A INFLUÊNCIA DE FATORES NA PRODUTIVIDADE DO DESENVOLVIMENTO DE SOFTWARE DE ACORDO COM UM MODELO DE ESTRUTURAS TEÓRICAS

Wladmir Araujo Chapetta

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Guilherme Horta Travassos

Rio de Janeiro
Dezembro de 2018

A INFLUÊNCIA DE FATORES NA PRODUTIVIDADE DO DESENVOLVIMENTO DE SOFTWARE DE ACORDO COM UM MODELO DE ESTRUTURAS TEÓRICAS

Wladmir Araujo Chapetta

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Guilherme Horta Travassos, D.Sc.

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

Prof. Toacy Cavalcante de Oliveira, D.Sc.

Dr. Carlos Henrique Cabral Duarte, Ph.D.

Profª. Andreia Malucelli, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2018

Chapetta, Wladmir Araujo

A Influência de Fatores na Produtividade do Desenvolvimento de Software de acordo com um Modelo de Estruturas Teóricas / Wladmir Araujo Chapetta. – Rio de Janeiro: UFRJ/COPPE, 2018.

XIII, 215 p.: il.; 29,7 cm.

Orientador: Guilherme Horta Travassos

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 119-139.

1. Produtividade do Desenvolvimento de Software. 2. Medição de Software. 3. Engenharia de Software Experimental. I. Travassos, Guilherme Horta. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Ao amor da minha vida, minha esposa, sem a qual este trabalho não teria sido concluído. Ao meu filho, pelo qual sempre vale a pena combater o bom combate.

Agradecimentos

Em primeiro lugar, à minha esposa, Mariana, pelo apoio e amor incondicionais. Viver ao seu lado é minha razão de viver.

Ao meu filho Luciano, sempre ao lado, me ensinando muito, me mostrando diariamente as coisas que importam e o que a vida pode me oferecer de melhor.

Aos meus pais por terem me dado todas as oportunidades e uma educação para poder chegar até aqui.

Ao meu orientador, Prof. Guilherme Horta Travassos, por ter aceitado mais uma vez minha companhia. Sob sua tutela, comecei realmente a aprender o que ciência significa. Minha admiração pela sua determinação em fazer ciência em um alto nível. É uma honra ter sido seu aluno por todos estes anos em diferentes etapas da minha vida.

Aos integrantes da banca Ana Regina Cavalcanti da Rocha, Andreia Malucelli, Carlos Henrique Cabral Duarte e Toacy Cavalcante de Oliveira, por terem aceitado o convite para a minha defesa de tese e darem suas contribuições para sua melhoria.

Ao Paulo Sérgio Medeiros dos Santos, pelas conversas e discussões nos mais diversos temas.

Ao Grupo de Engenharia de Software Experimental da COPPE/UFRJ, pelas sugestões, contribuições e o agradável convívio.

Ao pessoal técnico-administrativo do PESC, pela atenção e o zelar com os alunos.

Ao Raphael Machado e Rodolfo Sabóia pelo apoio e compreensão no dia-a-dia do trabalho.

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

A INFLUÊNCIA DE FATORES NA PRODUTIVIDADE DO DESENVOLVIMENTO DE SOFTWARE DE ACORDO COM UM MODELO DE ESTRUTURAS TEÓRICAS

Wladmir Araujo Chapetta

Dezembro/2018

Orientador: Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta um modelo baseado em evidências que descreve efeitos de alguns fatores na produtividade do desenvolvimento de software, obtidos através de um método de síntese de evidências em Engenharia de Software. Deste modo, as relações entre um conjunto de fatores e a produtividade do desenvolvimento de software (fenômenos observados) são descritas como resultados da combinação de estruturas teóricas capazes de expressar e tratar diferenças entre efeitos e incertezas variadas de acordo com os tipos de estudos primários encontrados na literatura. Além disso, para avaliar o modelo encontrado, seus achados são confrontados com uma pesquisa de opinião realizada para capturar a percepção de profissionais da prática (gestores e líderes de projetos de software em organizações brasileiras). O grau de concordância entre a pesquisa (o modelo) e a prática (a percepção dos profissionais) demonstra que, aparentemente, o conhecimento científico não diverge consideravelmente da realidade vivenciada pelos projetos de software no Brasil, quando ambos se referem à influência de fatores na produtividade do desenvolvimento de software. Persiste a impressão, entretanto, de que a pesquisa e a prática no tema percorrem caminhos distintos. De acordo com este trabalho, a impressão do distanciamento parece estar relacionadas à questão do uso de medidas não-padronizadas e, talvez, inapropriadas para mensurar os fatores e a produtividade do desenvolvimento de software.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

