilizando ferramentas de comunicação assíncrona como e-mail e listas de discussões. Esta tomada de decisão descentralizada difere bastante do

Informações da QPc		
		modelo tradicional onde o controle é rígido e o poder de tomada de decisão, em geral, é proporcional ao nível hierárquico. • **Limites organizacionais:** as comunidades de software livre não possuem limites organizacionais bem definidos, pois a participação em uma comunidade é muito fluida e os indivíduos têm liberdade para entrar ou sair de uma comunidade. • **Estruturas informais:** nas comunidades de software livre não existe uma estrutura organizacional formal. Mesmo as estruturas informais mudam de acordo com o tempo, as necessidades e as ações dos membros da comunidade. • **Autoridade:** as comunidades de software livre são baseadas na meritocracia, onde a reputação do indivíduo é estabelecida através da qualidade das contribuições oferecidas e que podem levar ao reconhecimento e a um papel de liderança dentro do grupo.  Figura 13 – Modelo de software livre **2) Processos** **2.1) Processos de governança:** uma característica importante das comunidades de software livre é a sua auto-governança. Em geral, o desenvolvedor inicial do software mantém um papel de liderança, mas a autoridade é do grupo. Os projetos são divididos em módulos delegados a indivíduos ou equipes. Os líderes são os responsáveis pela coordenação das equipes. • **Gestão das participações:** a participação nas comunidades de software livre é aberta a todos que desejem colaborar. Em alguns casos, existe um estágio transitório de avaliação antes do novo membro ser oficialmente aceito pela comunidade. Enquanto a participação geral é aberta ao público, novos membros só são adicionados ao grupo principal quando é um participante frequente, indicado por um dos demais membros do grupo principal e aprovado pelos demais. • **Regras:** ao contrário do modelo tradicional, onde as regras são criadas pelo nível gerencial, as comunidades de software livre criam colaborativamente os seus próprios conjuntos de regras e normas que são modificadas ao longo do tempo de acordo com a maturidade do projeto. • **Monitoramento e sanções:** de forma similar ao que existe nas organizações tradicionais, porém com menos formalismo e rigidez, as comunidades de software livre também possuem mecanismos para observar o comportamento e garantir o cumprimento das regras estabelecidas para o projeto. Existe uma pressão social contra os indivíduos que não cumprem as regras da comunidade, mesmo os líderes. Geralmente, os membros sancionados por uma comunidade mudam o seu comportamento ou deixam o grupo. • **Reputação:** construir e manter uma reputação é um dos principais motivadores dos desenvolvedores em uma comunidade de software livre. O desejo de construir uma reputação atrai novos membros para as comunidades e a preocupação em manter esta reputação faz com que eles concluam as suas tarefas no prazo e apresentem contribuições de qualidade. **2.2) Processos de desenvolvimento de software:** em um projeto de

Informações da QPc		
		software livre, os desenvolvedores interagem, através de um conjunto comum de ações, enquanto trabalham no código fonte do software. • **Descoberta do problema:** os problemas são informados e discutidos através de listas de discussões de desenvolvimento, sistemas de controle de erros. Os erros informados têm alta prioridade e recebem a atenção de todos os desenvolvedores ativos do projeto. • **Encontrando voluntários:** depois que o problema é descoberto, voluntários se apresentam para trabalhar nele. Os voluntários preferem trabalhar em problemas relacionados as partes familiares. Novos desenvolvedores trabalham em partes onde os desenvolvedores anteriores já não estão mais interessados ou no desenvolvimento de novas arquiteturas e funcionalidades. • **Identificação da solução:** em geral, várias alternativas de solução são apresentadas e são escolhidas aquelas soluções mais genéricas e portáveis. A alternativa escolhida é divulgada na lista de discussão dos desenvolvedores para *feedbacks* antes da sua implementação. • **Desenvolvimento do código e testes:** o desenvolvedor implementa as mudanças em uma cópia local do código fonte e testa as mudanças no seu próprio ambiente. • **Revisão da mudança no código:** a solução testada é divulgada na lista de discussão dos desenvolvedores para revisão. Outros desenvolvedores testam a solução. Se eles encontrarem algum problema, eles sugerem melhorias ao desenvolvedor do código. Este processo é repetido até que a mudança no código seja aprovada. • **Implantação de código e documentação:** a implantação do código, através da ferramenta de gerência de configuração, resulta em um resumo das principais modificações realizadas que também é armazenado no repositório e divulgado aos demais desenvolvedores para revisão. • **Gerência de versões:** alguns voluntários atuam como gerentes de versões quando o projeto se aproxima do lançamento de uma nova versão do software. **3) Cultura:** seguindo o *framework* do Schein (Schein, 1996), a cultura pode ser entendida examinando os artefatos, valores e as principais suposições. • **Artefatos:** no modelo de software livre, os artefatos incluem um livre fluxo de informação, onde a comunicação é mediada por computador e assíncrona, o que permite que desenvolvedores de diferentes localidades e culturas participem do projeto. • **Valores:** diferentemente dos valores das organizações tradicionais que enfatizam os benefícios financeiros e o poder, os membros das comunidades de software livre dão valor ao altruísmo, reciprocidade, doação, reputação e ideologia. Apesar de motivados pelos seus benefícios pessoais de usar uma versão melhorada do produto, as recompensas financeiras não parecem tão importantes. Eles também dão valor à justiça, transparência e ao consenso na tomada de decisão. Como consequência, o trabalho é realizado no ambiente aberto e visível da Internet, onde o desempenho de cada membro pode ser monitorado pelo restante da comunidade. Os membros das comunidades de software livre acreditam em riscos, recompensas e propriedade compartilhados. • **Suposições:** as principais suposições nas comunidades de software livre são confiança e lealdade, fatores estes que não são utilizados como base nas organizações tradicionais.
	Informações da QS2	
	Descrição da proposta de conciliação:	Apesar do grande número de benefícios obtidos com a implantação do software livre, este modelo pode não ser adequado a todas as organizações e todos os tipos de projeto. Entretanto, os autores defendem que as organizações podem incorporar algumas características, com diferentes intensidades, no seu ambiente de desenvolvimento, criando um ambiente híbrido que pode lhes ajudar a alcançar alguns dos benefícios proporcionados pelo modelo de desenvolvimento de software livre. Assim, para ajudar as organizações na criação deste ambiente híbrido, a partir da teoria organizacional, baseada nos aspectos estrutura, processo e cultura, os autores propõem um *framework* para a criação e gestão de uma comunidade híbrida de software livre dentro da organização. Este *framework* é composto por três elementos principais (Figura 14): **1) Construção da comunidade:** as organizações podem começar

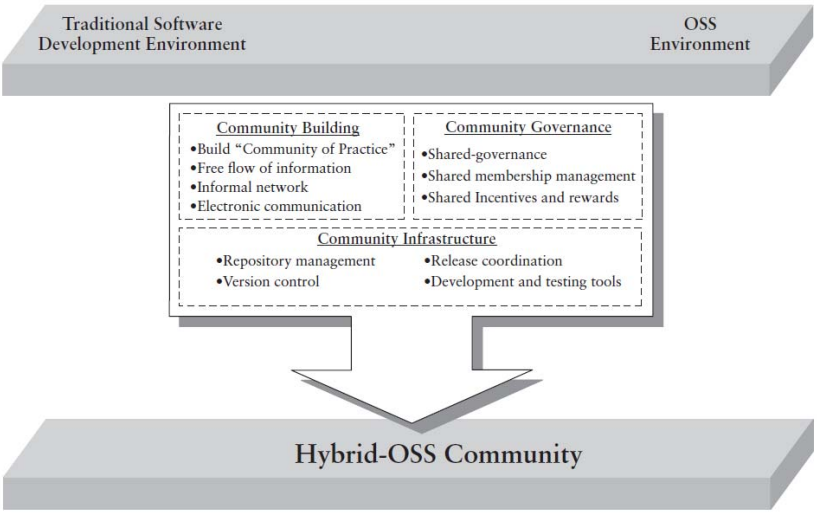
Informações da QPc		
		construindo uma comunidade de práticas que promova a livre troca de idéias e informações entre os seus participantes. Assim, será possível compartilhar o conhecimento e identificar oportunidades para inovações. Para estimular este comportamento, a organização terá que flexibilizar as suas estruturas formais e oferecer mecanismos para que os membros da comunidade possam realizar as suas tarefas através de relacionamentos informais.  Figura 14 – *Framework* para criar uma comunidade híbrida de desenvolvimento de software **2) Governança da comunidade:** • **Governança compartilhada:** devem ser implementados mecanismos de governança que sejam transparentes. Sem um senso de justiça, a motivação dos participantes pode diminuir. Os gerentes também deverão abrir mão da prática de impor uma estrutura de comando e controle centralizada. Os membros da comunidade devem poder trabalhar em grupos e devem receber poder para tomar decisões através de discussões e votações. Depois que a comunidade estiver construída, os seus membros devem poder escolher o que desejam fazer com base em suas habilidades, competências e conhecimentos. • **Gestão dos membros da comunidade:** a organização deve oferecer mecanismos para as pessoas participarem da comunidade e contribuírem com o projeto. Os membros devem poder ingressar ou sair da comunidade. • **Incentivos e recompensas:** as organizações devem recompensar e promover os membros que contribuírem para alcançar os objetivos da comunidade e da organização. A comunidade como um todo deve ser responsabilizada pelo trabalho realizado e deve ser recompensada ou penalizada coletivamente. Para este sistema funcionar, é necessário que a comunidade tenha um alto nível de confiança entre os seus membros. **3) Infraestrutura da comunidade:** as organizações devem fornecer as ferramentas necessárias e a infraestrutura para o desenvolvimento do software e gestão do projeto. Apesar da proposta do *framework*, os autores reconhecem que a transição para esta comunidade híbrida não é trivial. Historicamente, o desenvolvimento de software livre tem ocorrido em domínios horizontais focados na infraestrutura de software onde existem padrões de projeto. No caso das organizações, provavelmente elas vão precisar desenvolver este ambiente híbrido como foco em um domínio vertical que vai apresentar requisitos particulares. A possibilidade de uma organização migrar para um ambiente híbrido depende dos seguintes fatores: habilidade dos gerentes e funcionários de entender a filosofia do software livre; desenvolvimento de confiança mútua entre os gerentes e funcionários; percepção dos participantes de que estão envolvidos em um projeto novo e desafiante; motivação dos funcionários para participar destes projetos.

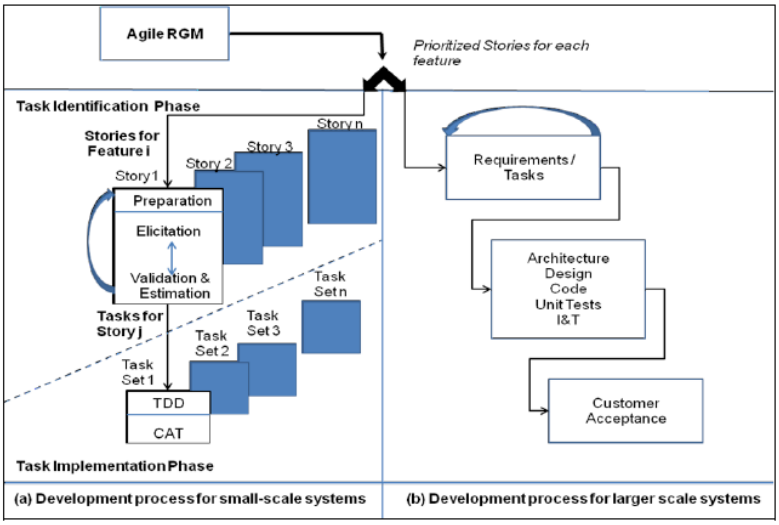
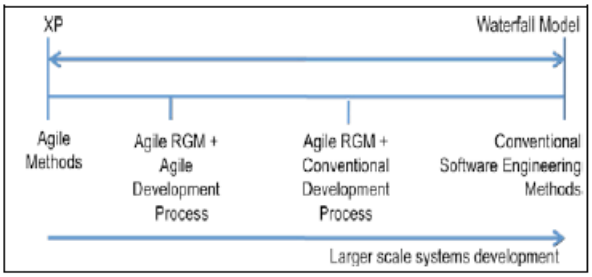
Tabela 20 – Informações extraídas dos artigos da QPc

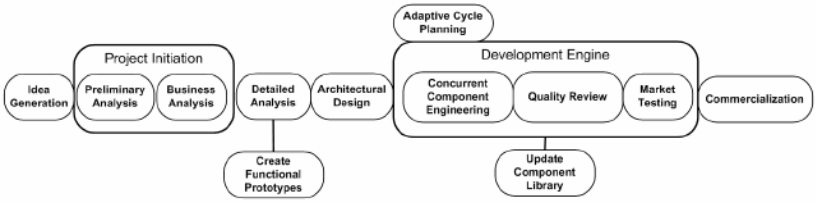
5.2.4. QPd

As informações extraídas do artigo da QPd são apresentadas pela Tabela 21.

Informações da QPd		
17.	**Título do documento:**	Applying Agile Principles for Distributed Software Development
	Autor(es):	R. Phalnikar; V.S. Deshpande; S.D. Joshi
	Data de publicação:	2009
	Fonte:	International Conference on Advanced Computer Control (ICACC)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS1	
	Oportunidades para a conciliação:	Geralmente, as organizações *offshore* favorecem o modelo tradicional, onde requisitos detalhados e projetos de *design* são enviados para serem implementados. Adaptar as práticas ágeis do SCRUM a um ambiente de desenvolvimento distribuído pode trazer benefícios e ajudar a vencer os desafios inerentes a este modelo de desenvolvimento. Os principais benefícios são: • A integração contínua de código pode reduzir ou eliminar os problemas de gerência de configuração. • O desenvolvimento iterativo, com entregas frequentes ao cliente, pode tratar o problema da perda de visibilidade do projeto, através do acompanhamento da quantidade de software sendo entregue.
	Desafios para a conciliação	Os desafios para a combinação de métodos ágeis com o desenvolvimento distribuído de software são: comunicação distribuída, diferenças de fusos horários, barreiras culturais e de idiomas e falta de interação direta com os especialistas do negócio.
	Informações da QS2	
	Descrição da proposta de conciliação:	Para conciliar o desenvolvimento distribuído com os métodos ágeis, os principais desafios em relação à estrutura da equipe do projeto são: • Sincronização completa do trabalho. • Distribuir o trabalho entre os membros da equipe por funcionalidade e não por disciplina para evitar equipes remotas de desenvolvimento ou testes. • Adotar a integração contínua do código. • Garantir boa largura de banda para que as equipes possam acessar o repositório de gerência de configuração sem latência. • Distribuir as equipes para aumentar os talentos e não para cortar os custos. • Novas equipes serão necessárias para integração, documentação e validação. • A equipe de documentação deve participar das reuniões de planejamento. • A equipe de integração é responsável pelas integrações noturnas e instalações. • A equipe de validação se encarrega dos testes e validação. • Gerentes de projeto são colocados em cada localidade. • Reuniões rotineiras ajudam a resolver as ambiguidades. • A equipe de uma localidade deve ficar situada próxima fisicamente para facilitar a comunicação. • A organização precisará acomodar horários de trabalho flexíveis para lidar com as diferenças de fusos horários de forma que a equipe *offshore* e a equipe *onshore* possam trabalhar em conjunto durante parte do tempo. Para resolver estes desafios, os autores propõem duas estruturas possíveis para a equipe do projeto:

Informações da QPd		
		1) Modelo 1 – Parcialmente *offshore* Neste modelo a equipe de design fica localizada *onshore* e envia os requisitos para a equipe *offshore*. A equipe *offshore* responde com o código implementado e testado. Então, a equipe *onshore* realiza os testes de validação com o cliente. O objetivo deste modelo é criar requisitos e uma arquitetura de qualidade já que mantém os analistas de negócio e os arquitetos próximos ao cliente, enquanto distribui as tarefas de nível mais baixo como a implementação e os testes. Este modelo sofre de expressivos e custosos problemas de comunicação **2) Modelo 2 – Totalmente *offshore*** Este modelo tenta deixar toda a equipe de desenvolvimento o mais próxima possível. A única pessoa no lado *onshore* é o cliente. O cliente envia as funcionalidades e a equipe *offshore* responde com o código funcionando e testado. Este modelo resolve os problemas de comunicação dentro da equipe, mas dificulta a comunicação do cliente com a equipe.
18.	**Título do documento:**	A Soft-Structured Agile Framework for Larger Scale Systems Development
	Autor(es):	S. Soundararajan; J.D. Arthur
	Data de publicação:	2009
	Fonte:	International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O desenvolvimento de sistemas em larga escala requer um processo que possa acomodar as mudanças. Porém, nem todas as práticas ágeis são adequadas. Assim, uma abordagem híbrida, que combine as vantagens das práticas ágeis e dos métodos estruturados, pode ser uma solução mais efetiva. Movidos por esta motivação, os autores propõem um *framework* para combinar os princípios dos métodos ágeis e tradicionais de desenvolvimento que trata os problemas das mudanças rápidas de requisitos para sistemas em larga escala. Este *framework* consiste de duas partes: uma abordagem para levantamento de requisitos que reflete a metodologia ágil (*Agile Requirements Generation Model*); e um processo de desenvolvimento customizado, direcionado a uma abordagem mais ágil ou mais convencional, que pode ser aplicado baseado no tamanho do sistema (Figura 15). Conforme ilustrado pela Figura 15, se o sistema a ser construído é pequeno, é proposta a decomposição das estórias previamente identificadas em tarefas e a implementação destas tarefas usando uma abordagem de desenvolvimento orientada a testes. Por outro lado, se o sistema que será construído é grande, as estórias priorizadas são implementadas de forma mais convencional. A Figura 16 mostra o *spectrum* das abordagens de engenharia de software para desenvolvimento de sistemas em larga escala. O *framework* proposto define um meio termo entre os métodos ágeis e as abordagens convencionais de engenharia de software. A ideia é que o *framework* possa oferecer estrutura aos métodos ágeis e, ao mesmo tempo, evite as limitações dos métodos convencionais. A principal contribuição deste *framework* é que ele reflete tanto os valores da comunidade ágil e os princípios declarados no Manifesto Ágil, quanto os valores da indústria. Porém, esta proposta ainda não foi validada em nenhum projeto.