THE INFLUENCE OF FACTORS ON SOFTWARE DEVELOPMENT PRODUCTIVITY ACCORDING TO A MODEL OF THEORETICAL STRUCTURES

Wladmir Araujo Chapetta

December/2018

Advisor: Guilherme Horta Travassos

Department: Systems Engineering and Computer Science

This work presents an evidence-based model describing the effects of a set of factors on software development productivity, obtained through an evidence synthesis method in Software Engineering. Thus, the relationships among this set and the software development productivity (observed phenomena) are described as results of combining theoretical structures capable of expressing and dealing with differences between different effects and uncertainties varying according to the types of studies found in the literature. Besides, to evaluate the model found, its findings are confronted with a survey capturing the practitioners' perception (managers and leaders of software projects in Brazilian organizations). The degree of agreement between research (the model) and practice (the practitioners' perception) shows that scientific knowledge does not differ considerably from the reality experienced by software projects when both of them refer to the influence of factors on software development productivity. The impression that research and practice on the theme go through different paths persists. According to this work, the reasons for this impression are more related to the use of non-standardized and, perhaps, inappropriate measures used to perceive and monitor the influence of factors as well as to measure the software development productivity.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1: Introdução

Neste capítulo são apresentadas as motivações e objetivos deste trabalho, bem como são apresentados a organização de conteúdo deste manuscrito.

1.1. Motivação

Na indústria de software contemporânea, as organizações visam manter ou melhorar sua vantagem competitiva e crescimento contínuo em relação aos seus competidores. Neste cenário, estas organizações tentam mensurar sua capacidade de produzir e rentabilizar um fluxo de melhorias capazes de agregar valor a produtos e serviços de acordo com as necessidades dos clientes, inclusive no que diz respeito às suas exigências de qualidade. Este fluxo pode ser alcançado modificando atributos de processos, produtos ou serviços.

Ao lidar com processos, os efeitos de tais modificações visam maximizar a produção consumindo o mínimo de recursos e alcançar objetivos dentro ou superando expectativas de *stakeholders*. Estes efeitos devem ser medidos e controlados para serem percebidos, ou seja, a mudança no desempenho de um processo deve ser gerenciada.

De acordo com a família de normas ISO 9000 (ABNT, 2015a), a medição de desempenho deve estabelecer os métodos para medir a eficácia e a eficiência dos processos produtivos das organizações, como uma forma de acompanhar o progresso das iniciativas de melhoria em direção aos seus objetivos. O termo eficácia diz respeito à medida em que as atividades planejadas são executadas e os resultados esperados são alcançados. Já a eficiência diz respeito à relação entre os resultados alcançados e os recursos utilizados, ou seja, diz respeito ao caminho para se chegar no resultado.

Assim, as organizações devem buscar a gestão de desempenho como uma boa prática para produzir os resultados esperados em condições repetitivas, de preferência para consumir recursos sem folgas ou perdas, e para avaliar mudanças em seu ambiente organizacional continuamente. No contexto do desenvolvimento de software, pode-se dizer que folgas estão principalmente relacionadas à ociosidade da força de trabalho e perdas estão principalmente relacionadas ao retrabalho na execução de atividades. A questão da gestão de desempenho também pode ser

observada em modelos com foco na gestão da qualidade de processos de software, tais como, CMMI-DEV (SEI, 2010) e MPS-SW (SOFTEX, 2016).

Já a produtividade geralmente se refere à relação entre unidades de produção por recursos consumidos (Aquino & Meira, 2009; Monteiro & de Oliveira, 2011; Petersen, 2011). Nesse cenário, a produtividade pode ser interpretada como uma dimensão da eficiência de um processo.

Medir produtividade é essencial para controlar e melhorar o desempenho do processo de desenvolvimento (Mendes, Di Martino, Ferrucci, & Gravino, 2008; Mishra & Shukla, 2013). A medição da produtividade tem um papel importante em atividades de melhoria de processos de software e permite, por exemplo, que reavaliações sejam realizadas uma vez que as melhorias foram implementadas (M. Tsunoda & Amasaki, 2017). Além disso, medições da produtividade são importantes como referências para determinar se há necessidade de melhorias em projetos ou processos para que as organizações se mantenham competitivas. Dentro de um *benchmarking* entre organizações, medir produtividade é relevante para identificar os melhores desempenhos e, em seguida, aprender com estas organizações (Cheikhi, Al-Qutaish, & Idri, 2012; Parthasarathy & Sharma, 2016).

Um outro aspecto a se considerar é que “não se pode predizer nem controlar o que não se pode medir” (Fenton & Pfleeger, 1997). De acordo com Petersen (2011), a predição de produtividade de software busca quantificar a forma como a produtividade de um processo de software será no futuro. Predições de produtividade são úteis para determinar se ações corretivas são necessárias, por exemplo, quando os dados indicam um declínio da produtividade e o modelo de predição mostra que esta tendência provavelmente permanecerá. A predição da produtividade também deveria auxiliar gestores na avaliação de alternativas de melhoria, por exemplo, verificar como a produtividade muda se for implementada uma melhoria em atividades de teste. Desta forma, a predição da produtividade refere-se a determinar se ações corretivas são necessárias, e a descobrir quais ações seriam alternativas de melhoria viáveis para produzir os melhores resultados.

Sem uma fase de manufatura (Santos, 2015), a medição e a predição da produtividade do desenvolvimento de software são difíceis, porque não está claro como a produtividade pode ser observada quando o produto é software. Uma vez que a maior parte do esforço gasto no desenvolvimento de software é altamente criativo e dependente do ser humano, as saídas das atividades relacionadas ao desenvolvimento variam de acordo com cada solução de software. Do ponto de vista

do processo de desenvolvimento de software, a produtividade tem sido estudada há pelo menos três décadas no âmbito acadêmico (Aquino & Meira, 2009; Hannay & Benestad, 2010; Monteiro & de Oliveira, 2011; Petersen, 2011), sendo ainda um tema de pesquisa essencial e reconhecido (Lavazza, Morasca, & Tosi, 2016; Pai, Subramanian, & Pendharkar, 2015; Ramírez-Mora & Oktaba, 2017). Muitos métodos para medir a produtividade do desenvolvimento de software têm sido propostos e avaliados experimentalmente, mas estes não parecem atrair os profissionais da prática para o seu uso (Fritz, Mark, Murphy, & Zimmermann, 2017).

O cenário descrito acima revela um desafio para a Engenharia de Software Experimental (ESE): buscar, por meio de evidências, a construção e aperfeiçoamento de um conhecimento geral sobre um tema específico (produtividade do desenvolvimento de software) que seja explicitado a partir de um conjunto de proposições que, formuladas, podem ser observadas e comparadas com resultados de estudos experimentais novos ou já executados, ou em processos de desenvolvimento das organizações que desenvolvem software. Assim, a expectativa é que este conhecimento possa servir como um *baseline* para novas investigações em um futuro próximo, ou para que decisões e ações sobre melhorias no processo de desenvolvimento de software possam ser realizadas na prática. Certamente, da mesma forma que outras disciplinas científicas como a Medicina, que se beneficiam de uma visão mais homogeneizada do conhecimento, a Engenharia de Software pode tentar tirar proveito de uma maior uniformização do seu conhecimento (Santos, 2015).

1.2. Problema

A literatura técnica confirma o interesse científico na produtividade do desenvolvimento de software, embora questões em aberto ainda persistam em relação ao tema (Aquino & Meira, 2009; Cataldo & Herbsleb, 2013; Duarte, 2017; Colomo-Palacios, Casado-Lumbreras, Soto-Acosta, García-Peñalvo, & Tovar, 2014; de O. Melo, S. Cruzes, Kon, & Conradi, 2013; Fritz *et al.*, 2017; Hernández-López, Colomo-Palacios, Soto-Acosta, & Lumberas, 2015; Jeffery, Staples, Andronick, Klein, & Murray, 2015; Meyer, Fritz, Murphy, & Zimmermann, 2014; Meyer, Zimmermann, & Fritz, 2017; Monteiro & de Oliveira, 2011; Nguyen-Duc, Cruzes, & Conradi, 2015; Paiva, Barbosa, Lima, & Albuquerque, 2010; Petersen, 2011; Ramasubbu, Bharadwaj, & Tayi, 2015; Ramírez-Mora & Oktaba, 2017; Scholtes, Mavrodiev, & Schweitzer, 2016; Yilmaz, O'Connor, & Clarke, 2016), que em linhas gerais se resumem em:

- O que significa produtividade de/para indivíduos, equipes e organizações e como medi-la em cada caso?