Informações da QPd		
		 Figura 15 – Processo de desenvolvimento  Figura 16 – *Spectrum* das abordagens de engenharia de software para desenvolvimento de sistemas em larga escala
19.	**Título do documento:**	Designing an agile methodology for mobile software development: A hybrid method engineering approach
	Autor(es):	V. Rahimian; R. Ramsin
	Data de publicação:	2008
	Fonte:	International Conference on Research Challenges in Information Science (RCIS)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	Os requisitos e restrições associados ao desenvolvimento de sistemas móveis criaram novos desafios ao desenvolvimento tradicional para atender as necessidades deste ambiente. Apesar dos métodos ágeis serem considerados os mais adequados para o desenvolvimento móvel de aplicações, as especificidades deste tipo de desenvolvimento também demandam certos ajustes nos métodos ágeis. Motivados pela crença de que nenhuma metodologia atende todas as situações, a Engenharia de Metodologias foi introduzida como uma forma disciplinada de construir metodologias para atender as necessidades das organizações e dos projetos. Existem diversas abordagens para a Engenharia de Metodologias. Entre as opções existentes, os autores utilizaram a *Hybrid Methodology Design* (Projeto de Metodologia Híbrida). Com base nas necessidades do desenvolvimento móvel foram definidos os requisitos para uma metodologia híbrida de desenvolvimento de software móvel que aproveite algumas práticas ágeis combinadas com o desenvolvimento tradicional. A metodologia proposta foi criada com base na

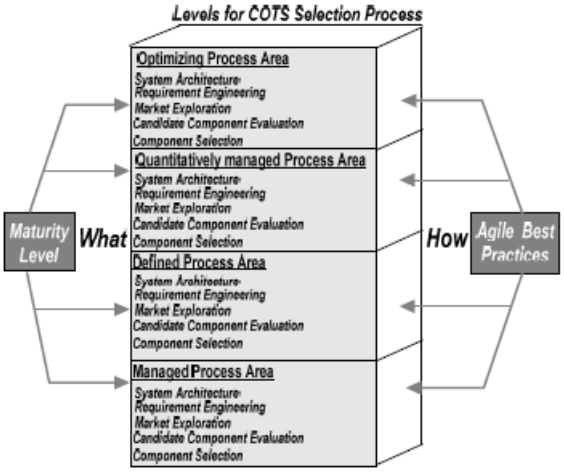
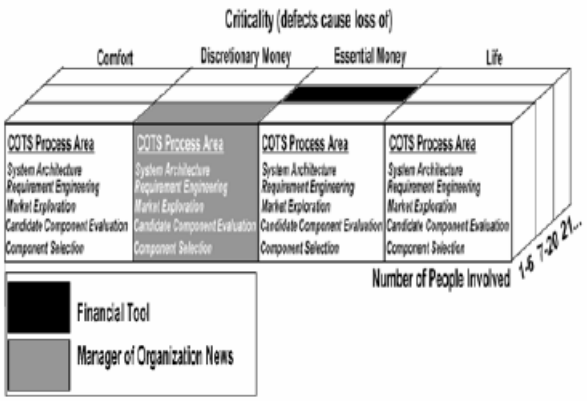
Informações da QPd		
		abordagem de engenharia de metodologias e aplicou ideias das abordagens ágeis ASD (*Adaptive Software Development*) e NPD (*New Product Development*). A Figura 17 resume as fases gerais da metodologia proposta.  Figura 17 – Fases gerais da metodologia proposta
20.	**Título do documento:**	Quantitative process improvement in XP using six sigma tools
	Autor(es):	S.I. Hashmi; J. Baik
	Data de publicação:	2008
	Fonte:	International Conference on Computer and Information Science (ICIS)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	A redução de defeitos no software é o objetivo principal de praticantes e pesquisadores da engenharia de software. Esta busca começou pelo advento da melhoria de processos, encarnada em modelos como o CMM e o CMMI, onde se acredita que ao ajustar os processos os defeitos do produto são minimizados e aumenta-se a satisfação do cliente. Uma das principais causas de defeitos no software é a dificuldade de entender e atender os requisitos do cliente. Para superar este desafio, foram introduzidos processos ágeis que são flexíveis o suficiente para acomodar os requisitos a qualquer momento durante o desenvolvimento do software. Historicamente, tem sido difícil combinar os processos ágeis com os modelos de maturidade de processos por três motivos: diferentes atividades dos métodos ágeis são realizadas ao mesmo tempo, então é difícil identificar as práticas de cada fase; os modelos de maturidade requerem uma grande quantidade de artefatos nas diferentes fases do ciclo de vida de desenvolvimento de software e que não são produzidos pelos métodos ágeis; os modelos de maturidade focam em uma grande estrutura organizacional, enquanto os processos ágeis são executados por pequenas organizações para o desenvolvimento de certos componentes e com um pequeno número de desenvolvedores trabalhando. O Six Sigma fornece um conjunto de ferramentas de análise quantitativa que pode ser útil para controlar o desempenho dos processos, através da análise dos dados do *Extreme Programming* (XP). A partir desta ideia, os autores mapearam as práticas do XP para as ferramentas do Six Sigma. Algumas das ferramentas do Six Sigma que podem ser usadas no XP são ferramentas de trabalho em equipe (diagrama de afinidade, SQFD, análise de Kano e SWFMEA). Já a modelagem de processos do Six Sigma parece ter menos aplicabilidade no XP, pois não existem processos complexos para serem mapeados. Outras ferramentas do Six Sigma aplicáveis ao XP são: análise de pareto, diagrama de causa e efeito, gráficos de controle, ANOVA, t-test, diagramas de dispersão, análise de correlação e análise de regressão. Alguns exemplos de uso de Six Sigma com XP são: • **Estimativa de duração:** as estimativas de duração do XP geralmente são feitas com base nos dados históricos sobre as estórias e na quantidade e natureza das estórias. O *t-test* pode ser usado para comparar as médias de duas distribuições (previsto e realizado). Para observar os efeitos de uma entrada em uma saída, podem ser usados os gráficos *box-and-whisker*. Além disso, potenciais fontes de variações

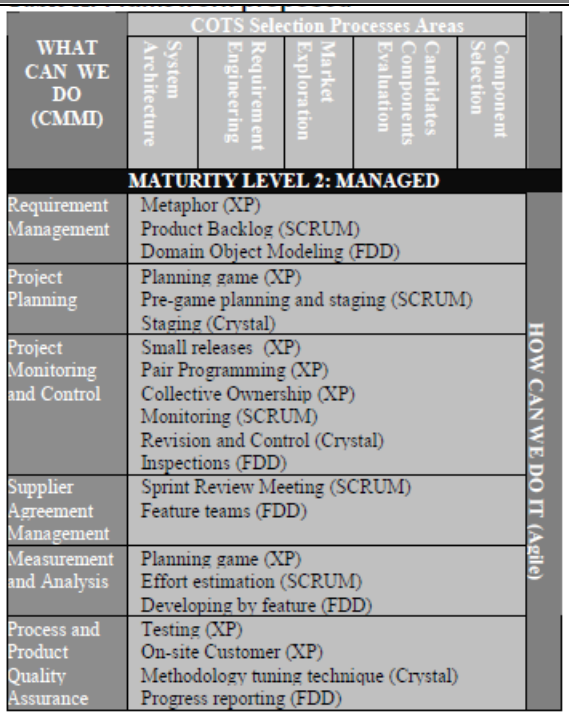
Informações da QPd		
		também podem ser identificadas por análises gráficas. A análise de regressão pode ajudar a verificar a validade dos dados históricos existentes. • **Efeito da duração de uma iteração na produtividade da equipe:** como a produtividade é uma grande preocupação do XP, é importante acompanhar a relação entre a duração de uma iteração e a produtividade alcançada pela equipe. O gráfico de probabilidade normal pode ajudar nesta análise.
21.	**Título do documento:**	Mature agile with a twist of CMMI
	Autor(es):	C.R. Jakobsen; K.A. Johnson
	Data de publicação:	2008
	Fonte:	Agile Conference
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS3	
	Estratégias adotadas pelas organizações:	Neste artigo, os autores apresentam a experiência da empresa Systematic sobre como o CMMI pode amplificar os métodos ágeis e recomendam um conjunto de atividades que um projeto ágil pode adotar do CMMI para melhorar o seu desempenho. Os defensores puristas dos métodos ágeis ou aqueles acostumados a aplicar os métodos ágeis em projetos pequenos, podem achar estas atividades improdutivas, mas em projetos grandes e distribuídos estas atividades são realmente necessárias e vão agregar valor. A Systematic é uma empresa certificada no nível 5 do CMMI que decidiu adotar o SCRUM. A ideia é que o CMMI contribua dizendo quais processos são necessários para manter uma organização madura e disciplinada, capaz de prever e melhorar o desempenho dos seus projetos. Por sua vez, o SCRUM fornece os guias para uma gestão efetiva dos projetos de forma flexível e adaptável. Ao combinar as duas abordagens, o que se imagina é que o SCRUM ajude a garantir que os processos estão implantados de forma eficiente nos projetos, enquanto acomodam as mudanças, e que o CMMI ajude a garantir que todos os processos relevantes foram considerados com a devida disciplina. Individualmente, tanto o CMMI quanto o SCRUM têm os seus benefícios e as suas falhas. Assim, a combinação de ambos pode ser benéfica para a organização nos seguintes aspectos: • O planejamento do CMMI pode ser considerado uma espécie de *sprint* zero disciplinado, onde um *framework* para o projeto é estabelecido, incluindo um *backlog* para o produto, a definição de uma linha de produção e o objetivo e visão geral do projeto. • A gestão de riscos do CMMI trata os possíveis impedimentos antes que eles sejam encontrados pela equipe. • O plano de qualidade do CMMI especifica de forma mais eficiente e acurada os objetivos de qualidade do projeto e ajuda os desenvolvedores a interpretarem os critérios de completeza e os objetivos do *sprint.* Ao explicitar o plano, a equipe também conhece o cronograma de garantia de qualidade e sabe quais atividades de garantia de qualidade serão utilizadas para garantir que os objetivos de qualidade serão atingidos. • O CMMI ajuda a garantir que o projeto como um todo será acompanhado, permitindo que a equipe se concentre no *sprint* atual, sabendo que eles periodicamente serão informados do status geral do projeto. • O SCRUM requer disciplina em relação aos testes automatizados e a integração do código e o CMMI pode atender esta necessidade de disciplina. • O CMMI espera que o projeto persiga metas objetivas de desempenho dos processos. No SCRUM o progresso dos *sprints* é medido nas reuniões de revisão. • O CMMI garante que os métodos ágeis serão institucionalizados,

Informações da QPd		
		incluindo a implementação consistente na organização, a melhoria contínua e o treinamento dos papéis da equipe do projeto. Ao combinar o CMMI com o SCRUM, pode-se perceber que o CMMI engloba o SCRUM, pois tem mais práticas e suporte ao planejamento inicial e ao fechamento e entrega do projeto. Nesta abordagem combinada, o planejamento do projeto é dividido em duas partes: um planejamento geral do projeto aos moldes do CMMI, com todos os planos de projeto, e um planejamento ágil detalhado, de acordo com o SCRUM, composto por uma lista do *backlog* do produto e das funcionalidades priorizadas para cada iteração (*sprint*). O acompanhamento do projeto também acaba sendo realizado nos dois níveis. O gerente do projeto faz o acompanhamento geral do projeto e a equipe do projeto acompanha o sprint. Quando a Systematic adotou o SCRUM, os papéis "Scrum Master", "Product Owner" e "Team" foram introduzidos na organização. Como todos os projetos da empresa têm um responsável que se comunica com o cliente, esta pessoa recebeu o papel de Product Owner. De acordo com a experiência da empresa, o Product Owner é quase sempre o gerente de projeto, mas também pode ser o arquiteto de software ou um engenheiro de software com experiência em lidar com o usuário. O Scrum Master é, em muitos casos, equivalente ao papel de líder de equipe já existente na empresa. Os projetos na Systematic são compostos por pessoas trabalhando em horário integral e a equipe fica toda situada próxima fisicamente. Quando a duração de um *sprint* é de um mês, é necessário investir na automação de todos os testes possíveis. A implementação da automação do teste é parte da própria atividade de implementação. Os testes automatizados são usados no repositório compartilhado da equipe de desenvolvimento e devem ser executados toda vez que um desenvolvedor introduzir um novo código no ambiente comum do projeto, para diminuir as chances da introdução de defeitos. Além disso, como parte da entrega de todo *sprint*, um produto de trabalho é avaliado e, para cada entrega ao cliente, uma configuração funcional é auditada para garantir que o *build* do produto está completa e correta. A experiência mostrou que esta prática ajuda a construir bons hábitos de uso do *build* pelos desenvolvedores que se disciplinam pelo uso da gerência de configuração.
22.	**Título do documento:**	Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course
	Autor(es):	L. Layman; L. Williams; K. Slaten; S. Berenson; M. Vouk
	Data de publicação:	2008
	Fonte:	International Journal of Engineering Education
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O artigo descreve um curso de engenharia de software, que educa os estudantes tanto nos métodos ágeis quanto nas abordagens orientadas ao planejamento, ensinando-os a compor uma metodologia de desenvolvimento apropriada ao projeto e à equipe de desenvolvimento, enfatizando a colaboração e o aprendizado. Os estudantes precisam de prática com as metodologias ágeis e com as metodologias orientadas ao planejamento. Além disso, as empresas sempre reforçam a necessidade dos alunos serem treinados em habilidades de comunicação e trabalho em grupo. Estas habilidades são demandadas por ambas as abordagens e somam-se a necessidade do estudante saber aplicar as técnicas apropriadas para um determinado projeto de desenvolvimento de software para que possa desenvolver um sistema de qualidade. Assim, o curso coloca o aluno para praticar técnicas, ferramentas e processos com os quais ele vai se deparar no desenvolvimento profissional

Informações da QPd		
		de software. Durante as primeiras nove semanas do curso, os alunos aprendem práticas de desenvolvimento de software independentes do processo de desenvolvimento. Por exemplo, os alunos aprendem três formas diferentes de documentar requisitos: documento de requisitos, caso de uso e estórias informais. A turma discute os prós e contras de cada uma das abordagens, o que dá aos alunos a base para decidir quais práticas aplicar em cada cenário. Em seguida, as leituras do curso ajudam o aluno a perceber que um processo de desenvolvimento de software consiste em um determinado agrupamento das práticas que ele aprendeu. Assim, um determinado conjunto de práticas compõe o modelo em cascata, enquanto outro conjunto com práticas distintas vai resultar no XP, por exemplo. Os alunos aprendem que as práticas são os tijolos dos processos e que o melhor processo para um determinado projeto ou equipe deve ser escolhido com base nas características específicas deste projeto ou equipe. Mais especificamente, os alunos aprendem a escolher entre uma metodologia ágil, uma metodologia orientada ao planejamento ou criar uma metodologia híbrida usando a abordagem orientada a riscos do Boehm.
23.	**Título do documento:**	Mapping CMMI project management process areas to SCRUM practices
	Autor(es):	A.S.C. Marçal; B.C.C. de Freitas; F.S. Furtado Soares; A.D. Belchior
	Data de publicação:	2007
	Fonte:	International Conference on Software Engineering (ICSE)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O artigo apresenta o mapeamento entre o CMMI e o SCRUM para mostrar como o SCRUM poderia tratar as áreas de processo de Gerência de projetos do CMMI. Este mapeamento leva em consideração a representação em estágios do CMMI e endereça cada prática das áreas de processo do CMMI com as práticas equivalentes do SCRUM. Os resultados mostraram que 31,1% das práticas específicas das áreas de processos de gerência de projetos do CMMI são satisfeitas pelo SCRUM, 16,4% são parcialmente satisfeitas e 52,5% não são satisfeitas. Assim, pode-se observar que o SCRUM não é totalmente aderente as práticas de gerência de projetos do CMMI, principalmente no que se refere à gestão de acordo com fornecedores, gestão de riscos e gerência quantitativa de projetos.
24.	**Título do documento:**	Mastering dual-shore development - The tools and materials approach adapted to agile offshoring
	Autor(es):	A. Kornstadt; J. Sauer
	Data de publicação:	2007
	Fonte:	International Conference on Software Engineering Approaches for Offshore and Outsourced Development (SEAFOOD)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	Claramente, a expectativa dominante nas organizações que recorrem ao *outsourcing* (no todo ou em parte) das suas áreas de TI é a redução de custos, que pode chegar até 90%. Entretanto, a prática de *outsourcing*

Informações da QPd		
		também enfrenta grandes desafios: altos custos de infraestrutura, comunicação, viagens e treinamento cultural; baixa produtividade devido à alta rotatividade da equipe no site *offshore* e devido à baixa moral da equipe no site *onshore*; problemas de gestão devido à diferença cultural e a um pobre compartilhamento de informações; problemas na comunicação com o cliente; e dificuldades técnicas. Diante destes desafios para o desenvolvimento de software *offshore*, modelos de processos que funcionam bem para projetos convencionais precisam ser adaptados a este cenário. Assim, este artigo apresenta uma proposta de extensão da abordagem *Tools & Materials* (T&M) para o desenvolvimento de projetos parcialmente *onshore* e parcialmente *offshore*. A abordagem T&M facilita o desenvolvimento de software fornecendo orientações em relação à arquitetura do software e ao processo de desenvolvimento de software. Ela é baseada na orientação a objetos e nos processos ágeis. Além disso, foca em dois aspectos da comunicação entre os *stakeholders*: frequência e precisão. A abordagem T&M já foi utilizada com sucesso em muitos projetos de desenvolvimento de software ágeis *single-site*. As extensões propostas são para projetos *dual-shore*. Estas propostas foram validadas em um primeiro estudo de caso com equipes *onshore* e *offshore*. A separação geográfica das equipes no modelo *dual-shore* impede que os desenvolvedores *offshore* tenham o cliente presente e que participem do planejamento das iterações. Diante deste cenário, os papéis são distribuídos de forma desigual entre as duas equipes. A equipe *onshore* possui analistas de negócio, arquitetos de software e desenvolvedores, enquanto a equipe *offshore* é composta inteiramente por desenvolvedores que recebem tarefas de implementação de componentes. Para corrigir essa desigualdade, as primeiras iterações devem ser realizadas com as equipes misturadas, de forma que eles possam desenvolver um conhecimento compartilhado do domínio do negócio e do processo de desenvolvimento. Esta prática também ajuda a estabelecer uma base de comunicação que persistirá ao longo do projeto. Mesmo que a abordagem T&M já tenha um foco na arquitetura, esta característica se torna mais crítica no desenvolvimento *offshore*. A comunicação da equipe também se beneficia da uniformidade da linguagem e uma base técnica comum. A definição da arquitetura também ajuda a designar tarefas que possam ser realizadas de forma independente pelas equipes distantes. Assim, o projeto pode ser organizado de acordo com a estrutura do produto, reduzindo a necessidade de coordenação entre equipes diferentes. As regras de arquitetura também podem ser definidas pelos arquitetos, divulgadas para todos os desenvolvedores e verificadas com ferramentas automatizadas. As estórias informais capturam apenas a essência dos requisitos. Os detalhes precisam ser entendidos com o cliente. Isso é difícil em um contexto *offshore* e aumenta a necessidade de comunicação. Para solucionar este problema, a abordagem T&P pode ser adaptada para trabalhar com desenvolvimento baseado em componentes. Seguindo esta estratégia, a equipe offshore começaria o desenvolvimento pelos componentes menores e não precisariam entender todo o domínio e a lógica do negócio logo no início. Os componentes vão sendo integrados em componentes mais complexos ou na própria aplicação pelos desenvolvedores mais experientes da equipe *onshore*.
25.	**Título do documento:**	Reconciling agility and discipline in COTS selection processes
	Autor(es):	F. Navarrete; P. Botella; X. Franch
	Data de publicação:	2007
	Fonte:	International IEEE Conference on Commercial-off-the-Shelf-Based Software Systems (ICCBSS)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	

Informações da QPd		
	Descrição da proposta de conciliação:	Atualmente os sistemas de informação são construídos através da integração e customização de componentes de prateleira. Porém, o processo de seleção de componentes ainda sofre por falta de maturidade e falta de agilidade. Por este motivo, a contribuição deste trabalho é propor um *framework* para conciliar as abordagens ágeis e disciplinadas, tirando vantagem dos benefícios de ambos, para a seleção de componentes de prateleira, incluindo as práticas ágeis no nível 5 do CMMI de aquisição (Figura 18).  Figura 18 – Proposta para conciliação com vistas à aquisição Para alcançar um equilíbrio entre as duas abordagens e regular o quanto de disciplina ou de agilidade precisa ser aplicado, os autores sugerem trabalhar com duas dimensões que influenciam qualquer metodologia de desenvolvimento: criticidade do sistema e número de pessoas na equipe do projeto (Figura 19).  Figura 19 – Dimensões para equilibrar disciplina e agilidade Para apoiar esta proposta de solução, foi criado ainda um *framework*, que segue a representação em estágios do CMMI, para organizar os níveis de maturidade para a seleção de componentes. Estes níveis de maturidade são compostos por práticas ágeis. Um exemplo de parte deste *framework* é apresentado na Figura 20.