    - Que medidas representam entradas e saídas em cada um desses grãos (ex: Hernández-López *et al.*, 2015)? Em qualquer domínio (ex: Jeffery *et al.*, 2015; Ramírez-Mora & Oktaba, 2017)?
    - Medidas baseadas em linhas de código/pontos de função são adequadas para representar saídas do processo de desenvolvimento (ex: Aquino & Meira, 2009)? Se não, quais as opções (ex: Scholtes *et al.*, 2016; Duarte, 2017)?
- Quais são os fatores que afetam a produtividade do desenvolvimento de software e como eles a afetam (ex: de Barros Sampaio *et al.*, 2010; Paiva *et al.*, 2010; Wagner & Ruhe, 2018)? E quais medidas são usadas ou adequadas para operacionalizá-los (ex: Monteiro & de Oliveira, 2011; Yilmaz *et al.*, 2016)?
- Quais as implicações de medir produtividade com os métodos e medidas hoje disponíveis? Quais os ajustes necessários para utilizá-los em organizações de software (Meyer *et al.* 2017)?
- Quais e como conduzir estudos que podem ajudar a endereçar os pontos acima citados (Fritz *et al.*, 2017)?

Propriedades específicas do desenvolvimento de software, tais como a rápida evolução das tecnologias e ausência de fase de manufatura, fazem com que em muitas circunstâncias o conhecimento da prática seja o único disponível, pelo menos em um primeiro momento (Santos, 2015). Devido a estas peculiaridades e por questões de oportunidade, a pesquisa em ES tem focado mais na execução de estudos primários e, posteriormente, secundários do que na elaboração e discussão de uma fundamentação teórica a ser melhor estudada, compreendida e posta em prática.

Em parte, isso pode explicar os problemas que encontramos ao tentar combinar ou agregar resultados de estudos em Engenharia de Software (ES) usando métodos tradicionais de meta-análise (como a ausência de uso de medidas comuns, a ausência de grupos de controle em estudos primários, a falta de informação estatística, dados faltantes, entre outros) (Miller, 2000, 2005). Mesmo que estudos controlados pudessem ser empregados para investigar qualquer fenômeno em Engenharia de Software, a quantidade e variedade de contextos onde tecnologias de desenvolvimento de software devem ser observadas torna praticamente inviável a configuração de estudos para contemplar todos os casos (Kitchenham, 2007; Dybå, Sjøberg, & Cruzes, 2012).

Tais fatos tornam difícil para pesquisadores observar um mesmo fenômeno em diferentes organizações, processos, projetos, atividades e participantes, de modo que

hipóteses e medidas sejam compartilhadas e seus resultados facilmente replicados (Beecham, OLeary, Richardson, Baker, & Noll, 2013). Isto torna a produtividade complexa de medir no contexto de desenvolvimento de software e dificulta que os conhecimentos produzidos em diferentes iniciativas sejam capazes de sensibilizar profissionais que atuam em organizações de desenvolvimento de software (Fritz *et al.*, 2017; Meyer *et al.*, 2017).

Petersen (2011) percebe essa falta de homogeneidade do conhecimento no tema e atenta que as considerações de Scacchi (1991) e Dale & van der Zee (1992) deveriam ser atualizadas (ambos os estudos do início dos anos 1990) e devidamente agregadas à literatura técnica de produtividade no contexto do desenvolvimento de software. Embora ambos os trabalhos tenham sido considerados em estudos posteriores (Hernández-López, Colomo-Palacios, García-Crespo, & Cabezas-Isla, 2013; Trendowicz & Münch, 2009; Wagner & Ruhe, 2018), nenhuma síntese que homogeneizasse o conhecimento encontrado até o momento foi realizada, considerando as evidências apenas sob um ponto de vista meramente aditivo e exploratório. E sob este aspecto, como uma disciplina científica, a ES não tem atentado, de forma ampla e sistemática, para a questão de fornecer, de forma suficientemente simples, abstrações conceituais que permitam a sua estruturação de forma concisa e precisa, e que, desta forma, facilitem a comunicação de ideias e conhecimento (Hannay, Sjoberg, & Dyba, 2007). A pesquisa em ES não tem oferecido conhecimento suficientemente abstrato para permitir a sua reflexão, avaliação ou emprego independente de tempo e lugares específicos.