Informações da QPd		
		

WHAT CAN WE DO (CMMI)	COTS Selection Processes Areas: System Architecture	Requirement Engineering	Market Exploration	Candidates Components Evaluation	Component Selection
MATURITY LEVEL 2: MANAGED					
Requirement Management	Metaphor (XP) Product Backlog (SCRUM) Domain Object Modeling (FDD)				
Project Planning	Planning game (XP) Pre-game planning and staging (SCRUM) Staging (Crystal)				
Project Monitoring and Control	Small releases (XP) Pair Programming (XP) Collective Ownership (XP) Monitoring (SCRUM) Revision and Control (Crystal) Inspections (FDD)				
Supplier Agreement Management	Sprint Review Meeting (SCRUM) Feature teams (FDD)				
Measurement and Analysis	Planning game (XP) Effort estimation (SCRUM) Developing by feature (FDD)				
Process and Product Quality Assurance	Testing (XP) On-site Customer (XP) Methodology tuning technique (Crystal) Progress reporting (FDD)				

HOW CAN WE DO IT (Agile)

Figura 20 – Exemplo do *framework*

26.	**Título do documento:**	Can agile and traditional systems development approaches coexist? An ambidextrous view
	Autor(es):	V. Vinekar; C.W. Slinkman; S. Nerur
	Data de publicação:	2006
	Fonte:	Information Systems Management
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	• **Mudança:** o desenvolvimento tradicional tenta minimizar as mudanças durante o andamento do projeto através de um processo rigoroso de gestão de requisitos. A intenção é atingir uma alta qualidade através de um cronograma controlado. Já os métodos ágeis assumem que as mudanças ao longo do processo de desenvolvimento de software são inevitáveis e necessárias. O foco é na adaptação e inovação ao invés de na predição e controle.
	Informações da QS1	
	Desafios para a conciliação:	Uma organização que tenta utilizar as duas abordagens de desenvolvimento em projetos diferentes, enfrenta diversos desafios para sustentar duas culturas tão distintas. Estes desafios ocorrem em quatro níveis diferentes: gestão e organização, pessoas, processos e tecnologia. Os níveis gestão e organização e pessoas impõem desafios significativos a existência simultânea do desenvolvimento ágil e tradicional nas organizações que precisam enfrentar o dilema da busca pela otimização dos processos enquanto preparam o ambiente para poder responder rapidamente as mudanças. • **Gestão e organização:** as culturas organizacionais, os estilos de gestão, a estrutura organizacional e o sistema de recompensa do desenvolvimento ágil e tradicional são conflitantes. Estruturas organizacionais que encarnam uma cultura de controle hierárquico vão encontrar uma dificuldade especial em acomodar algumas das características do desenvolvimento ágil, como a auto-organização da equipe, a tomada de decisão descentralizada e o ambiente colaborativo. Os sistemas de recompensas também precisam ser revistos para encorajar o trabalho em grupo e os objetivos coletivos e não mais focar

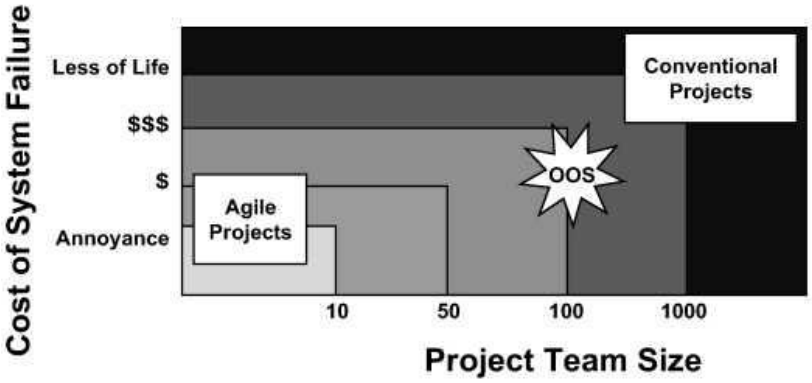
	Informações da QPd	
		as avaliações apenas nos indivíduos. • **Pessoas:** o desenvolvimento ágil enfatiza a dinâmica do trabalho em grupo. Os papéis dos membros do grupo são flexíveis e a auto-organização da equipe é uma característica chave. O papel tradicional de gerente de projeto como um planejador e controlador é substituído pelo papel de facilitador que gerencia o esforço colaborativo da equipe, mas sem prejudicar a criatividade. Os processos devem ser flexíveis e dinâmicos o suficiente para se adaptar as competências e habilidades dos indivíduos. Este foco nas pessoas distingue bastante o desenvolvimento ágil do tradicional onde o foco está nos processos.
	Informações da QS2	
	Descrição da proposta de conciliação:	A adoção do desenvolvimento ágil está crescendo na indústria. Apesar da maioria das organizações que adotaram os métodos ágeis considerarem que eles melhoraram a produtividade e a qualidade, estas mesmas organizações também acham que outras metodologias são necessárias. As evidências parecem indicar que muitas organizações de desenvolvimento de software estão tentando utilizar a abordagem ágil junto com a tradicional, motivadas pela necessidade de balancear interesses conflitantes – agilidade e estabilidade. Estas evidências sugerem que o conceito de ambivalência (originado do trabalho de Duncan – 1976), que permite a busca simultânea por abordagens contrastantes, é uma solução efetiva e viável para este dilema das organizações entre agilidade e estabilidade. Assim, os autores propõem uma forma de organização ambivalente para conciliar o desenvolvimento de software tradicional e o ágil, preservando as culturas organizacionais necessárias a cada um deles, através da separação dos modelos de desenvolvimento em unidades organizacionais distintas subordinadas a uma gerência maior comum. Esta gerência maior ficaria então encarregada de avaliar a adequação de cada uma das abordagens a cada novo projeto que surgisse na organização e delegar o projeto a subgerência encarregada. Além disso, esta gerência maior também seria capaz de aprender com o sucesso e falhas de ambas as subgerências e modificá-las de acordo com o necessário. Os indivíduos mais tolerantes a ambiguidade e aqueles que trabalham melhor em equipe, podem ser parte da unidade de desenvolvimento ágil. Já os gerentes e funcionários acostumados com uma estrutura hierárquica que enfatiza o comando e o controle, a especialização de papéis, o trabalho individual e um alto grau de formalização podem se sentir mais confortáveis na unidade de desenvolvimento tradicional.
27.	**Título do documento:**	Future implementation and integration of Agile methods in software development and testing
	Autor(es):	W.H. Kee
	Data de publicação:	2006
	Fonte:	Innovations in Information Technology (IIT)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS1	
	Desafios para a conciliação:	Abaixo são apresentados alguns desafios para a integração dos métodos ágeis com o processo de desenvolvimento tradicional e as respectivas propostas de solução: **1) Requisitos ambíguos:** algumas declarações de requisitos ambíguas poderão ser produzidas se o cliente e o desenvolvedor falharem na colaboração nos estágios iniciais do desenvolvimento. Este risco aumenta no desenvolvimento ágil, pois os requisitos são escritos através de cartões de estória que dependem de comunicação face-a-face para resolver as dúvidas de interpretação. • **Dois níveis de requisitos:** o primeiro nível de requisitos corresponde a estória dos usuários e delimita o escopo geral do projeto. O segundo nível de requisitos é desenvolvido de forma colaborativa com o cliente

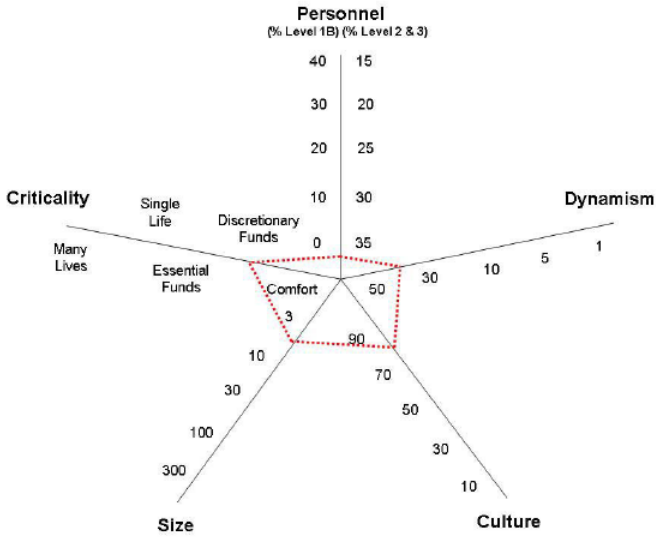
Informações da QPd		
		para detalhar o trabalho da próxima iteração. • **Pessoa-ponte:** esta pessoa participa das reuniões de equipe e ajuda a registrar as discussões, ajudando assim a manter uma documentação simples do projeto que será útil tanto para a equipe de desenvolvimento quanto para o cliente. **2) Projetos grandes e distribuídos:** em projetos grandes e distribuídos, o cliente não pode estar presente todo o tempo e a falta de comunicação com o cliente pode prejudicar o andamento do projeto. • **Representante do cliente:** alguém com experiência no domínio do negócio e que possa atuar como representante do cliente para manter o ritmo e a velocidade normais do projeto, respondendo as perguntas rapidamente ou contactando os especialistas necessários. **3) Conflitos do processo de desenvolvimento:** os conflitos surgem com a necessidade de combinar os métodos ágeis com o desenvolvimento tradicional sem comprometer a agilidade e nem o processo da organização. • **Documentar e comunicar o processo:** analisar o processo existente e o processo proposto, documentar os termos-chave, papéis e responsabilidades e apresentar o processo adaptado a todas as partes envolvidas para que todos possam compreender os riscos e soluções resultantes da combinação dos dois processos. **4) Conflitos pessoais:** a mudança de paradigma pode não ser aceita pela equipe de desenvolvimento e/ou pelo cliente. • **Base de conhecimento e construção de equipe:** os métodos ágeis colocam mais ênfase na competência da equipe do projeto. Assim, para integrar os dois paradigmas de desenvolvimento, uma base de conhecimento pode ajudar a aumentar o conhecimento dos membros da equipe do projeto. Além disso, os métodos ágeis precisam que os membros da equipe tenham um objetivo comum, confiança e respeito mútuos e um processo de tomada de decisão colaborativo e rápido. Assim, atividades de construção de equipe são muito importantes para melhorar a comunicação, a colaboração e o relacionamento entre os membros da equipe.
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	Motorola Wireless Systems Group (WSG) começou a adotar gradualmente os métodos ágeis e já conseguiu resultados expressivos em alguns projetos, em relação à redução de defeitos e melhoria na produtividade.
28.	**Título do documento:**	Formalizing agility, part 2: How an agile organization embraced the CMMI
	Autor(es):	S.W. Baker
	Data de publicação:	2006
	Fonte:	Agile Conference
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS3	
	Estratégias adotadas pelas organizações:	Muitas organizações precisam do melhor dos dois mundos: um software funcionando que apóie as mudanças frequentes e as necessidades do negócio; e um processo para entregar o software que seja previsível, treinável e auditável. A empresa DTE Energy conseguiu esta combinação: uma metodologia leve e ao mesmo tempo compatível com o CMMI. Em 1998 uma parte da organização começou a utilizar métodos ágeis para adotar ciclos curtos de desenvolvimento e um desenvolvimento iterativo e incremental. Em 1999, movida pelo desejo de melhorar também o processo de desenvolvimento de software, outra parte da empresa começou um programa para definir os seus processos de acordo com o CMM. A partir daí, o trabalho dos 2 grupos progrediu, mas apesar de cada um ter o seu valor individualmente, para a organização era importante reconciliar as duas iniciativas em busca da melhoria contínua. O grupo responsável por esta missão de melhoria de processos decidiu adotar a própria metodologia ágil, já utilizada na empresa, na iniciativa de

Informações da QPd		
		melhoria de processos. O projeto de melhoria de processos foi tratado como um projeto ágil e dividido em iterações. O CMMI contribuiu ajudando a definir os requisitos que precisariam ser atingidos para uma avaliação futura e também com o método de avaliação (SCAMPI). A iteração 0 se dedicou ao planejamento do projeto e a organização da equipe. Além disso, nesta iteração foi feita uma pré-avaliação cujos resultados serviram como insumo para guiar as próximas iterações. A iteração 1 focou as áreas de processo do nível 3 do CMMI e fez uma avaliação informal. Na iteração 2 foram tratados os defeitos da iteração 1 e mais as áreas de processo de nível 2 do CMMI. Por fim, a iteração 3 abordou outras áreas de processo dos níveis 2 e 3 do CMMI e mais os defeitos identificados nas iterações anteriores. Na iteração 1, a empresa se deparou com o dilema de tentar manter a agilidade do projeto de melhoria de processos e dos projetos de desenvolvimento de software, enquanto tentava atender aos requisitos de documentação do CMMI. Assim, eles observaram que, na verdade, o CMMI não exigia nenhum artefato de trabalho em particular, mas sim evidências de que os processos estão realmente sendo executados. Em seguida, a organização também se deu conta que apenas definir os processos e artefatos de trabalho não seria o suficiente para que eles fossem utilizados pelas equipes de desenvolvimento, pois também era necessário que estas equipes conhecessem esta documentação e soubesse como utilizá-la. Assim era necessária uma comunicação e um material de treinamento efetivos para sustentar a melhoria de processos. As áreas de processos foram divididas em quatro mini-projetos tratados por equipes distintas que receberam a missão de identificar o estado atual da área de processo, mapeá-lo em um fluxograma simples, inventariar os produtos de trabalho já existentes, identificar os *gaps* utilizando o CMMI como modelo de referência e fazer uma proposta do mínimo necessário para cobrir os *gaps*. Devido às interfaces entre as áreas de trabalho, os desafios de comunicação e sincronização entre os produtos de trabalho, entre as diferentes equipes logo começaram a aparecer. Para resolver este problema, nas iterações seguintes foram feitos eventos externos com todos os membros das equipes envolvidas para construir em conjunto uma visão geral do processo que trate as interdependências entre as diferentes áreas de processo. Os processos e produtos de trabalho foram avaliados em cinco projetos piloto que também serviram para preparar a equipe interna de avaliação. A implantação da área de processo de gerência de configuração ajudou a adaptar quais produtos de trabalho seriam criados em cada projeto de desenvolvimento para que o escopo pudesse ser definido e o projeto devidamente planejado. Em relação ao fluxo do processo, a organização optou por adotar um fluxo de processo genérico, metodologicamente neutro, que utilizava o processo ágil como base e assinalava os pontos de variação quando necessário. Além disso, foi disponibilizado um conjunto de guias para a adaptação do processo. Como as ferramentas são facilitadores e não substitutas dos processos, até a iteração 2, a empresa não investiu na implantação de ferramentas que pudessem acelerar a execução do processo, pois desejava entender e institucionalizar o processo em si antes de automatizá-lo. Somente a partir da iteração 3, a empresa começou a explorar o potencial de uso de algumas ferramentas que poderiam automatizar alguns aspectos do processo, como a rastreabilidade de requisitos e a gerência de configuração de software.
29.	**Título do documento:**	The role of extreme programming in a plan-driven organization
	Autor(es):	H. Dahlberg; F.S. Ruiz; C.M. Olsson
	Data de publicação:	2006
	Fonte:	Springer
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Os métodos orientados a planejamento podem ser caracterizados como sendo aqueles em que o trabalho começa com a elicitação e documentação

Informações da QPd	
	do conjunto completo de requisitos, seguido pela definição arquitetural, um projeto de alto nível e o desenvolvimento. Por outro lado, os métodos ágeis defendem a especificação incremental dos requisitos, uma quantidade mínima de documentação e o foco nos indivíduos e interações entre eles. As abordagens orientadas ao planejamento geralmente procuram reduzir os riscos investindo em ciclos de vida, arquitetura e planos de longo prazo. Estas abordagens são mais adequadas quando se consegue determinar os requisitos previamente e quando os requisitos são relativamente estáveis. Entretanto, quando os requisitos mudam com frequência vai se tornando difícil manter os requisitos completos, consistentes, testáveis e rastreáveis. Apesar de ser vital ter uma boa documentação quando se está desenvolvendo um software crítico, esta mesma burocracia pode não ser eficiente em projetos menores. Em contrapartida, os aspectos centrais dos métodos ágeis são velocidade e simplicidade.
Informações da QS1	
Desafios para a conciliação:	Questões principais da mudança do desenvolvimento de software orientado ao planejamento para as abordagens ágeis: **1) Gestão e organização** • **Cultura organizacional:** quando se planeja mudar os processos dentro de uma organização é importante levar em consideração o impacto na cultura organizacional. • **Estilo de gestão:** existe uma perda de autoridade, pois o estilo de gestão muda do comando e controle para liderança e colaboração. • **Estrutura organizacional:** estruturas organizacionais burocráticas e formais são um obstáculo a implantação dos métodos ágeis. • **Gestão do conhecimento:** a melhoria de processos deve enfatizar a criação de conhecimento. No desenvolvimento tradicional, o conhecimento é explicitado através da documentação. Como os métodos ágeis incentivam a simplicidade, grande parte do conhecimento permanece tácito na mente dos membros da equipe de desenvolvimento, mas é compartilhado por toda a equipe. • **Reconhecimento e recompensa:** o sistema de reconhecimento e recompensa da empresa precisa ser revisto, de forma a estimular a comunicação e o trabalho em equipe, para que os métodos ágeis possam ser implantados com sucesso. **2) Pessoas** • **Capacidade de trabalho em equipe:** trabalhar de forma ágil significa ser parte de uma equipe, onde a comunicação e a interação social têm um papel importante. As ideias de aprendizado compartilhado, workshops de reflexão, programação em pares e tomada de decisão colaborativa podem ser difíceis para programadores acostumados a trabalhar individualmente. • **Alto nível de competência:** o desenvolvimento ágil depende da competência dos membros da equipe mais fortemente do que o desenvolvimento tradicional onde o desenvolvimento vai apensar seguir o conhecimento que já está todo documento e revisado. • **Relacionamento com cliente:** quando uma organização trabalha para melhorar os seus processos de desenvolvimento de software, o cliente é fortemente afetado. Assim, também são necessárias ações para melhorar o relacionamento com o cliente. **3) Processos** • **Mudança de centrado em processos para centrado em pessoas:** os métodos ágeis se baseiam nas pessoas e na sua criatividade ao invés de processos e mudar esse paradigma é um grande desafio. • **Pequeno e iterativo:** a prática de pequenas *releases* do XP não é fácil de aplicar se o cliente não tem tempo para testá-las. **4) Tecnologia** • **Adequação das tecnologias e ferramentas existentes:** ferramentas desempenham um papel fundamental na implantação de uma metodologia de desenvolvimento de software. Assim, a tecnologia existente na organização pode impactar o esforço de migração para uma metodologia ágil. As organizações que planejam adotar metodologias ágeis devem investir em ferramentas que apóiem o desenvolvimento rápido, gerência de configuração e refatoração. Os funcionários devem ser treinados para usar as ferramentas**.**
Informações da QS3	

Informações da QPd		
	Estratégias adotadas pelas organizações:	Apesar do desenvolvimento de software orientado ao planejamento continuar sendo a abordagem mais utilizada, geralmente as organizações acabam fazendo adaptações *ad-hoc* ou improvisações nos seus processos que se afastam do que prescrevem os modelos. Por outro lado, a mudança da abordagem orientada ao planejamento para uma abordagem ágil geralmente envolve refinar os processos para identificar aqueles mais adequados à organização. O artigo apresenta o relato de experiência da Volvo Technology que não queria se afastar completamente dos seus processos, mas sim adaptá-los de forma que também pudesse utilizar o *Extreme Programming* (XP) no seu dia-a-dia de trabalho. Assim, seria possível manter a credibilidade comumente associada aos métodos orientados ao planejamento e, ao mesmo tempo, flexibilizar os produtos de trabalho para responder melhor às mudanças. Como a Volvo pretendia ser avaliada segundo a ISO/IEC 15504 e ainda poder trabalhar com o XP, a empresa buscou por uma estratégia que permitisse adotar a abordagem ágil em um contexto orientado ao planejamento. Esta experiência permitiria identificar os efeitos que esta combinação teria na organização. **1) Gestão e Organização** • **Cultura organizacional:** as culturas tradicionais tendem a ser mais orientadas ao processo, enquanto as culturas ágeis são mais sociais e orientadas a equipe. Como a melhoria de processos é uma iniciativa de médio a longo prazo, é necessário um forte apoio gerencial para mudar a forma como as pessoas estão acostumadas a trabalhar. • **Estilo de gestão:** na abordagem híbrida adotada pela Volvo optou-se por um estilo de gestão sem um forte controle de cada passo do processo. Ao invés disso, existem líderes de desenvolvimento mais experientes que apóiam a equipe de desenvolvimento e que contam com a confiança tanto dos gerentes quanto dos próprios desenvolvedores. • **Estrutura organizacional:** a Volvo é uma organização adequada para usar métodos ágeis, pois possui capacidade de adaptação e conhecimento, ao invés de ser totalmente burocrática e formal. • **Gestão do conhecimento:** na Volvo a prioridade é entregar o software funcionando antes da documentação. O conhecimento é tácito e comunicado entre os membros da equipe através da socialização. **2) Pessoas** • **Capacidade de trabalho em equipe:** na Volvo existem profissionais fortemente capacitados que costumam compartilhar ideias e defendê-las. Na empresa também existe uma atitude positiva em relação a programação em pares, mas a refatoração e a propriedade compartilhada do código precisam ser introduzidas com cuidado. Um grande desafio para a implantação do XP na Volvo é o horário flexível de trabalho que conflita com a necessidade de ter toda a equipe presente no mesmo horário de forma que práticas como a programação em pares e a refatoração possam funcionar. • **Alto nível de competência:** a Volvo possui uma equipe com um alto nível de experiência e formação profissional. • **Relacionamento com cliente:** entre os clientes da Volvo, o grau de competência varia e a maioria tem limitações de tempo que impedem o seu envolvimento integral com o projeto. **3) Processos** • **Mudança de centrado em processos para centrado em pessoas:** abandonar o modelo de desenvolvimento orientado ao planejamento, poderia trazer prejuízos para a Volvo, por ser uma empresa com uma longa tradição de trabalhar de acordo com os padrões e com processos de garantia da qualidade. Assim, o esforço de mudança de paradigma também precisava levar esse objetivo de negócio em consideração. • **Pequeno e iterativo:** um problema identificado nos processos de desenvolvimento da Volvo é a demora no *feedback* do cliente, o que acaba tornando este *feedback* irrelevante e incompatível com a ideia de ciclos curtos de desenvolvimento. • **Gestão de projetos grandes e escaláveis:** a agilidade, em geral, está associada a equipes de desenvolvimento pequenas ou médias e localizadas próximas fisicamente. Os projetos de desenvolvimento da Volvo atualmente têm equipes com 4 a 15 desenvolvedores. Este tamanho reduzido faz com que o uso de métodos ágeis seja adequado em termos de tamanho. • **Seleção do método ágil apropriado:** na Volvo, a principal

Informações da QPd		
		preocupação em relação à seleção de um método ágil era manter o nível de garantia da qualidade e a possibilidade de avaliação na ISO/IEC 15504. **4) Tecnologia** • **Adequação das tecnologias e ferramentas existentes:** a linguagem de programação adotada na Volvo é C e ela não é usada estritamente orientada a objetos, o que pode ser uma dificuldade inicial para a adoção da metodologia ágil.
30.	**Título do documento:**	Army simulation program balances agile and traditional methods with success
	Autor(es):	J. Surdu; D.J. Parsons
	Data de publicação:	2006
	Fonte:	CrossTalk
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	Métodos tradicionais são orientados a otimização, previsibilidade e controle. Métodos ágeis focam em adaptação a mudança, flexibilidade e inovação. A nova arte do desenvolvimento de software é encontrar o ponto de balanceamento apropriado entre as práticas disponíveis. Para alcançar este balanceamento, é proposto um *spectrum* que indica a relação entre os projetos tradicionais e ágeis, baseada no tamanho da equipe e nos custos de falha do sistema Figura 21.  Figura 21 – *Spectrum* por tipo de projeto considerando custo do defeito e tamanho da equipe
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	Incluído neste *spectrum* está um programa de simulação do exército americano, denominado OOS. O OOS foi desenvolvido levando em consideração aspectos de métodos ágeis, como o XP, assim como estratégias tradicionais. Nenhuma dessas abordagens foi seguida como uma receita de bolo. Pelo contrário, apenas as práticas que pareceram adequadas foram implementadas. Ainda existe espaço para algumas melhorias e o processo de melhoria contínua do CMMI nível 5 vai se encarregar dessas oportunidades. Porém, não é possível afirmar quais práticas foram mais benéficas e se elas poderiam ser usadas com o mesmo sucesso em outros projetos, pois a escolha da prática mais adequada depende da natureza do projeto.
31.	**Título do documento:**	Experiences using Agile Software Development for a Marketing Simulation
	Autor(es):	D. Mills; L. Sherrell; J. Boydstun; Guoqing Wei
	Data de publicação:	2006

	Informações da QPd	
	Fonte:	IEEE Southeast Conference
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	Selecionar uma metodologia é uma tarefa importante e difícil, mas já existem algumas ferramentas para ajudar neste processo de tomada de decisão, como a abordagem orientada a riscos de Boehm que utiliza um gráfico com cinco eixos radianos para representar os fatores críticos do projeto. De acordo com a posição dos pontos, o gráfico indica se o projeto está mais próximo do desenvolvimento ágil ou do desenvolvimento tradicional de software. O artigo apresenta a experiência dos autores no uso da proposta do Boehm de *home-ground* para determinar qual abordagem adotar no desenvolvimento de uma ferramenta de simulação de compras para estudos de marketing. O gráfico do projeto de Marketing Internacional é apresentado na Figura 22 e claramente favorece a abordagem ágil.  Figura 22 – Gráfico de *home-ground* do projeto de Marketing Internacional Após a decisão de utilizar uma metodologia ágil, o próximo passo é escolher qual abordagem ágil aplicar. No final, a equipe de desenvolvimento optou pelo XP por ser uma metodologia bem conhecida e bem documentada, devido à experiência prévia de alguns membros da equipe e a prática de releases frequentes também se encaixava bem com as necessidades do cliente.
32.	**Título do documento:**	Management challenges to implementing agile processes in traditional development organizations
	Autor(es):	B. Boehm; R. Turner
	Data de publicação:	2005
	Fonte:	IEEE Software
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica

Informações da QPd		
	Informações da QS1	
	Desafios para a conciliação:	Desenvolvedores e gerentes de projeto acreditam que as práticas ágeis estão mais alinhadas às necessidades de mudanças contínuas e desenvolvimento rápido da indústria, mas se sentem frustrados pela dificuldade de integrá-las ao desenvolvimento tradicional já existente nas organizações. Em uma série de workshops anuais, levantou-se uma lista com aproximadamente 40 obstáculos a essa implementação, mas alguns destes obstáculos depois foram percebidos como não sendo problemas reais. Estes obstáculos podem ser agrupados em três categorias, que resumem os principais desafios para incorporar as práticas ágeis nos projetos de desenvolvimento de software das organizações: **1) Conflitos do processo de desenvolvimento** • **Variabilidade:** as organizações precisam prestar atenção especial na sincronização de equipes trabalhando com processos distintos (tradicional e ágil) e desenvolvendo software para o mesmo produto. • **Diferentes ciclos de vida:** o desenvolvimento ágil foca em entregar funcionalidade imediatamente, enquanto o desenvolvimento tradicional foca em otimizar o desenvolvimento em um período mais longo. • **Sistemas legados:** aplicar os processos ágeis aos sistemas legados, tanto para manutenções quanto para novos desenvolvimentos, traz numerosos problemas, pois estes sistemas, em geral, não são fáceis de refatorar para que possam ser desenvolvidos em incrementos. • **Requisitos:** no desenvolvimento ágil, os requisitos tendem a ser predominantemente funcionais e informais. Isso nem sempre funciona na verificação e validação do sistema. Então, pode ser necessária acrescentar informações adicionais a este processo de gestão de requisitos ágil. Para contornar estes desafios são apresentadas as seguintes sugestões: • Realizar uma preparação prévia, conduzindo uma análise dos processos já existentes, dos requisitos e expectativas para os novos processos. • Construir os processos de baixo para cima, com base nas necessidades dos projetos, ao invés de tentar adaptá-los de cima para baixo. • Definir funcionalidades ou responsabilidades: deixar claro quais casos serão tratados com desenvolvimento ágil. • Desenvolver arquiteturas compartimentalizadas de forma que as equipes ágeis e tradicionais possam ser divididas e que cada uma possa trabalhar na parte da arquitetura mais relacionada com as suas características. • Redefinir os marcos de revisão tradicionais para se encaixar em uma abordagem iterativa. • Iniciar a implementação pelas práticas ágeis que apóiem os processos existentes e as prioridades organizacionais. • Avaliar os riscos para determinar o quanto de agilidade é necessária. **2) Conflitos do processo de negócio** • **Recursos humanos:** os membros das equipes de desenvolvimento ágil geralmente atravessam os limites dos seus papéis e precisam de mais habilidades e experiências para desempenhar o seu papel adequadamente. A organização precisa aprender a acomodar estas diferenças. • **Medição do progresso:** os contratos, marcos e avaliações tradicionais podem ser inadequados para acompanhar o passo rápido dos processos ágeis. Para estes casos, novas métricas ágeis (como o número de requisitos ou funções do *backlog* que foram implementadas) podem substituir as métricas tradicionais. • **Certificação dos processos:** uma preocupação das organizações mais maduras é saber como os processos ágeis vão afetar as suas certificações ISO ou CMMI. Apesar dos métodos ágeis estarem alinhados com o conceito de melhoria contínua do nível 5 do CMMI, muitos dos métodos ágeis realmente não atendem as exigências de documentação e infraestrutura destes modelos, pois isso poderia torná-los menos efetivos. Para contornar estes desafios são apresentadas as seguintes sugestões, sendo algumas delas oportunidades de pesquisa: • Realizar um projeto piloto e testar o seu impacto nos processos tradicionais. • Desenvolver práticas de gestão e arquiteturais que sejam adequadas

Informações da QPd		
		para abordagens híbridas • Atualizar as práticas contratuais para apoiar os conceitos ágeis • Identificar incompatibilidades e compatibilidades entre os modelos e trabalhar para eliminar, na medida do possível, as incompatibilidades e para encorajar as sinergias. • Conduzir estudos empíricos sobre quais classes de mudanças são mais imprevisíveis e, portanto, mais adequadas aos métodos ágeis e quais são mais adequadas ao desenvolvimento tradicional. • Pesquisar como modificar os sistemas legados para permitir e facilitar reengenharia, manutenção, substituição e extensão de acordo com os métodos ágeis. • Estabelecer guias para avaliação de maturidade de processos de forma compatível com os métodos ágeis. **3) Conflitos pessoais:** os obstáculos relacionados aos fatores humanos são os mais críticos e precisam ser tratados para que as práticas ágeis possam ser incorporadas aos processos das organizações. • **Atitudes de gestão:** os gerentes tendem a associar os empregados a papéis específicos o que pode causar alguma dificuldade diante da característica de multi-tarefas das equipes de desenvolvimento ágil. O papel do gerente de projeto também muda e ele passa a ser um protetor e um orientador. O protetor age como uma barreira para minimizar as perturbações durante o ciclo de desenvolvimento do projeto, enquanto o orientador fornece ajuda técnica quando necessário. • **Questões de logística:** equipes ágeis precisam ficar fisicamente próximas e com um espaço de trabalho preparado para a programação em pares, o compartilhamento de informações, os quadros de mensagens, a integração contínua e os testes de regressão. • **Projetos piloto de sucesso:** após o piloto tomar o cuidado para não separar a equipe, pois isso destrói os seus relacionamentos técnicos e pessoais, dilui o conhecimento adquirido e as lições aprendidas e ainda passa uma mensagem para o restante da equipe que pode ser perigoso tentar aprender coisas novas. • **Gestão da mudança:** as resistências a mudanças precisam ser gerenciadas para evitar comportamentos destrutivos. Atenção também será necessária na preparação dos clientes para o novo papel que eles terão que desempenhar durante o desenvolvimento do software. Para contornar estes desafios são apresentadas as seguintes sugestões: • Entender como a comunicação acontece dentro da equipe de desenvolvimento. • Educar os *stakeholders*. • Enfatizar os valores do desenvolvimento ágil tanto para a equipe de desenvolvimento quanto para os clientes. • Manter os bons profissionais e recompensar os bons resultados dos projetos piloto. • Rever os sistemas de recompensa para reconhecer tanto as contribuições individuais quanto as contribuições da equipe. Obstáculos técnicos também existem, mas as organizações podem superá-los mais facilmente com trabalho e paciência.
33.	**Título do documento:**	Stretching agile to fit CMMI level 3 - the story of creating MSF for CMMI process improvement at Microsoft corporation
	Autor(es):	D.J. Anderson
	Data de publicação:	2005
	Fonte:	Agile Conference
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	O autor interpreta o desenvolvimento ágil de software como tendo em sua essência a confiança – nos desenvolvedores – para construir o software corretamente – e com os clientes – através de entregas frequentes e *feedback*. Por outro lado, a indústria aeroespacial e de defesa são partes do governo onde falta confiança, a verificação se baseia em auditorias, aprovações são necessárias com frequência ao longo do ciclo de vida de desenvolvimento e os artefatos precisam de rastreabilidade bidirecional

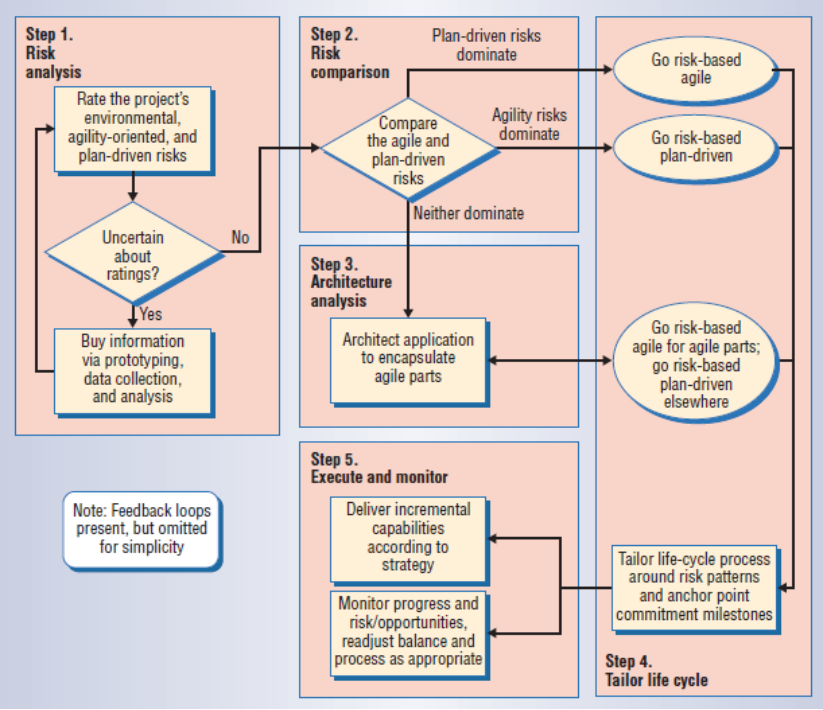
Informações da QPd		
		para facilitar as auditorias.
	Informações da QS3	
	Estratégias adotadas pelas organizações:	Ao contrário de outras tentativas de usar os métodos ágeis, dentro do *framework* do CMMI, com algumas soluções pontuais de práticas que podiam ser conciliadas, a Microsoft estava em busca de um método de ciclo de vida completo que fosse genuinamente ágil, mas que atendesse pelo menos aos requisitos do CMMI nível 3. Nesta busca, a empresa descobriu que a maior parte da literatura existente sobre a conciliação dos processos de desenvolvimento ágil e tradicional, defendia a sua compatibilidade, mas falhava em oferecer exemplos ou detalhes práticos. Assim, a Microsoft começou a pesquisar formas de relacionar o mundo do CMMI com a filosofia do desenvolvimento ágil. Como resultado desta busca, a Microsoft chegou na teoria do conhecimento profundo de Deming que oferece um mecanismo objetivo para medir e entender a variação natural do processo e utiliza a análise estatística para verificar se o processo está sob controle, ou seja, se o processo está sendo executado em conformidade com o previsto. Esta teoria se baseia na ideia de que a qualidade está relacionada com a conformidade ao processo e não com a conformidade a especificação. Assim, se o software estiver sendo construído da forma certa, então o software certo será construído e o cliente ficará satisfeito. A partir desta ideia, o autor defende que a compatibilidade do CMMI, em todos os seus níveis, com os métodos ágeis, pode ser alcançada ao usar métricas ágeis e projetar métodos de planejamento e controle baseados nesta análise estatística.
34.	**Título do documento:**	When software engineers met research scientists: A case study
	Autor(es):	J. Segal
	Data de publicação:	2005
	Fonte:	Empirical Software Engineering
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	O artigo descreve um estudo de caso onde engenheiros de software seguiram uma abordagem tradicional para construir uma biblioteca de componentes para um grupo de cientistas. Durante este estudo de caso, dois problemas foram revelados. O primeiro problema está relacionado aos requisitos: os cientistas tinham experiência em construir seus próprios sistemas no laboratório, de forma altamente iterativa, e com requisitos surgindo a cada iteração. Assim, eles tinham dificuldades em expressar os requisitos completos previamente, como demandava a metodologia. O segundo problema está relacionado à comunicação: usando apenas os documentos contratuais de especificação de requisitos, especificação de software e anotações de reuniões, os engenheiros de software não estavam conseguindo construir um conhecimento compartilhado do projeto com os cientistas. Assim, os autores discutem a importância de adaptar uma metodologia a um contexto particular. Em especial, os métodos ágeis se apresentaram como uma alternativa para resolver os problemas percebidos durante o estudo de caso. Porém, os métodos ágeis sozinhos também não atenderiam todas as necessidades do projeto. O projeto tem algumas facetas ideais para o desenvolvimento tradicional e outras facetas ideais para o desenvolvimento ágil de aplicações. Para combinar as abordagens tradicional e ágil, os autores se basearam na proposta de análise de riscos de Boehm.
35.	**Título do documento:**	Combining Agile methods with stage-gate project management

Informações da QPd	
Autor(es):	D. Karlström; P. Runeson
Data de publicação:	2005
Fonte:	IEEE Software
Processos de desenvolvimento de software:	Tradicional e Ágil
Práticas:	Não se aplica
Critérios de comparação:	Não se aplica
Informações da QS4	
Resultados obtidos pelas experiências das organizações	Os resultados dos estudos de caso em três grandes empresas de software mostram os benefícios que podem ser obtidos e os obstáculos que ainda precisam ser superados para integrar os métodos ágeis com os modelos de gestão de projetos *stage-gate*. Estes modelos de gestão de projetos *stage-gate* têm fases bem definidas e as fases são ligadas por marcos onde a gerência do projeto toma uma decisão de seguir ou não adiante no projeto. Duas das empresas estudadas – ABB Automation e Ericsson Microwave Systems – estavam tentando usar o XP em partes limitadas dos seus projetos de desenvolvimento de software que eram governados por modelos *stage-gate*. Em ambos os casos, as empresas mudaram as metodologias de desenvolvimento de software sem mudar o modelo de gestão do projeto. Os resultados obtidos pelas experiências destas três organizações podem ser agrupados em quatro áreas, onde são mostrados os resultados positivos (+) ou os pontos de atenção (!) provocados pelas práticas ágeis: **1) Planejamento e priorização** • **Funcionalidade mais importante primeiro:** a equipe de desenvolvimento está sempre trabalhando na funcionalidade mais importante e terminando ela primeiro. + *Feedback* mais cedo sobre as funcionalidades + Sem postergação das funcionalidades mais importantes • **Micro-planejamento:** o micro-planejamento permite que a equipe de desenvolvimento possa negociar a entrada de uma nova funcionalidade em relação a saída de alguma funcionalidade prevista anteriormente. + Evita o congestionamento de requisitos + Evita os planos fixos ! Pouco apoio ao planejamento de longo prazo **2) Comunicação e *follow-up:*** • **Equipes coesas:** + Boa comunicação interna ! Potencial de isolamento da equipe ágil dentro da organização • **Testes automatizados:** + Mecanismo para comunicação pessoal entre os membros em relação a mudança realizada que está provocando falhas nos testes automatizados + Alta qualidade • **Tarefas pequenas e gerenciáveis e Integração contínua:** + Menos confusão e sentimento de estar no controle do trabalho dividido em partes menores + Aumento da qualidade do produto + Medidas concretas do progresso do projeto de desenvolvimento **3) Modelo de processos e papéis** • **Envolvimento do cliente:** + *Feedback* contínuo + Funcionalidades relevantes + Gerente do produto é um bom candidato a representante do cliente • **Tarefas de documentação:** + Prioridades resolvidas entre documentação e código ! Conflitos visíveis entre diferentes quantidades de documentação – a requerida pelo modelo de gestão *stage-gate* e a realmente necessária para os engenheiros de software. **4) Gerência de projeto** • **Aumento do poder do nível da engenharia:** + Engenheiros se sentem motivados ! Gerentes se sentem temerosos inicialmente

Informações da QPd		
		! Necessidade de treinamento • **Foco:** + Engenheiros focam na *release* passada e atual e os gerentes focam na *release* atual e futura + ! Questões técnicas aparecem (muito) cedo para gestão • **Iniciativa do nível da engenharia:** + Pouca resistência a mudanças
36.	**Título do documento:**	Adapting PROFES for use in an agile process: An industry experience report
	Autor(es):	A. Jedlitschka; D. Hamann; T. Gohlert; A. Schroder
	Data de publicação:	2005
	Fonte:	International Conference on Product Focused Software Process Improvement (PROFES)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O objetivo deste trabalho é adaptar a metodologia de melhoria de processos PROFES (*Product Focused Improvement of Embedded Software Processes*) para que possa ser usada em um contexto industrial, com processos ágeis, para gerar produtos de qualidade previsível. O PROFES oferece um *framework* de métodos e ferramentas que apóia as indústrias na melhoria dos seus processos. Ao contrário das abordagens tradicionais de melhoria de processos, que se baseiam nos modelos de maturidade e na capacidade dos processos, o PROFES foca na qualidade do produto para identificar oportunidades de melhorias. Os autores sugerem como usar e evoluir a metodologia PROFES no ambiente ágil. Os processos ágeis precisam ser estendidos nos aspectos necessários para que estes processos possam se beneficiar do PROFES. Ao mesmo tempo, apesar desta adaptação, os processos e o ambiente devem continuar ágeis, o que leva também a uma necessidade de adaptação do próprio PROFES. Como o ambiente é ágil, os objetivos e os atributos de qualidade não são fixados no início, como prescreve o PROFES, mas sim durante o projeto. Além disso, como os processos ágeis trabalham com iterações muito curtas, fica difícil seguir todos os passos estabelecidos originalmente pelo PROFES. Na fase inicial do projeto, depois da elicitação dos requisitos, os objetivos iniciais do projeto já estão disponíveis. Com base nestes objetivos já é possível aplicar o PROFES. Nas iterações subseqüentes, a aplicação do PROFES é útil se aspectos importantes tiverem sido modificados.
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	Não existem muitas experiências em melhorias de processos de desenvolvimento de software ágeis. Basicamente dois cenários são possíveis: definir uma abordagem de melhoria de processos especificamente ágil; ou adaptar uma abordagem de melhoria de processos existente às necessidades do desenvolvimento ágil. Como a empresa BMW Car IT, queria uma solução pragmática com resultados rápidos, foi escolhido o segundo cenário. Em comparação com a IDEAL, a metodologia para a adaptação selecionada foi a PROFES que costuma ser utilizada para o domínio embarcado. A BMW Car IT escolheu um processo ágil, que combina práticas de XP, FDD e SCRUM, devido à necessidade de flexibilidade em relação aos requisitos dos clientes, entrega rápida de versões do produto e envolvimento do cliente. O processo resultante usa desenvolvimento iterativo, programação em pares seletiva, testes de unidade, refatoração e testes de aceitação. Neste estudo foram utilizados oito projetos de desenvolvimento de software e a implantação da abordagem se deu de forma *top-down* em todos os projetos ao mesmo tempo.
37.	**Título do documento:**	Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods

Informações da QPd		
	Autor(es):	B. Boehm; R. Turner
	Data de publicação:	2004
	Fonte:	International Conference on Software Engineering (ICSE)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	Os métodos ágeis prometem satisfação do cliente, baixa taxa de defeitos, velocidade no desenvolvimento e a solução para as mudanças de requisitos. As abordagens orientadas ao planejamento prometem previsibilidade estabilidade. Entretanto, as duas abordagens têm limitações e a compatibilidade entre elas ainda é pouco explorada. Em geral, os métodos ágeis e as abordagens de desenvolvimento orientadas ao planejamento são vistas como opositoras e a retórica dos especialistas de ambos os lados ainda continua essencialmente confrontacional. Entretanto, os autores propõem uma abordagem para balancear disciplina e agilidade. Primeiro, são caracterizados os cenários onde cada uma das abordagens tem mais chances de sucesso, de acordo com as suas forças e fraquezas em quatro áreas: aplicação, gestão, técnico e pessoal. A partir daí, são identificadas 5 dimensões (tamanho, criticidade, dinamismo, pessoal e cultura) para descrever o *spectrum* dos métodos ágeis e das abordagens orientadas a planejamento. Por último, é apresentado um método baseado em riscos, constituído de cinco passos, para desenvolver estratégias balanceadas, que tire vantagem das forças e minimize as fraquezas de ambas as abordagens de desenvolvimento de software, enquanto leva em consideração os objetivos, restrições e prioridades dos projetos de desenvolvimento de software das organizações.
38.	**Título do documento:**	Balancing the human and the engineering factors in software development
	Autor(es):	E. Mnkandla; B. Dwolatzky
	Data de publicação:	2004
	Fonte:	Conference in Africa (AFRICON)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O que torna difícil balancear qualidade, custo e prazo no desenvolvimento de software é a dependência do fator humano. Então, os autores propõem colocar mais peso no fator humano do desenvolvimento de software, assim como preconizado pelos métodos ágeis de desenvolvimento de software. Os autores utilizam como estudo de caso uma empresa de desenvolvimento da África do Sul. Os resultados do estudo de caso demonstram que, apesar dessa necessidade de ênfase no fator humano, é preciso que exista um equilíbrio entre estes fatores humanos e os fatores de engenharia para evitar possíveis problemas técnicos, tais como documentação incompleta. Para balancear os fatores humanos e de engenharia nos projetos de desenvolvimento de software, os autores afirmam que é preciso customizar uma metodologia, de acordo com a cultura e a filosofia da organização.
39.	**Título do documento:**	Using risk to balance agile and plan-driven methods

Informações da QPd	
Autor(es):	B. Boehm; R. Turner
Data de publicação:	2003
Fonte:	Computer
Processos de desenvolvimento de software:	Tradicional e Ágil
Práticas:	Não se aplica
Critérios de comparação:	**1) Aplicação** • **Objetivo principal:** no desenvolvimento ágil, os objetivos principais são entregar valor rapidamente e responder às mudanças. No desenvolvimento tradicional, os objetivos estão voltados à previsibilidade, estabilidade e qualidade. • **Tamanho:** os métodos ágeis costumam ser mais bem sucedidos com equipes e projetos pequenos, enquanto os métodos orientados a planejamento escalam melhor para equipes e projetos grandes. • **Ambiente:** os métodos ágeis se adaptam melhor a ambientes turbulentos e com muitas mudanças, enquanto os métodos orientados a planejamento atingem melhor os seus objetivos em ambientes estáveis. **2) Gestão** • **Relação com cliente:** os métodos ágeis prescrevem clientes dedicados ao projeto, trabalhando em conjunto com a equipe de desenvolvimento e focado nas funcionalidades priorizadas para uma determinada iteração. Já o desenvolvimento tradicional foca nas especificações do contrato e interage com o cliente apenas quando necessário. • **Planejamento e controle:** enquanto o desenvolvimento ágil trabalha com planos internos e controla o projeto de forma qualitativa, o desenvolvimento tradicional investe em planos documentados e no controle quantitativo do projeto. • **Comunicação:** nos métodos ágeis a comunicação é interpessoal entre os membros da equipe e favorece a troca de conhecimento tácito. Nos métodos orientados ao planejamento, a comunicação é baseada no conhecimento explicitamente documentado. **3) Técnico** • **Requisitos:** os métodos ágeis priorizam as estórias informais e os casos de testes, enquanto os métodos orientados ao planejamento formalizam e documentam os requisitos. • **Desenvolvimento:** o desenvolvimento ágil trabalha com um design simples, incrementos pequenos e refatoração. Os métodos orientados ao planejamento investem no projeto da arquitetura, incrementos mais longos e consideram a refatoração custosa. • **Testes:** os métodos ágeis adotam casos de testes automatizados para testar continuamente o sistema, enquanto os métodos orientados ao planejamento documentam o plano e os procedimentos de testes. **4) Pessoal** • **Clientes:** no desenvolvimento ágil os clientes devem ser dedicados ao projeto e trabalhar fisicamente próximos em conjunto com a equipe de desenvolvimento. Já no desenvolvimento tradicional, em geral, os clientes não são colocados junto com a equipe de desenvolvimento. • **Desenvolvedores:** o desenvolvimento ágil requer desenvolvedores mais talentosos para lidar com a necessidade de comunicação e interação frequentes do que o desenvolvimento tradicional. • **Cultura:** o desenvolvimento ágil requer uma cultura organizacional que valorize e dê liberdade aos desenvolvedores para a tomada de decisão. Já no desenvolvimento tradicional, predomina um ambiente controlado por políticas e procedimentos.
Informações da QS2	
Descrição da proposta de conciliação:	Apesar de cada abordagem (tradicional e ágil) ter um conjunto de características de projeto onde funciona muito bem, e muito melhor do que a outra, elas também têm limitações. Assim, fora do nicho de cada uma, uma abordagem combinada é preferível. As organizações não precisam de agilidade ou de qualidade, mas sim de ambas. Os métodos ágeis sozinhos ou o desenvolvimento tradicional sozinho não podem atender a esta necessidade. Assim, é necessária uma combinação de cada um deles. Para atender a esta necessidade, o autor

Informações da QPd		
		propõe uma abordagem baseada em gestão de riscos para balancear agilidade e disciplina. Os autores propõem uma abordagem baseada em riscos para incorporar nos projetos ambas as abordagens ágil e tradicional, nas proporções que o projeto necessita. Com base na exposição ao risco seria possível determinar o quanto de planejamento é suficiente e necessário introduzir em um projeto. Esta exposição ao risco, calculada para um determinado projeto, leva em consideração as características dos pontos fortes de cada uma das abordagens. Esta abordagem, por ser adaptável as características dos projetos, permite que a equipe de desenvolvimento se beneficie tanto dos métodos ágeis quanto dos métodos orientados ao planejamento, enquanto mitiga muitas das suas limitações. A abordagem proposta é composta de cinco passos (Figura 23) e utiliza a análise de riscos para adaptar os processos em uma estratégia de desenvolvimento: • **Passo 1:** aplicar análise de riscos em categorias de risco específicas associadas ao ambiente, aos métodos ágeis ou aos métodos orientados ao planejamento. • **Passo 2:** avaliar os resultados da análise de riscos para determinar se o projeto seria adequado para uma abordagem puramente ágil ou puramente tradicional. Neste caso, em que o projeto se encaixa perfeitamente no território onde uma das abordagens tem mais condições de sucesso, seguir para o passo 4. • **Passo 3:** este passo é atingido quando o projeto não se adapta a uma única abordagem. A equipe do projeto deve criar uma arquitetura que apóie o uso das práticas ágeis e utilizar o método orientado ao planejamento no restante do trabalho. • **Passo 4:** o foco deste passo é desenvolver uma estratégia que trate os riscos identificados. • **Passo 5:** como as decisões podem deixar de ser as mais adequadas depois de algum tempo, este passo existe para lembrar a gerência do projeto de monitorar e avaliar o desempenho dos processos selecionados, de acordo com o ambiente. Se o processo estiver com algum problema, a equipe de desenvolvimento poderá ajustar os níveis de agilidade e disciplina estabelecidos inicialmente.  Figura 23 – Resumo dos cinco passos do método de análise de risco
40.	**Título do documento:**	XP and the CMM
	Autor(es):	D.J. Reifer

Informações da QPd		
	Data de publicação:	2003
	Fonte:	IEEE Software
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	O CMM pode ser utilizado como um *framework* de melhoria de processos para acomodar as práticas do XP. A infraestrutura e a experiência com processos do CMM podem ajudar na institucionalização das práticas do XP nas organizações. Além disso, as práticas de gestão do CMM podem contribuir para sobrepor as dificuldades de escalabilidade do XP.
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	Especialistas da engenharia de software argumentam que o XP e o CMM são compatíveis e que as organizações que implementam práticas XP podem encaixá-las dentro do CMM, pois este representa um *framework* de melhoria de processos. Entretanto, utilizando como base os estudos realizados com projetos de duas organizações distintas durante seis meses, o autor verificou que é difícil implantar o XP dentro de organizações que já trabalham orientadas ao CMM e que faltam pesquisas e orientações neste sentido. Quando estas organizações, que tinham processos organizacionais avaliados e certificados de acordo com o CMM, iniciam a implementação do XP, o que se observa são conflitos entre as equipes que trabalham com processos distintos.
41.	**Título do documento:**	Certifying for CMM Level 2 and IS09001 with XP@Scrum
	Autor(es):	C. Vriens
	Data de publicação:	2003
	Fonte:	Agile Development Conference (ADC)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS1	
	Desafios para a conciliação:	• **Envolvimento da gerência sênior:** por um lado, a introdução do SCRUM diminui a necessidade de envolvimento gerencial, visto que as equipes se tornam mais auto-organizadas. Entretanto, para satisfazer os requisitos do CMM e da ISO, um maior envolvimento é necessário. • **Garantia de qualidade:** a ISO e o CMM exigem a existência de um grupo de garantia de qualidade, de preferência independente. A programação em pares é uma técnica efetiva para alcançar altos índices de qualidade, mas não dá visibilidade à gerência sobre os problemas e não-conformidades, como pedem os modelos.
	Informações da QS4	
	Resultados obtidos pelas experiências das organizações:	O artigo relata a experiência do departamento de Serviços de Engenharia de Software, da Philips, que conseguiu se certificar em CMM nível 2 e ISO 9001 utilizando uma metodologia de desenvolvimento ágil que combinava XP e SCRUM. A empresa utilizou o XP como ponto de partida, visto que ele já vinha sendo usado na organização por iniciativa dos próprios desenvolvedores, o que se diferencia da estratégia de implantação do CMM ou ISO que costuma ser *top-down*.

Informações da QPd		
		A experiência mostrou que, apesar de rejeitarem inicialmente a ideia da programação em pares, por acreditar que ela vai elevar os custos de desenvolvimento, os clientes acabam percebendo as suas vantagens em relação à diminuição de defeitos e a redução do ciclo de desenvolvimento. A empresa também percebeu que poderia estabelecer algumas regras para determinar quando a programação em pares é realmente necessária ou não. Em princípio, todo desenvolvimento é feito em pares, porém, no início de um novo desenvolvimento, é feita uma sessão de design e caso se considere que o código é simples ou rotineiro, então ele pode ser desenvolvido individualmente. Por outro lado, as correções são sempre feitas em pares, visto que a maioria dos erros são problemas de interface entre dois ou mais módulos. Tanto o cliente quanto os desenvolvedores gostaram de trabalhar de forma iterativa com pequenos incrementos. O cliente valoriza a entrega rápida do software e os desenvolvedores valorizam o aprendizado pela experiência. Ambos os grupos se sentem mais confiantes conforme as funcionalidades vão sendo disponibilizadas. Progressivamente, as reuniões diárias de *stand-up meeting* substituíram as reuniões normais de acompanhamento que demoravam muito tempo e mobilizavam muitos participantes. Depois de um ano de uso do XP, a empresa percebeu algumas limitações nesta abordagem e se deu conta de que estas limitações poderiam ser tratadas pelo SCRUM. Assim, surgiu a ideia de combinar o XP com o SCRUM, já que o XP foca nas práticas de engenharia de software e o SCRUM se dedica a gerência do projeto. Em paralelo com a introdução do XP e SCRUM, a empresa se certificou no CMM nível 2 e na ISO 9001:2000. Como o foco principal do CMM são grandes projetos e grandes organizações, a Philips começou escrevendo procedimentos para aplicar cada uma das áreas de processo do nível 2 do CMM, de acordo com o ambiente da organização. Estes documentos eram curtos, com no máximo duas páginas, e contêm guias sobre o que deve ser feito e como deve ser feito. Além disso, também foram criados alguns *templates* e *checklists* para uniformizar os artefatos gerados pelos projetos. Por fim, a aderência ao CMM e a ISO também exigiu a introdução de métricas como base para o controle dos projetos. Os resultados obtidos pela empresa levaram a algumas lições aprendidas: • **Contratar os melhores profissionais:** a contratação dos melhores engenheiros de software deve ser prioridade para a organização, pois nenhum processo, tradicional ou ágil, pode substituir o talento e as habilidades de pessoas competentes. • **Espaço para customizações:** diferentes projetos têm diferentes necessidades de processos devido à sua criticidade, domínio do problema, tecnologia usada, tamanho e distribuição geográfica da equipe. • **A equipe de qualidade deve servir de orientação e não de policiamento:** a melhor forma de uma equipe de qualidade atuar em um ambiente ágil é como orientadora da equipe de desenvolvimento em relação ao processo definido. As não-conformidades devem ser consideradas oportunidades de melhoria na forma de trabalho da equipe do projeto. • **Uso adequado do tempo:** engenheiros e clientes precisam aprender que uso adequado do tempo não é pressionar os engenheiros ao final de cada iteração para trabalhar além da hora. Ao contrário, os clientes é que devem ser forçados a tomar decisões, sobre o que vai ser implementado ou não em uma determinada iteração, para manter o andamento do projeto. • **Workshops de reflexão:** workshops de reflexão, ao final de cada iteração, ajudam a aumentar a maturidade e a efetividade da equipe de desenvolvimento. • **Orçamento:** apesar de costumarem recomendar que a organização faça uma reserva de 15% da sua capacidade de recursos, durante dois anos, para que consiga sair do nível 1 de maturidade do CMM para o nível 2, a organização ao reutilizar as definições de processos, *templates* e etc já existentes na própria organização ou disponíveis livremente, e ao trabalhar de uma forma ágil, conseguiu reduzir estes custos para 5%. • **Cliente presente:** a experiência da empresa mostrou que era mais fácil conseguir um espaço adequado perto do cliente do que convencê-lo a se mudar ou visitar frequentemente a equipe de desenvolvimento. Isso resultou em uma divisão da equipe e na perda de

Informações da QPd		
		compartilhamento de conhecimento técnico, mas facilitou a interação com o cliente. • **XP não é para qualquer um:** XP e SCRUM são metodologias altamente disciplinadas e que enfatizam a comunicação verbal. Nem todo engenheiro e cliente apreciam esta forma de trabalho e nem todos vão aderir a ela.
42.	**Título do documento:**	Requirements engineering and agile software development
	Autor(es):	F. Paetsch; A. Eberlein; F. Maurer
	Data de publicação:	2003
	Fonte:	International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Gerência de requisitos
	Critérios de comparação:	Geralmente, a engenharia de requisitos e os métodos ágeis são vistos como incompatíveis, já que a engenharia de requisitos é fortemente baseada em documentação para compartilhamento de conhecimento e os métodos ágeis focam a colaboração face-a-face entre os clientes e desenvolvedores. Este artigo compara a abordagem de engenharia de requisitos tradicional com o desenvolvimento de software ágil, analisando as similaridades e diferenças entre eles. Com base nesta análise, os autores tentam descobrir se algumas técnicas da engenharia de requisitos podem ser usadas no desenvolvimento ágil e se isso poderia resultar em melhorias nas abordagens ágeis. • **Envolvimento do cliente:** a engenharia de requisitos tradicional e os métodos ágeis concordam sobre a importância do envolvimento do cliente. Entretanto, os métodos ágeis assumem que o cliente tem uma representatividade ideal, enquanto a engenharia de requisitos emprega técnicas diferentes de elicitação de requisitos para obter o máximo de conhecimento possível dos *stakeholders* e resolver as inconsistências. Outra diferença é que na abordagem tradicional o cliente é envolvido principalmente durante o início do projeto, enquanto nos métodos ágeis o cliente participa ao longo de todo o processo de desenvolvimento de software. • **Entrevistas:** a técnica de elicitação de requisitos mais utilizada nos métodos ágeis é a entrevista. A entrevista é vista como a melhor forma de obter informações do cliente e de evitar mal-entendidos. Além disso, ela ajuda a estabelecer a confiança entre a equipe de desenvolvimento e o cliente. • **Priorização:** a priorização existe tanto na abordagem ágil quanto na engenharia de requisitos tradicional. Como os métodos ágeis procuram implementar primeiro as funcionalidades mais prioritárias, a priorização de requisitos é revista ao longo de todo o processo de desenvolvimento para lidar com os novos requisitos que são adicionados. • **Documentação:** no desenvolvimento de software ágil, criar um documento de requisitos completo e consistente é visto como não efetivo. Alguns métodos ágeis incluem algum tipo de documentação, mas não com a quantidade de detalhes recomendada pela engenharia de requisitos. Esta carência de documentação pode causar problemas a longo prazo como, por exemplo, na entrada de novos membros da equipe ou na manutenção futura do sistema. Por outro lado, no desenvolvimento tradicional a elaboração de uma documentação excessiva e as tentativas de mantê-la atualizada, prejudicam a produtividade do desenvolvimento de software. • **Validação:** os métodos ágeis utilizam as reuniões de revisão e os testes de aceitação como estratégia de validação dos requisitos. • **Gestão:** do ponto de vista da engenharia de requisitos, seria possível rastrear as mudanças nos requisitos. A rastreabilidade dos requisitos para os outros artefatos resulta em mais trabalho e em documentação adicional para que se possa saber por que as mudanças foram realizadas. Os métodos ágeis gerenciam os requisitos através de cartões de estórias informais ou listas de *backlog*. Os detalhes sobre os requisitos são levantados apenas no momento da sua implementação na respectiva iteração. • **Observação, análise social e *brainstorming*:** estas técnicas não são

Informações da QPd		
		mencionadas em nenhuma metodologia ágil, mas podem ser usadas com qualquer abordagem. • **Requisitos não funcionais:** os métodos ágeis não definem como lidar com os requisitos não funcionais. Resumindo, as principais fases da engenharia de requisitos (elicitação, análise e validação) estão presentes no desenvolvimento ágil, porém as técnicas usadas variam e essas fases não são tão claramente separadas quanto no desenvolvimento tradicional. Além disso, essas fases se repetem a cada iteração do processo de desenvolvimento de software.
43.	**Título do documento:**	Agile development: Good process or bad attitude?
	Autor(es):	R. Turner
	Data de publicação:	2002
	Fonte:	Product Focused Software Process Improvement
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS2	
	Descrição da proposta de conciliação:	Este artigo discute os relacionamentos entre os métodos ágeis e a melhoria de processos e avalia o suporte que o CMMI pode oferecer aos métodos ágeis. Olhando para os métodos ágeis no contexto do CMMI, os resultados de um workshop de métodos ágeis, com mais de 40 participantes, mostraram que apenas 17 dos 40 componentes foram considerados em conflito ou possivelmente em conflito. Outros 22 componentes foram considerados como apoio ou neutros aos métodos ágeis. Porém, o autor contesta estes resultados. O escopo do CMMI ampliou-se em relação ao CMM, então mapeia melhor para os métodos ágeis. Ainda assim, o CMMI ainda define o que fazer e não o como fazer. Assim, provavelmente, o melhor mapeamento dos métodos ágeis seria com um modelo como o *Personal Software Process* (PSP) ou o *Team Software Process* (TSP). As áreas de processos de gestão de processo são as mais problemáticas quando se considera os métodos ágeis. A filosofia ágil é orientada à equipe de desenvolvimento e não à organização que a apóia. Apesar de incluir alguns requisitos de infraestrutura para os projetos, os métodos ágeis abordam pouco do nível organizacional. Isso não significa que esta abordagem organizacional seja inapropriada, mas apenas que não foi especificamente tratada. Analisando cada uma das áreas de processo do CMMI, o autor conclui que os processos ágeis podem se adequar a melhoria de processos, mas para isso o CMMI precisa ser interpretado de uma forma mais liberal e menos literal.
44.	**Título do documento:**	Toward maturity model for extreme programming
	Autor(es):	J. Nawrocki; B. Walter; A. Wojciechowski
	Data de publicação:	2001
	Fonte:	Euromicro Conference
	Processos de desenvolvimento de software:	Tradicional e Ágil
	Práticas:	Não se aplica
	Critérios de comparação:	Não se aplica
	Informações da QS5	
	Modelo de processos com	Muitas pessoas não entendem corretamente o XP e o utilizam como uma desculpa para não adotar boas práticas de engenharia de software. Assim,

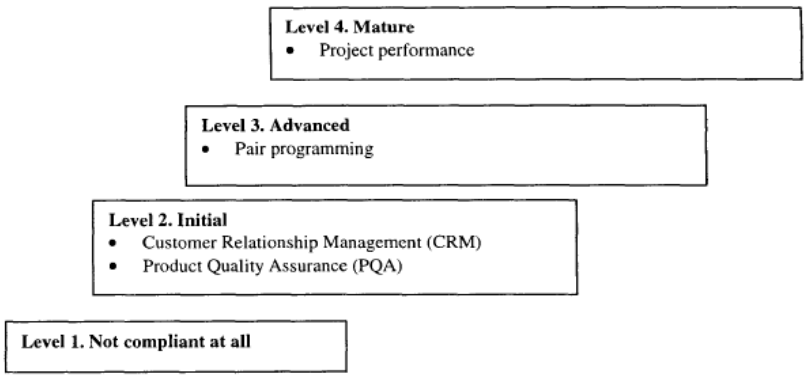
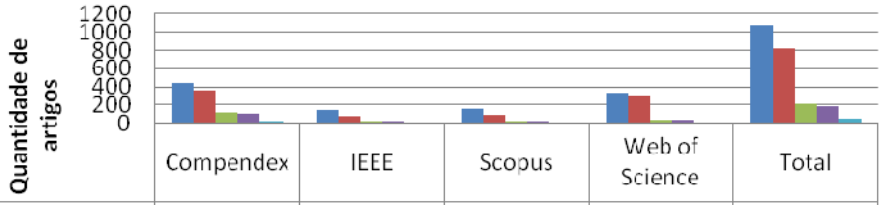
Informações da QPd		
	conciliação:	um modelo de maturidade é necessário para demonstrar o risco associado com um projeto, para deixar claro que o projeto não está seguindo nem o XP e nem o CMMI e para indicar um caminho de melhoria. O modelo de maturidade proposto para o XP (XPMM) é composto por quatro níveis de maturidade (Figura 24), onde cada um, com exceção do primeiro, apresenta um conjunto de práticas do XP que devem ser atingidas para que um determinado nível de maturidade possa ser alcançado:  Figura 24 – Modelo de maturidade para *Extreme Programming* (XPMM) Como o XP é uma metodologia leve, o seu modelo de maturidade também pretende ser e assim não oferece uma extensa documentação. Como muitas práticas XP, tais como simplicidade, são difíceis de serem avaliadas, esta avaliação deve ser feita através de conversas e observações. Apesar de subjetivas, estas técnicas vão permitir obter uma riqueza maior de informações que poderão auxiliar o avaliador em sua análise. No momento da publicação do artigo, o modelo XPMM proposto ainda estava em uma fase de testes experimentais.

Tabela 21 – Informações extraídas dos artigos da QPd

5.3.Considerações sobre os resultados

Nesta seção, os resultados obtidos com a revisão *quasi*-sistemática são discutidos. Inicialmente, a execução do protocolo nas máquinas de busca retornou um total de 1.078 artigos. Estes artigos foram submetidos a um processo de filtragem, composto por quatro passos (Figura 25): eliminação de duplicatas, filtro por título e resumo, filtro por disponibilidade de texto completo e filtro pela leitura completa do conteúdo do artigo. O número de artigos retornados de cada biblioteca digital, durante cada passo do processo de filtragem, também pode ser visto na Figura 25.



	Compendex	IEEE	Scopus	Web of Science	Total
Resultados das máquinas de busca	436	150	162	330	1078
Após exclusão das duplicatas	356	75	85	302	818
Após filtro por título e resumo	113	27	25	43	208
Após filtro por texto completo	99	27	20	36	182
Após filtro por conteúdo	22	8	4	10	44

Figura 25 – Processo de filtragem de artigos

Em relação às máquinas de busca, é possível observar, no gráfico da Figura 25, que aquelas com maior número de contribuições iniciais também se

mantiveram com o maior número de contribuições relevantes ao final da filtragem. Além disso, o % de exclusão de artigos é equivalente em todas as máquinas de busca. Assim, destacam-se as contribuições da Compendex.

Finalmente, 44 artigos foram efetivamente selecionados como resultado da execução da revisão *quasi*-sistemática. Este total de artigos selecionados demonstra que apenas 4% dos documentos obtidos inicialmente contribuíram efetivamente para a revisão *quasi*-sistemática. A grande maioria dos documentos lidos foi excluída, pois não trazia uma contribuição direta para caracterizar a conciliação de processos de desenvolvimento de software. Observando-se, nos apêndices, os critérios de exclusão aplicados, é possível perceber que as principais razões para exclusão de trabalhos foram: foco apenas nos processos de desenvolvimento de software de forma individual (CE4); falta de relevância para o tema da presente pesquisa (CE3); e foco em ferramentas ou ambientes computacionais e não no processo de desenvolvimento de software (CE5) (Figura 26).

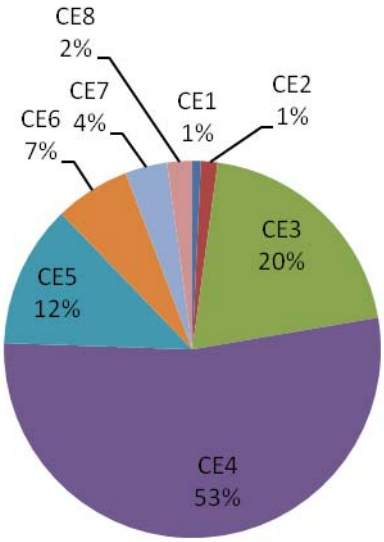


Figura 26 – Gráfico dos critérios de exclusão

A maneira como os artigos se referenciam, considerando os 44 artigos obtidos na execução da revisão *quasi*-sistemática, é mostrada pela Figura 27. Esta representação gráfica auxilia na identificação dos artigos mais influentes e de possíveis tendências de pesquisa. Este gráfico foi construído com base no exemplo de Kalinowski (2009).

Os artigos mais citados estão destacados e nota-se claramente a influência dos artigos de Boehm e Turner neste tema de pesquisa. Através da análise deste gráfico (Figura 27) também é possível perceber que, apesar de não ter sido imposta nenhuma restrição temporal na busca, os resultados obtidos datam a partir de 2001, o que demonstra ainda uma imaturidade desta área de pesquisa e a necessidade, também reforçada por alguns autores, de mais trabalhos científicos relacionados ao tema. Esta juventude da área se deve também a percepção inicial de que os processos de desenvolvimento tradicional, ágil e livre seriam opositores e que não haveria espaço para nenhuma tentativa de integração entre eles. Porém, é importante destacar o crescimento do número de trabalhos relacionados ao tema nos últimos anos, o que também pode ser observado na Figura 27.

Para uma análise completa dos resultados obtidos com esta revisão *quasi*-sistemática, é importante considerar as limitações desta revisão. Em primeiro lugar, a ausência de bibliotecas digitais, que indexem os trabalhos nacionais, impede que estudos realizados por pesquisadores brasileiros, possivelmente relevantes ao tema

de pesquisa, tenham sido incluídos, o que ameaça a completude desta revisão. Para minimizar este risco, estes trabalhos são acompanhados através de pesquisas em conferências e *workshops* nacionais. Alguns trabalhos nesta situação já foram identificados (ARIMOTO ET AL., 2009, SANTANA ET AL., 2006), mas eles também se encaixam na caracterização da conciliação de processos apresentada na próxima seção, como uma proposta do tipo combinação de práticas de modelos.

Além disso, outros trabalhos, pertinentes ao tema e publicados internacionalmente, não são indexados pelas máquinas de busca, devido à pouca relevância do veículo de publicação. Estes trabalhos (GLASS, 2001, GLAZER ET AL., 2008) foram identificados na revisão inicial da literatura e utilizados como base para a preparação do protocolo de busca, ainda que não possam ser utilizados como artigos de controle para calibrar as máquinas de buscas.

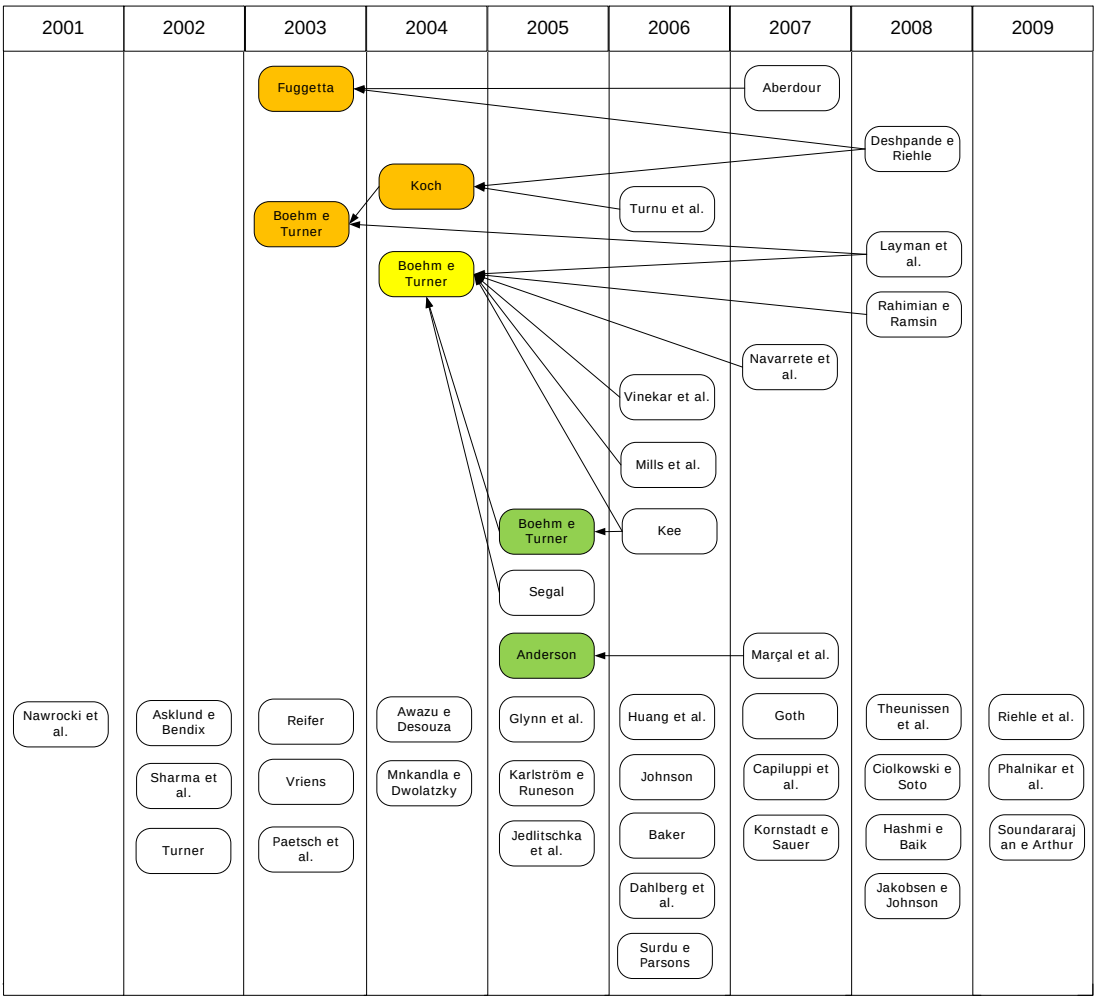


Figura 27 – Mapa de citações entre os artigos selecionados

6. Caracterização da conciliação de processos de desenvolvimento de software

O objetivo desta seção é caracterizar a conciliação de processos de desenvolvimento de software, de acordo com os resultados obtidos com a revisão *quasi*-sistemática. Como a questão de pesquisa foi subdividida em questão principal e questões secundárias, cada uma destas questões será respondida. Além disso, também serão analisadas as combinações possíveis entre dois ou dos três processos de desenvolvimento de software tradicional, livre e ágil.

QP: Como caracterizar a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?

Em geral, o que pôde ser percebido através da execução do estudo é a escassez de relatos na literatura sobre a conciliação dos três processos de desenvolvimento de software. Ao final da revisão *quasi*-sistemática, foi obtido um único artigo englobando os três processos. Este artigo trata dos desafios para a conciliação dos processos de desenvolvimento ágil e livre dentro de uma cultura corporativa (THEUNISSEN ET AL., 2008). O modelo de conciliação considerado no artigo em questão diz respeito a uma equipe ágil trabalhando dentro de uma organização e colaborando com um projeto de software livre externo. De fato, o enfoque do artigo não é exatamente nos três processos de desenvolvimento de software, visto que do ponto de vista tradicional não é abordado o processo de desenvolvimento, mas sim o ambiente e a cultura organizacional. Além disso, esta conciliação não acontece com os três modelos sendo usados dentro da mesma organização. Contudo, este trabalho foi o mais próximo disso que foi possível encontrar.

Tal resultado nos leva a seguinte indagação: por que será que não existem mais trabalhos relacionando os três processos de desenvolvimento de software? Após algumas reflexões, a primeira possibilidade considerada é que nem todos os autores e pesquisadores enxergam a existência de um processo de desenvolvimento no software livre, por considerar que os projetos de software livre são realizados de forma *ad-hoc*.

Entretanto, a alternativa que nos parece mais razoável para responder a esta questão é que a conciliação de apenas dois processos de desenvolvimento de software já é um grande desafio. Assim, os pesquisadores estariam iniciando os trabalhos nesta área de pesquisa, que ainda é bem recente se considerarmos que os primeiros trabalhos foram publicados em 2001, tentando combiná-los dois a dois, antes de envolver um terceiro elemento complicador nesta equação.

Esta hipótese também é embasada pela observação de que existe uma quantidade mais significativa de trabalhos combinando os processos tradicionais e ágeis, principalmente XP e SCRUM (ver Tabela 21). Começar a conciliação por estes dois processos parece ser uma opção natural se considerarmos que a distância entre eles é menor e que este poderia ser um caminho de evolução progressiva para se chegar na conciliação dos demais.

A conciliação entre o processo de desenvolvimento ágil e o livre também é incipiente ainda e, de forma geral, se baseia na conciliação de uma prática específica, como a integração de código ou o desenvolvimento orientado a testes (ver Tabela 19).

QS1 - Quais são as oportunidades e os desafios para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?

As principais oportunidades para a conciliação dos processos de desenvolvimento tradicional, ágil e livre são resumidas pela Tabela 22 e ordenados de acordo com o número de citações nos artigos. Estes resultados sinalizam uma busca pela disponibilidade do código, que é vista como uma oportunidade para a inovação no desenvolvimento de software. Além disso, as outras oportunidades citadas são argumentos comuns para pesquisas relacionadas à Engenharia de Software e aos processos de desenvolvimento de software em geral. Outra observação interessante é que apenas cinco artigos mencionam oportunidades de pesquisa nesta área, apesar da grande quantidade de trabalhos existentes. A análise dos textos lidos nos permitiu observar que tal fato se deve, predominantemente, a crença de que as motivações para este tema de pesquisa podem se basear apenas nos problemas comumente enfrentados pelo desenvolvimento de software tradicional.

#	Oportunidades	Descrição	Quantidade de artigos
1.	Disponibilidade do código	Disponibilidade e transparência no acesso ao código fonte.	02
2.	Colaboração	Facilitar a colaboração entre os participantes do processo.	01
3.	Qualidade	Melhoria na qualidade do código.	01
4.	Custo	Redução de custos nos projetos.	01

Tabela 22 – Principais oportunidades para a conciliação de processos de desenvolvimento de software

Os principais desafios para a conciliação dos três processos de desenvolvimento de software são resumidos pela Tabela 23, onde é possível observar uma diversidade de itens. Dentre os desafios citados, destaca-se o reconhecimento, pelos próprios pesquisadores e empresas que estão recorrendo à conciliação de processos, da carência de estudos científicos que possam contribuir com resultados experimentais e orientações para as organizações.

#	Desafios	Descrição	Quantidade de artigos
1.	Falta de estudos científicos sobre o tema	Carência de orientações para as organizações e ausência de resultados experimentais.	02
2.	Gestão de conhecimento	Conhecimento explícito *versus* conhecimento tácito. Conhecimento como fonte de poder *versus* conhecimento compartilhado.	02
3.	Resistência a mudança	Os clientes e desenvolvedores podem não aceitar a mudança do paradigma desenvolvimento.	02
4.	Estrutura organizacional	Controle hierárquico *versus* flexibilidade.	02
5.	Sistema de reconhecimento e recompensa	Avaliação individual *versus* avaliação do grupo.	02
6.	Comunicação	Comunicação face-a-face *versus* através da Internet.	01
7.	Quantidade de artefatos	Elaboração de uma grande quantidade de artefatos *versus* simplicidade e parcimônia.	01
8.	Tomada de decisão	Centralizada *versus* descentralizada.	01
9.	Cultura organizacional	Barreiras culturais.	01
10.	Planejamento do trabalho	Planejamento formal e documentado *versus* planejamento informal.	01
11.	Acompanhamento do trabalho	Monitoramento das atividades desempenhadas pelos desenvolvedores *versus* trabalho voluntário.	01
12.	Garantia da qualidade	Revisão por pares *versus* técnicas formais que dão visibilidade das não-conformidades à gerência sênior.	01
13.	Relacionamento com cliente	Necessidade de tomar ações para preparar o cliente para o novo processo de desenvolvimento.	01
14.	Tecnologia	Adequação das tecnologias e ferramentas existentes a implantação da metodologia.	01
15.	Sistemas legados	Os sistemas legados não são fáceis de refatorar para que possam ser desenvolvidos em incrementos.	01
16.	Requisitos	Completos e documentados *versus* funcionais e informais.	01
17.	Manter a certificação dos processos	As organizações se preocupam em melhorar os seus processos de desenvolvimento de software e, ao mesmo tempo, conseguir manter a certificação dos seus processos.	01

Tabela 23 – Principais desafios para a conciliação de processos de desenvolvimento de software

QS2 - Quais são as propostas, de outros pesquisadores ou especialistas, para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?

As propostas para conciliação de processos de desenvolvimento de software foram agrupadas em categorias e são apresentadas pela Tabela 26. O tipo de proposta mais comum é aquela que sugere a adaptação de modelos tradicionais para introduzir práticas de desenvolvimento ágil. Outro tipo, bastante recorrente, de proposta apresentada na literatura é a combinação de práticas de diferentes modelos.

O problema deste tipo de proposta é que a natureza complexa do desenvolvimento de software e a grande variedade de métodos existentes tornam a tarefa de comparação dos modelos, uma tarefa árdua e imprecisa (BOEHM AND TURNER, 2003). Além disso, esta rigidez na integração entre os modelos de desenvolvimento limita o potencial de sinergia entre eles, pois constrói uma metodologia híbrida e incompleta (SANTANA ET AL., 2009), onde não se consegue garantir que o processo resultante realmente tenha as características desejadas. Assim, destacam-se as propostas da Microsoft de uso de controle estatístico (ANDERSON, 2005) e da análise de riscos (BOEHM AND TURNER, 2003) que são as que mais se aproximam dos objetivos que este trabalho de pesquisa visa alcançar.

#	Propostas	Descrição	Quantidade de artigos
1.	Adaptação de modelos tradicionais	Este tipo de proposta envolve a adaptação de modelos tradicionais para incorporar características do desenvolvimento ágil de forma que possam ser aplicados a contextos específicos como reutilização de software de prateleira, desenvolvimento distribuído e desenvolvimento de aplicações móveis.	06
2.	Combinação de práticas de modelos de desenvolvimento	Este tipo de proposta envolve o mapeamento da equivalência das práticas dos modelos de desenvolvimento distintos, tais como XP com CMMI ou SCRUM com ISO, e geralmente resulta em uma análise do quanto os métodos ágeis estão ou não aderentes aos modelos de qualidade tradicionais e como os métodos ágeis podem ser modificados para incorporar as práticas que faltam.	02
3.	Análise de riscos	Proposta de Boehm e Turner para utilizar a análise de riscos para balancear a agilidade e a disciplina nos projetos de desenvolvimento de software, levando-se em consideração as características e necessidades do projeto. No mapa de citações dos artigos (Figura 27), esta proposta se mostrou a mais referenciada pelos demais pesquisadores.	02
4.	Práticas específicas	Comparação e transferência de práticas específicas, como gerência de configuração e gestão de conhecimento, de um processo de desenvolvimento de software para outro.	02
5.	Controle estatístico	Proposta que envolve o uso de mecanismos objetivos para medir e entender a variação natural do processo e utiliza a análise estatística para verificar se o processo está sob controle, ou seja, se o processo está sendo executado de acordo com o previsto.	02
6.	Ensino	Uma das propostas envolve o ensino de como compor uma metodologia de desenvolvimento apropriada ao projeto e à equipe de desenvolvimento, enfatizando a colaboração e o aprendizado. O aluno é levado a perceber que um processo de	01

#	Propostas	Descrição	Quantidade de artigos
		desenvolvimento de software consiste em um determinado agrupamento de práticas.	
7.	Ambivalência	Esta proposta prevê uma estrutura organizacional para conciliar as culturas organizacionais necessárias ao desenvolvimento de software tradicional e o ágil, através da separação dos modelos de desenvolvimento em unidades organizacionais distintas subordinadas a uma gerência maior comum. Esta proposta separa as equipes de desenvolvimento de processos de desenvolvimento distintos, o que dificulta a colaboração e a comunicação, sem lidar com a conciliação dos processos de desenvolvimento já que pressupõe que um projeto será desenvolvido de forma totalmente ágil ou puramente tradicional.	01
8.	Construção de uma comunidade híbrida de desenvolvimento de software	Esta proposta incentiva que a construção de uma comunidade híbrida dentro das organizações comece pela criação de uma comunidade de práticas. Para fomentar esta comunidade devem ser estabelecidos também mecanismos de governança e oferecida uma infraestrutura de apoio.	01
9.	*Spectrum*	O balanceamento apropriado entre as práticas do desenvolvimento tradicional e do ágil pode ser alcançado através de um *spectrum* que indica a relação entre os projetos tradicionais e ágeis, baseada no tamanho da equipe e nos custos de uma possível falha do sistema.	01
10.	*Framework* de melhoria de processos	Esta proposta defende que o CMM pode ser utilizado como um *framework* de melhoria de processos para acomodar as práticas do XP. A infraestrutura e a cultura de processos do CMM podem ajudar na institucionalização do XP nas organizações. Além disso, as práticas de gestão do CMM também podem contribuir para sobrepor as dificuldades de escalabilidade do XP.	01
11.	Foco no aspecto humano	Esta proposta preconiza mais ênfase no fator humano do desenvolvimento de software, mas de forma que exista um equilíbrio entre estes fatores humanos e os fatores técnicos. Para balancear os fatores humanos e de engenharia nos projetos de desenvolvimento de software, é preciso customizar uma metodologia, de acordo com a cultura e a filosofia da organização.	01

Tabela 24 – Propostas para a conciliação de processos de desenvolvimento de software

QS3 - Quais são as estratégias adotadas, pelas organizações ou pelas comunidades, para a conciliação de processos de desenvolvimento de software tradicional, livre e ágil?

QS4 - Quais foram os resultados obtidos pelas organizações ou comunidades que conciliaram processos de desenvolvimento de software tradicional, livre e ágil?

As respostas a estas duas questões secundárias foram consolidadas e são apresentadas pela Tabela 25, onde é possível observar uma quantidade significativa de grandes empresas interessadas no assunto, ainda que a grande maioria das estratégias adotadas seja de combinação de modelos.

#	Organizações ou Comunidades	Estratégias	Descrição	Artigos
1.	ABB Automation e Ericsson	Adaptação de modelos tradicionais	Adaptação do modelo *stage-gate* para incorporar práticas ágeis do XP, resultando em	[35]

#	Organizações ou Comunidades	Estratégias	Descrição	Artigos
			um conjunto de lições aprendidas.	
2.	BMW	Adaptação de modelos tradicionais	Adaptação de um modelo de melhoria de processos tradicionais (PROFES) para o desenvolvimento ágil, combinando XP, FDD e SCRUM, e resultando em um conjunto de lições aprendidas.	[36]
3.	DTE Energy	Combinação de modelos	Uso da própria metodologia ágil na iniciativa de melhoria de processos que tinha como objetivo combinar o CMMI nível 3 com os métodos ágeis.	[28]
4.	Exército americano	Combinação de modelos	Combinação dos modelos CMMI nível 5 com XP, através da seleção das práticas mais adequadas para implementação	[30]
5.	Microsoft	Controle estatístico	Uso do controle estatístico para integrar o CMMI nível 5 com o desenvolvimento ágil.	[33]
6.	Motorola	Combinação de modelos	Combinação dos modelos CMMI nível 5 com métodos ágeis, obtendo redução de defeitos e melhoria na produtividade.	[27]
7.	Philips	Combinação de modelos	Combinação dos modelos CMM nível 2 e ISO 9001 com XP e SCRUM.	[41]
8.	Projeto PyPy	Combinação de práticas	Projeto de software livre que adota prática ágil de desenvolvimento orientado a *sprint*.	[3]
9.	SAP	Implantação de tecnologia	Facilitar a adoção de software livre nas organizações que adotam processos tradicionais através do uso de software *forge* que integra diversas ferramentas colaborativas de desenvolvimento de software.	[06]
10.	Systematic	Combinação de modelos	Combinação dos modelos CMMI e SCRUM.	[21]
11.	Universidade de Memphis	Aplicação da proposta de análise de riscos	Relato de experiência da aplicação da proposta de análise de riscos que resultou na escolha do XP para o projeto de desenvolvimento em questão.	[31]
12.	Volvo	Combinação de modelos	Combinação dos modelos ISO 15504 e XP.	[29]

Tabela 25 – Experiências de conciliação de processos de desenvolvimento de software das organizações

QS5 - Quais são os modelos de processos de desenvolvimento de software híbridos existentes que combinem as abordagens tradicional, livre e ágil?

Os modelos híbridos existentes, seja para processos de desenvolvimento de software ou para avaliação de processos ou produtos, são resumidos e ordenados, por número de ocorrências nos artigos, pela Tabela 26:

#	Modelo Híbrido	Descrição	Quantidade de artigos
1.	Cap Gemini`s Open Source Maturity Model (OSMM)	Modelo comercial que avalia o software em relação a um conjunto de indicadores do produto.	02
2.	Navica`s Open Source Maturity Model (OSMM)	Processo formal, publicado sob uma licença livre, que avalia a maturidade do software, suporte, documentação, treinamento e serviços.	02
3.	Business Readiness Rating (OpenBRR)	Modelo de avaliação que está em desenvolvimento por um conjunto de organizações que procura expandir os dois modelos anteriores para criar um padrão aberto.	02
4.	Qualification and Selection of Open Source (QSOS)	QSOS é um método concebido para qualificar, selecionar e comparar software livre de forma objetiva e rastreável.	01
5.	Extreme Programming Maturity Model (XPMM)	O objetivo deste modelo é demonstrar o risco associado com um projeto que não está seguindo as práticas do XP como deveria e indicar um caminho de melhoria. O XPMM é composto por quatro níveis de maturidade, onde cada um, com exceção do primeiro, apresenta um conjunto de práticas do XP que devem ser atingidas para que um determinado nível de maturidade possa ser	01

#	Modelo Híbrido	Descrição	Quantidade de artigos
		alcançado.	

Tabela 26 – Caracterização dos modelos híbridos

7.Conclusão

As organizações se engajam em uma grande variedade de projetos de desenvolvimento de software, mas estes projetos possuem características distintas, onde os modelos de desenvolvimento tradicional, ágil e livre, normalmente percebidos como opositores, se complementam, pois cada um funciona melhor ou enfrenta dificuldades em determinado aspecto. Assim, a conciliação de processos de desenvolvimento de software tenta entender as similaridades e distinções entre estes modelos, identificar sob que condições cada um funciona melhor e compreender como o melhor de cada um pode ser combinado, trazendo ganhos de produtividade e qualidade para as organizações.

Como a maioria dos projetos de desenvolvimento de software possui características diversificadas, nenhuma metodologia sozinha consegue oferecer todas as soluções necessárias. Assim, abordagens que equilibrem os diferentes processos de desenvolvimento, são viáveis e necessárias (BOEHM, 2002). Esta busca pelo equilíbrio pode ser observada pelo crescimento da adaptação dos processos de desenvolvimento de software por parte das organizações (HANSSON ET AL., 2006), enquanto decresce a quantidade de organizações que seguem um modelo de qualidade de modo totalmente prescritivo (PATEL ET AL., 2006).

Nada impede que uma organização tenha mais de um processo de desenvolvimento, de acordo com o tamanho e criticidade do projeto (ORR, 2002). Uma organização capaz de combinar os diferentes processos de desenvolvimento de software vai poder se beneficiar do que existe de melhor em todos eles (VINEKAR ET AL., 2006).

Neste contexto, neste trabalho de pesquisa estamos interessados em estudar a viabilidade da reconciliação de processos de desenvolvimento de software e identificar quais abordagens têm sido propostas por outros pesquisadores ou adotadas por organizações que tenham implementado práticas de processos de software distintos.

Com o objetivo de responder a estas questões de pesquisa, este trabalho apresentou uma *quasi*-revisão sistemática que caracterizou a conciliação de processos de desenvolvimento de software. Inicialmente, esta revisão identificou um total de 1078 artigos. Desse total, 44 artigos foram selecionados como parte desta revisão.

As informações extraídas desses artigos contribuíram para as seguintes observações: tratar poucos estudos com a conciliação da vida tradicional, ágil e livre / modelos de desenvolvimento aberto, a conciliação da vida tradicional, com o desenvolvimento ágil é mais comum, já que a diferença entre estes dois modelos não é tão grande, as grandes organizações são muito interessantes na reconciliação e eles estão tentando combinar estes modelos, sem qualquer orientação sobre como fazer isso.

Acreditamos que esta caracterização será importante para os profissionais que desejam manter-se atualizados com o estado da prática. Esta revisão também irá ajudar a comunidade científica que trabalha com desenvolvimento de software a construir um entendimento comum dos desafios que devem ser enfrentados, bem como para identificar os temas que têm sido pesquisados e onde a investigação ainda está faltando. Finalmente, os resultados de tal investigação serão relevantes para a indústria de software que está clamando por soluções nesta área.

Apesar das propostas já existentes, a customização de processos de desenvolvimento de software, obtendo uma mescla equilibrada do desenvolvimento tradicional, livre e ágil, ainda é uma questão em aberto, pois a maioria das propostas ainda está em um estágio inicial de pesquisa, onde se dedica a uma combinação rígida de práticas ou modelos, onde o resultado é híbrido, mas não necessariamente atende às necessidades dos projetos e organizações.

Neste trabalho de pesquisa defende-se que a conciliação significa mais do que a combinação de práticas dos diferentes modelos de processos. A hipótese investigada é que é possível equilibrar os aspectos de colaboração e disciplina, presentes, com diferentes ênfases, em cada um dos modelos de desenvolvimento, pois estes são os fatores determinantes à adaptação de um processo de desenvolvimento de software aos contextos específicos dos projetos e organizações.

Nossa agenda de pesquisa neste tópico envolve o desenvolvimento de uma infraestrutura para adaptar os processos de desenvolvimento software, balanceamento de colaboração e de disciplina, de acordo com o contexto das organizações e dos projetos.

Como trabalho futuro, pretende-se também re-executar, em 2010, este protocolo tentando captar referências mais recentes que estendem o espaço de busca em ordem cronológica. Além disso, também pode-se incluir outras bibliotecas digitais, em uma tentativa de recuperar documentos indexados somente por estas máquinas, que poderiam alargar o espaço de busca geograficamente. Embora a abordagem sistemática seguida neste estudo garanta a confiabilidade e integridade dos resultados, isto poderá ser amplificado durante uma re-execução.

Por fim, também é importante investigar se esses resultados são consistentes com o que se observa na prática, ou seja, na rotina das organizações. Para atingir este objetivo, é preciso planejar e conduzir um estudo experimental.

Agradecimentos

Este trabalho é parcialmente financiado pelo CNPq sob o processo no. 142006/2008-4. Os autores também gostariam de agradecer aos alunos de doutorado e de mestrado que ajudaram na validação dos resultados desta revisão *quasi*-sistemática.

Referências Bibliográficas

ABRANTES, J. F.; TRAVASSOS, G. H., 2007, *Revisão quasi-Sistemática da Literatura: Caracterização de Métodos Ágeis de Desenvolvimento de Software*, Relatório Técnico ES-714/07, PESC-COPPE. Disponível em: http://www.cos.ufrj.br.

ANDERSON, D., 2005, "Stretching agile to fit CMMI level 3 - the story of creating MSF for CMMI® process improvement at Microsoft corporation". *Agile Conference*, pp. 193-201, Denver, USA.

ARIMOTO, M. M.; MURAKAMI, E.; CAMARGO, V. V.; ET AL., 2009, "Adherence Analysis of Agile Methods According to MR-MPS Reference Model". *Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 249-263, Ouro Preto, Minas Gerais, Brasil.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; ET AL., 2001. Manifesto for Agile Software Development. Disponível em: http://agilemanifesto.org/. Acesso em: 15 Dec 2008.

BIOLCHINI, J.; MIAN, P. G.; NATALI, A. C. C.; ET AL., 2005, *Systematic Review in Software Engineering*, Relatório Técnico ES-679, PESC-UFRJ.

BOEHM, B., 2002, "Get ready for agile methods, with care", *IEEE Computer*, v. 35, n. 1, pp. 64-69.

BOEHM, B.; TURNER, R., 2003, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA, Addison-Wesley Professional.

BRERETON, P.; KITCHENHAM, B. A.; BUDGEN, D.; ET AL., 2007, "Lessons from applying the systematic literature review process within the software engineering domain", *J. Syst. Softw.*, v. 80, n. 4, pp. 571-583.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S., 2006, *CMMI: Guidelines for Process Integration and Product Improvement*. 2 ed. Boston, MA, Addison-Wesley Professional.

COCKBURN, A., 2001, *Agile Software Development*. Boston, Massachussetts, Addison-Wesley Professional.

CONBOY, K.; FITZGERALD, B., 2004, "Toward a conceptual framework of agile methods: a study of agility in different disciplines". In: *Proceedings of the Workshop on Interdisciplinary software engineering research*, pp. 37-44, Newport Beach, CA, USA.

CUBRANIC, D.; BOOTH, K. S., 1999, "Coordinating open-source software development". In: *Proceedings of the 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 61-66, Stanford, USA.

FELLER, J.; FITZGERALD, B., 2001, *Understanding Open Source Software Development*. Addison-Wesley Professional.

GLASS, R. L., 2001, "Agile Versus Traditional: Make Love, Not War!", *Cutter IT Journal*, v. 14, n. 12, pp. 12-18.

GLAZER, H.; DALTON, J.; ANDERSON, D.; ET AL., 2008, *CMMI or Agile: Why Not Embrace Both!*, SEI-CMU. Disponível em: http://www.sei.cmu.edu/publications/documents/08.reports/08tn003.html.

HAEFLIGER, S.; VON KROGH, G.; SPAETH, S., 2007, "Code Reuse in Open Source Software", *Management Science*, v. 54, n. 1, pp. 180-193.

HANSSON, C.; DITTRICH, Y.; GUSTAFSSON, B.; ET AL., 2006, "How agile are industrial software development practices?", *Journal of Systems and Software*, v. 79, n. 9, pp. 1295-1311.

HIGHSMITH, J.; COCKBURN, A., 2001, "Agile Software Development: The Business of Innovation", *IEEE Computer*, v. 34, n. 9, pp. 120-127.

ISO/IEC, 2007, *ISO/IEC 12207:1995, Information technology - Software life cycle processes*.

KALINOWSKI, M., 2009, *DBPI: Abordagem de Prevenção de Defeitos de Software para Apoiar Melhoria de Processos e Aprendizado Organizacional*. Exame de Qualificação, Universidade Federal do Rio de Janeiro (UFRJ)

KITCHENHAM, B., 2004, *Procedures for Performing Systematic Reviews*, Technical Report TR/SE-0401, Technical report Software Engineering Group, Department of Computer Science, Keele University. Disponível em: http://www.idi.ntnu.no/emner/empse/papers/kitchenham_2004.pdf.

ORR, K., 2002, "CMM Versus Agile Development: Religious Wars and Software Development", *Cutter Consortium*, v. 3, n. 7, pp. 1-34.

PAI, M.; MCCULLOCH, M.; GORMAN, J. D.; ET AL., 2004, "Systematic reviews and meta-analyses: an illustrated, step-by-step guide", *The National Medical Journal of India*, v. 17, n. 2, pp. 86-95.

PATEL, C.; LYCETT, M.; MACREDIE, R.; ET AL., 2006, "Perceptions of Agility and Collaboration in Software Development Practice". In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS)*, pp. 1-7, Kauai, Hawaii, USA.

PAULK, M., 2001, "Extreme programming from a CMM perspective", *Software, IEEE*, v. 18, n. 6, pp. 19-26.

PRESSMAN, R. S., 2001, *Software Engineering: A Practitioner's Approach*. 5 ed. McGraw-Hill.

RAYMOND, E. S., 2001, *The Cathedral & the Bazaar*. O'Reilly Media Inc.

REIS, C. R., 2003, *Caracterização de um Processo de Software para Projetos de Software Livre*. Dissertação de Mestrado, USP, São Paulo, SP, Brasil. Disponível em: http://www.async.com.br/~kiko/dissert.pdf.

SANTANA, C.; GUSMÃO, C.; SOARES, L.; ET AL., 2009, "Agile Software Development and CMMI: What We Do Not Know about Dancing with Elephants", *Agile Processes in Software Engineering and Extreme Programming*, , pp. 124-129.

SANTANA, C. A.; TIMÓTEO, A. L.; VASCONCELOS, A. M. L., 2006, "Mapeamento do modelo de Melhoria do Processo de Software Brasileiro (MPS.Br) para empresas que utilizam Extreme Programming (XP) como metodologia de desenvolvimento". In: *Anais do V Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 130-143, Vila Velha, Espírito Santo, Brasil.

SOFTEX, 2009, *Melhoria de Processo do Software Brasileiro – Guia Geral*, Modelo de Qualidade Versão 2.0 Disponível em: http://www.softex.br.

SOUZA, G. D. S., 2008, *Ambientes De Engenharia De Software Orientados a Corporação*. Doutorado, UFRJ Disponível em: http://teses2.ufrj.br/Teses/COPPE_D/GleisonDosSantosSouza.pdf.

THEUNISSEN, M.; KOURIE, D.; BOAKE, A., 2008, "Corporate-, Agile- and Open Source Software Development: A Witch's Brew or An Elixir of Life?", *Balancing Agility and Formalism in Software Engineering: Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques (CEE-SET)*, Springer-Verlag, pp. 84-95.

TRAVASSOS, G.; DOS SANTOS, P.; NETO, P.; ET AL., 2008, "An Environment to Support Large Scale Experimentation in Software Engineering". In: *Proceedings of 13th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pp. 193-202, Belfast, United Kingdom.

VINEKAR, V.; SLINKMAN, C. W.; NERUR, S., 2006, "Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View", *Information Systems Management*, v. 23, n. 3, pp. 31-42.

WARSTA, J.; ABRAHAMSSON, P., 2003, "Is Open Source Software Development Essentially an Agile Method?". In: *Proceedings of the Workshop on Open Source Software Development*, pp. 143-147, Portland.

Apêndice A – Dados das Publicações do Segundo Filtro da QPa

A Tabela 27 contém a listagem dos 8 artigos da QPa analisados no segundo filtro. Para cada publicação são detalhadas as seguintes informações: título, base de dados de onde essa publicação foi selecionada, disponibilidade do arquivo com o texto completo do artigo, resultado da análise do segundo filtro e o critério de inclusão ou exclusão aplicado. O artigo selecionado foi realçado.

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
1.	Agile, open source, distributed, and on-time - Inside the Eclipse development process	Compendex	não			
2.	An agile approach for integration of an open source health information system	Compendex	sim	não	CE5	
3.	An open source approach to developing software in a small organization	Compendex	sim	não	CE4	
4.	Convergence ignites growth of open-source collaboration	Scopus	sim	não	CE4	
5.	Corporate-, Agile- and Open Source Software development: A witch's brew or an elixir of life?	Compendex	obtido com os autores	sim		CI1, CI2, CI3
6.	Open source development and Agile methods	Compendex	não			
7.	Open Source Software Development: Structural Tension in the American Experiment	Scopus	não			
8.	Software production infrastructure to support agile methodologies	Compendex	sim	não	CE5	

Tabela 27 – Dados das publicações da Questão Principal A

Apêndice B – Dados das Publicações do Segundo Filtro da QPb

A Tabela 28 contém a listagem dos 20 artigos da QPb analisados no segundo filtro. Para cada publicação são detalhadas as seguintes informações: título, base de dados de onde essa publicação foi selecionada, disponibilidade do arquivo com o texto completo do artigo, resultado da análise do segundo filtro e o critério de inclusão ou exclusão aplicado. Os artigos selecionados foram realçados.

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
1.	Agile principles and open source software development: A theoretical and empirical discussion	WebOfScience	sim	sim		CI1
2.	Agile, open source, distributed, and on-time - Inside the Eclipse development process	Scopus	não			
3.	An agile approach for integration of an open source health information system	Scopus	sim	não	CE5	
4.	An open source approach to developing software in a small organization	Scopus	sim	não	CE4	
5.	An open source domain-specific tools framework to support model driven development of OSS	Compendex	sim	não	CE5	
6.	Applying Open Source development practices inside a company	Scopus	obtido com autores	não	CE5	
7.	Ascending into order: A reflective analysis from a small open source development team	Scopus	sim	não	CE4	
8.	Continuous integration in open source software development	Scopus	sim	sim		CI1
9.	Convergence ignites growth of open-source collaboration	Scopus	sim	não	CE4	
10	Corporate-, Agile- and Open Source Software development: A witch's brew or an elixir of life?	Scopus	obtido com autores	não	CE8	
11	Key transformational techniques to achieve enterprise-scale interoperability	Scopus	sim	não	CE5	
12	Making the leap to open source	Scopus	sim	não	CE4	
13	Modeling and simulation of open source development using an agile practice	Scopus	sim	sim		CI1
14	Open source development and Agile methods	Scopus	não			
15	Open Source Drives Innovation	IEEE	sim	não	CE4	
16	Open Source Software Development: Structural Tension in the American Experiment	Scopus	não			
17	Software production infrastructure to support agile methodologies	Scopus	sim	não	CE5	
18	Sprint driven development: Agile methodologies in a distributed open source project (PyPy)	Scopus	sim	não	CE7	
19	Sprinting toward open source development	Scopus	sim	sim		CI1, CI2, CI4
20	Traveling the open road: Using open source practices to transform our organization	Compendex	sim	não	CE4	

Tabela 28 – Dados das publicações da Questão Principal B

Apêndice C – Dados das Publicações do Segundo Filtro da QPc

A Tabela 29 contém a listagem dos 86 artigos da QPc analisados no segundo filtro. Para cada publicação são detalhadas as seguintes informações: título, base de dados de onde essa publicação foi selecionada, disponibilidade do arquivo com o texto completo do artigo, resultado da análise do segundo filtro e o critério de inclusão ou exclusão aplicado. Os artigos selecionados foram realçados.

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
1.	A case study of open source software development: the Apache server	IEEE	sim	não	CE4	
2.	A development process for building OSS-based applications	WebOfScience	sim	sim		CI1, CI2, CI3, CI4
3.	A framework for creating, hybrid-open source software communities	WebOfScience	obtido com autores	sim		CI1, CI2, CI3
4.	A history of IBM's open-source involvement and strategy	WebOfScience	não			
5.	A methodological framework for socio-cognitive analyses of collaborative design of open source software	Compendex	sim	não	CE3	
6.	A study of configuration management in open source software projects	Compendex	sim	sim		CI1, CI2, CI3
7.	A study of open source software evolution data using qualitative simulation	Compendex	obtido com autores	não	CE3	
8.	Achieving quality in open-source software	Compendex	sim	sim		CI1
9.	Adapting the "staged model for software evolution" to free/libre/open source software	Compendex	sim	sim		CI1
10.	Agile, open source, distributed, and on-time - Inside the Eclipse development process	Compendex	não			
11.	An agile approach for integration of an open source health information system	Compendex	sim	não	CE5	
12.	An empirical study on software development with open source components in the Chinese software industry	Compendex	obtido com autores	não	CE4	
13.	An empirical study on the relationship between software design quality, development effort, and governance in open source projects	WebOfScience	sim	não	CE4	
14.	An open source approach to developing software in a small organization	WebOfScience	sim	não	CE5	
15.	Applied quality assurance methods under the open source development model	Compendex	sim	não	CE4	
16.	Back-propagation of user innovations: The open source compatibility edge	Scopus	sim	não	CE6	
17.	Bayesian network to construct interoperability model of open source software	Compendex	sim	não	CE5	
18.	Boundary Organizations: Enabling Collaboration among Unexpected Allies	WebOfScience	não			
19.	Coase's penguin, or, Linux and The Nature of the Firm	WebOfScience	sim	não	CE4	
20.	Collaboration rules	Compendex	sim	não	CE4	
21.	Collaboration, peer review and open	WebOfScience	sim	sim		CI2

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
	source software					
22.	Commercial adoption of open source software: An empirical study	Compendex	sim	sim		CI2, CI4
23.	Competing fairly in the new economy: Lessons from the browser wars	WebOfScience	sim	não	CE6	
24.	Comprehensive commercialization of open source software	Compendex	sim	não	CE4	
25.	Contemplating open source enterprise systems	WebOfScience	sim	não	CE5	
26.	Conventional and open source software reuse at orbotech - An industrial experience	Compendex	sim	não	CE5	
27.	Corporate-, Agile- and Open Source Software development: A witch's brew or an elixir of life?	Compendex	obtido com autores	não	CE8	
28.	Designing a forecasting analysis to understand the diffusion of open source software in the year 2010	WebOfScience	sim	não	CE4	
29.	Development of a quality assurance framework for the open source development model	Compendex	sim	não	CE4	
30.	Easier said than done: An empirical investigation of software design and quality in open source software development	Compendex	sim	não	CE4	
31.	Economics of Linux adoption in developing countries	IEEE	sim	não	CE5	
32.	Embracing Open Source Methods for the Standardization of NGN Services and Enablers	Compendex	sim	não	CE3	
33.	Evaluating the post-delivery fault reporting and correction process in closed-source and open-source software	Compendex	sim	não	CE3	
34.	Evaluation of source code copy detection methods on FreeBSD	Compendex	sim	não	CE4	
35.	Evolution in open source software: a case study	Compendex	sim	não	CE4	
36.	Exploring the maintenance process through the defect management in the open source projects - four case studies	Compendex	sim	não	CE4	
37.	Framework for governance in open source communities	Compendex	sim	não	CE4	
38.	From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development	WebOfScience	sim	não	CE6	
39.	From Peer Production to Productization: A Study of Socially Enabled Business Exchanges in Open Source Service Networks	WebOfScience	sim	não	CE6	
40.	From research software to open source	WebOfScience	obtido com autores	não	CE4	
41.	Harvesting altruism in open-source software development	WebOfScience	sim	não	CE6	
42.	Increasing returns and the diffusion of linux in China	Compendex	sim	não	CE4	
43.	Indirectly predicting the maintenance effort of open-source software	WebOfScience	não			
44.	Industry trends and software assurance	Compendex	não			
45.	Innovation in open source software with knowledge - Three challenges for open source competence centres	Compendex	sim	não	CE4	
46.	Models for the evolution of OS	Compendex	sim	não	CE3	

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
	projects					
47.	Motivation, governance, and the viability of hybrid forms in open source software development	WebOfScience	sim	não	CE4	
48.	Open Collaboration within Corporations Using Software Forges	IEEE	sim	sim		CI1, CI3, CI4
49.	Open knowledge management: Lessons from the open source revolution	WebOfScience	sim	sim		CI2
50.	Open source and open content: a framework for global collaboration in social-ecological research	WebOfScience	sim	não	CE3	
51.	Open source development and Agile methods	Compendex	não			
52.	Open source enterprise systems: Towards a viable alternative	Scopus	sim	não	CE5	
53.	Open source research - the power of us	WebOfScience	sim	não	CE4	
54.	Open source software - An evaluation	WebOfScience	obtido com autores	sim		CI1
55.	Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future	WebOfScience	não			
56.	Open source software peer review practices: A case study of the apache server	Compendex	sim	não	CE4	
57.	Open source software research activities in NTT Laboratories	Compendex	sim	não	CE4	
58.	Open source software usage implications in the context of software development	Compendex	sim	não	CE4	
59.	Open source software: Free isn't exactly cheap!	Compendex	não			
60.	Open source software: The future ahead	WebOfScience	sim	não	CE4	
61.	Open source versus closed source: Software quality in monopoly and competitive markets	WebOfScience	sim	não	CE3	
62.	Open source vs. closed source	Compendex	sim	não	CE3	
63.	Opening minds: Cultural change with the introduction of open-source collaboration methods	WebOfScience	não			
64.	Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy	WebOfScience	obtido com autores	não	CE6	
65.	Public participation in proprietary software development through user roles and discourse	WebOfScience	sim	não	CE3	
66.	Quality assurance under the open source development model	WebOfScience	sim	não	CE4	
67.	Realizing the benefits of the CMMISM with the CeBASE Method	Compendex	sim	não	CE4	
68.	Self-organization process in open-source software: An empirical study	WebOfScience	sim	não	CE3	
69.	Sharing knowledge across boundaries	WebOfScience	obtido com autores	não	CE4	
70.	Silver bullet or fool's gold: supporting usability in open source software development	Compendex	sim	não	CE4	
71.	Software development as spiritual metaphor	Compendex	sim	não	CE3	
72.	Software in the year 2010	IEEE	sim	não	CE4	
73.	Software innovativeness. A comparison between proprietary and Free/Open Source solutions offered by Italian SMEs	WebOfScience	sim	não	CE4	

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
74.	Software production infrastructure to support agile methodologies	Compendex	sim	não	CE5	
75.	Techniques and processes for improving the quality and performance of open-source software	Compendex	sim	não	CE4	
76.	The Impact of Open Source Software on the Strategic Choices of Firms Developing Proprietary Software	WebOfScience	sim	não	CE6	
77.	The link between incentives and product performance in open source development: An empirical investigation	Scopus	obtido com autores	não	CE4	
78.	The transformation of open source software	WebOfScience	obtido com autores	não	CE4	
79.	The value of open standards and open-source software in government environments	WebOfScience	não			
80.	Towards a comprehensive approach for assessing open source projects	Compendex	não			
81.	Towards a process maturity model for open source software	Compendex	sim	sim		CI5
82.	Two case studies of open source software development: Apache and Mozilla	WebOfScience	sim	não	CE4	
83.	Understanding the requirements for developing open source software systems	Compendex	sim	não	CE4	
84.	Up from alchemy [open source development]	Compendex	sim	não	CE4	
85.	Which Kind of Collaboration Is Right for You?	WebOfScience	sim	não	CE3	
86.	Why open source software can succeed	WebOfScience	sim	não	CE4	

Tabela 29 – Dados das publicações da Questão Principal C

Apêndice D – Dados das Publicações do Segundo Filtro da QPd

A Tabela 30 contém a listagem dos 94 artigos da QPd analisados no segundo filtro. Para cada publicação são detalhadas as seguintes informações: título, base de dados de onde essa publicação foi selecionada, disponibilidade do arquivo com o texto completo do artigo, resultado da análise do segundo filtro e o critério de inclusão ou exclusão aplicado. Os artigos selecionados foram realçados.

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
1.	A Case Study: Introducing eXtreme Programming in a US Government System Development Project	IEEE	sim	não	CE4	
2.	A Lightweight Software Design Process for Web Services Workflows	IEEE	sim	não	CE3	
3.	A product manager's guide to surviving the big bang approach to agile transitions	Compendex	sim	não	CE3	
4.	A project retrospectives method in telecom software development	Compendex	sim	não	CE3	
5.	A Soft-Structured Agile Framework for Larger Scale Systems Development	IEEE	sim	sim		CI1, CI3
6.	A structured agile design approach to support customisation in wellness product development	Compendex	não			
7.	Adapting PROFES for use in an agile process: An industry experience report	Compendex	sim	sim		CI1, CI3
8.	Adaptive agility	Compendex	sim	não	CE4	
9.	Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course	WebOfScience	obtido com autores	sim		CI1
10.	Adept: A unified assessment method for small software companies	Compendex	sim	não	CE6	
11.	Adopting Agile in Distributed Development	IEEE	sim	não	CE4	
12.	Adopting best practices: "Agility" moves from software development to healthcare project management	WebOfScience	não			
13.	Agile customer engagement: A longitudinal qualitative case study	Compendex	sim	não	CE4	
14.	Agile deployment: Lean service management and deployment strategies for the SaaS enterprise	Compendex	sim	não	CE3	
15.	Agile development in computer science education: Practices and prognosis	Compendex	sim	não	CE3	
16.	Agile development: Good process or bad attitude?	WebOfScience	obtido com autores	sim		CI1, CI2
17.	Agile practices in software development - Experiences from student projects	Compendex	sim	não	CE4	
18.	Agile Program Management: Lessons Learned from the VeriSign Managed Security Services Team	IEEE	sim	não	CE4	
19.	Agile project management - Agilism versus traditional approaches	WebOfScience	sim	não	CE4	
20.	Agile requirements engineering for a social insurance for occupational risks organization: A case study	Compendex	sim	não	CE4	
21.	Agile Requirements Engineering Practices: An Empirical Study	IEEE	sim	não	CE4	
22.	Agile Software Process and its	Compendex	sim	não	CE4	

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
	experience					
23.	Agile, open source, distributed, and on-time - Inside the Eclipse development process	Compendex	não			
24.	Agility counts in developing small-size software	IEEE	sim	não	CE4	
25.	Agility versus discipline: Is reconciliation possible?	Compendex	sim	não	CE7	
26.	An agile approach for integration of an open source health information system	Compendex	sim	não	CE5	
27.	An experiment in teaching innovation in software engineering : Video presentation	Compendex	sim	não	CE3	
28.	Anatomy of a failed agile adoption	Scopus	sim	não	CE4	
29.	Applying Agile Principles for Distributed Software Development	IEEE	sim	sim		CI1, CI2, CI3
30.	Army simulation program balances agile and traditional methods with success	Compendex	sim	sim		CI1, CI4
31.	Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods	Compendex	sim	sim		CI1, CI3
32.	Balancing the human and the engineering factors in software development	Compendex	sim	sim		CI3, CI4
33.	Becoming Agile using Service Learning in the Software Engineering Course	IEEE	sim	não	CE3	
34.	Blending agile development methods with CMMI	Scopus	não			
35.	Can agile and traditional systems development approaches coexist? An ambidextrous view	WebOfScience	sim	sim		CI1, CI3
36.	Can an Agile software development team also embrace CMMI?	Compendex	não			
37.	Certifying for CMM Level 2 and IS09001 with XP@Scrum	IEEE	sim	sim		CI1, CI2, CI4
38.	Collaboration, Process Control, and Fragility in Evolutionary Product Development	IEEE	sim	não	CE3	
39.	Combining Agile methods with stage-gate project management	Compendex	sim	sim		CI1
40.	Combining agile software projects and large-scale organizational agility	Compendex	não			
41.	Corporate-, Agile- and Open Source Software development: A witch's brew or an elixir of life?	Compendex	obtido com autores	não	CE8	
42.	Deploying agile practices in organizations: A case study	Compendex	sim	não	CE4	
43.	Designing an agile methodology for mobile software development: A hybrid method engineering approach	IEEE	sim	sim		CI1, CI3
44.	Development practices for small software applications	Compendex	sim	não	CE4	
45.	Documentation in Systems Development: A Significant Criterion for Project Success	IEEE	sim	não	CE3	
46.	Enabling software process improvement in agile software development teams and organisations	Compendex	sim	não	CE4	
47.	Establishing an agile portfolio to align IT investments with business needs	Compendex	sim	não	CE6	
48.	Experiences in applying agile software development practices in new product development	Compendex	sim	não	CE3	
49.	Experiences using Agile Software	IEEE	sim	sim		CI1, CI4

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
	Development for a Marketing Simulation					
50.	Experimenting with agile practices - First Things First	Compendex	sim	não	CE4	
51.	Extreme customer innovation in the front-end: Learning from a new software paradigm	Compendex	sim	não	CE4	
52.	Factors that impact implementing an agile software development methodology	IEEE	sim	não	CE4	
53.	Formalizing agility, part 2: How an agile organization embraced the CMMI	Compendex	sim	sim		CI1, CI4
54.	Future implementation and integration of Agile methods in software development and testing	Compendex	sim	sim		CI1, CI2, CI3
55.	Get ready for agile methods, with care	IEEE	sim	não	CE7	
56.	How do Agile/XP development methods affect companies?	Compendex	sim	não	CE2	
57.	Integrating agile software development and software process improvement: A longitudinal case study	Compendex	sim	não	CE7	
58.	Integration of software engineering techniques through the use of architecture, process, and people management: An experience report	Compendex	sim	não	CE4	
59.	Introducing an agile process to an organization [software development]	IEEE	sim	não	CE4	
60.	Is extreme programming just old wine in new bottles: A comparison of two cases	Compendex	sim	não	CE4	
61.	Knowledge sharing in traditional and agile software processes	Compendex	não			
62.	Lessons learned in using agile methods for process improvement	Compendex	sim	não	CE4	
63.	Lightening the Software Production Process in a CMM Level 5 Framework	IEEE	sim	não	CE1	
64.	Light-weight development method: A case study	Compendex	sim	não	CE4	
65.	Management challenges to implementing agile processes in traditional development organizations	Compendex	sim	sim		CI1, CI2
66.	Mapping CMMI project management process areas to SCRUM practices	Compendex	sim	sim		CI1, CI2, CI3
67.	Mastering dual-shore development - The tools and materials approach adapted to agile offshoring	Compendex	sim	sim		CI1, CI3, CI4
68.	Mature agile with a twist of CMMI	Compendex	sim	sim		CI1, CI3, CI4
69.	Measuring and comparing the adoption of software process practices in the software product industry	Compendex	obtido com autores	não	CE3	
70.	Mission-critical development with XP agile processes	Compendex	sim	não	CE4	
71.	Moving from a plan driven culture to agile development	Compendex	sim	não	CE2	
72.	Project management in plan-based and Agile companies	Compendex	sim	não	CE4	
73.	Quantitative process improvement in XP using six sigma tools	Compendex	sim	sim		CI1, CI3
74.	Reconciling agility and discipline in COTS selection processes	Compendex	sim	sim		CI1, CI3
75.	Requirements engineering and agile software development	IEEE	sim	sim		CI1, CI3
76.	Rolling the DICE for Agile Software	Compendex	sim	não	CE3	

#	Título	Base	Texto completo	2o. Filtro	Critério Exclusão	Critério Inclusão
	Projects					
77.	Scrum and CMMI level 5: The magic potion for code warriors	Compendex	sim	não	CE7	
78.	Selection criteria for software development tools for SMES and Cooperatives in Venezuela	Compendex	não			
79.	Single goal set: A new paradigm for IT megaproject success	Compendex	sim	não	CE3	
80.	Software production infrastructure to support agile methodologies	Compendex	sim	não	CE5	
81.	Software quality and assurance in waterfall model and XP - A comparative study	Compendex	sim	não	CE3	
82.	Software quality assurance in XP and spiral - A comparative study	Compendex	sim	não	CE3	
83.	Stretching agile to fit CMMI level 3 - the story of creating MSF for CMMI process improvement at Microsoft corporation	Compendex	sim	sim		CI1, CI3
84.	The impact of agile practices on communication in software development	Compendex	sim	não	CE4	
85.	The impact of methods and techniques on outcomes from agile software development projects	Scopus	sim	não	CE4	
86.	The role of extreme programming in a plan-driven organization	Scopus	sim	sim		CI1, CI2, CI3, CI4
87.	The role of knowledge creation in adopting extreme programming model: An empirical study	Compendex	sim	não	CE3	
88.	Toward maturity model for extreme programming	IEEE	sim	sim		CI5
89.	Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study	IEEE	sim	não	CE4	
90.	Using return on investment to compare agile and plan-driven practices in undergraduate group projects	Compendex	sim	não	CE4	
91.	Using risk to balance agile and plan-driven methods	Compendex	sim	sim		CI1, CI3
92.	Using Simulation to Investigate Requirements Prioritization Strategies	IEEE	sim	não	CE4	
93.	When software engineers met research scientists: A case study	Compendex	sim	sim		CI1
94.	XP and the CMM	Compendex	sim	sim		CI1, CI2, CI3

Tabela 30 – Dados das publicações da Questão Principal D