Assim, pesquisadores e profissionais da prática da ES, quando se refere à produtividade do desenvolvimento de software, têm dificuldades em expressar: (1) o que sabem bem; (2) o que não sabem, e; (3) o que deveriam compreender melhor para buscar melhorias nas medidas e nos métodos de medição.

1.3. Objetivos

A partir das diferentes revisões da literatura técnica com discussões relevantes sobre o tema (Scacchi, 1991; Aquino & Meira, 2009; Dale & van der Zee, 1992; Petersen, 2011; Hernández-López, Colomo-Palacios, & García-Crespo, 2012; Hernández-López, Colomo-Palacios, & García-Crespo, 2013; Hernández-López *et al.*, 2015), percebe-se uma certa dificuldade em identificar uma tendência na investigação de fatores, e, consequentemente, relações causais investigadas na temática produtividade do desenvolvimento de software. Em nenhuma das discussões econtradas é proposto identificar um modelo geral de proposições extraídas a partir de

hipóteses, leis ou teorias de um conjunto de evidências. Por este motivo, podemos considerar que há uma certa ausência de consenso sobre quais e como os fenômenos devem ser observados e melhor compreendidos, ou seja, não existe uma definição de questões básicas para orientar pesquisadores de ES em estudos específicos sobre produtividade do desenvolvimento de software.

Por exemplo, Petersen (2011), em sua revisão sistemática, busca mapear o tema produtividade do desenvolvimento de software a partir das propostas de métodos de medição e predição para, então, tentar caracterizar alguns destes elementos. Este autor consegue identificar uma convergência quanto às medidas de entradas e saídas, mas não obtém sucesso ao capturar um modelo geral de produtividade de software devido ao pequeno número de estudos primários selecionados em sua revisão.

Embora não seja um objetivo direto desta tese alcançar suas respostas, a pesquisa experimental precisa traçar estratégias para que as seguintes questões precisam ser colocadas à Comunidade de ES: Como alcançar questões básicas relacionadas a fenômenos que poderiam ser observados no ambiente industrial? Como falar sobre iniciativas de melhoria de processo quando não há um entendimento mínimo entre pesquisadores e profissionais da prática sobre o que deve ser medido? Estas perguntas talvez possam apoiar mais incisivamente a discussão sobre direções na pesquisa básica no tema em ES.

Deste modo, o objetivo desta tese é fornecer um ponto de partida para que algumas destas questões possam ser endereçadas futuramente através de um conjunto de proposições capaz de representar a síntese em um corpo de conhecimento relacionado a produtividade do desenvolvimento de software. Visando contribuir com a evolução do conhecimento e o explicitando de forma concisa e precisa para facilitar a comunicação entre diferentes interessados na Comunidade de ES, surgem as seguintes questões de pesquisa para caracterizar o objeto de estudo desta tese:

RQ1: É possível encontrar proposições que sintetizem hipóteses, leis ou teorias utilizadas em estudos já realizados em produtividade no contexto do processo de desenvolvimento de software?

RQ1.1: Se sim, essas proposições podem ser usadas para oferecer um referencial teórico inicial no tema produtividade do

desenvolvimento de software a ser utilizado em diferentes situações pela Comunidade de ES?

1.4. Metodologia de Pesquisa

Para responder às questões de pesquisa apresentadas, foi proposto selecionar e agregar resultados de estudos primários previamente executados, realizando uma metodologia de pesquisa com as seguintes etapas:

1. ***Executar uma revisão sistemática da literatura*** (*systematic literature review* - SLR): O objetivo do protocolo de revisão sistemática executado foi principalmente caracterizar métodos de medição e predição de produtividade do desenvolvimento de software, com base no trabalho anterior de Petersen (2011). No entanto, nosso protocolo de pesquisa evolui a busca e acrescenta novas questões secundárias de pesquisa visando o aumento populacional. Ao final, 153 artigos são selecionados e classificados quanto ao rigor experimental, o método de medição, medidas utilizadas, fatores levados em conta na medição, dentre outros.
2. ***Identificar os fatores que afetam a produtividade do desenvolvimento de software***: Esta etapa consiste em identificar e descrever qualitativamente conceitos (construtos) usualmente estudados na literatura técnica sobre produtividade do desenvolvimento de software e levantados na etapa anterior, a fim de que sejam utilizados como os fatores do modelo proposto. Como resultado, foram identificados 25 fatores com potencial de síntese, de acordo com perspectivas bem definidas de combinação das evidências.
3. ***Representar e Combinar Evidências***: A partir dos fatores e evidências encontradas nas etapas anteriores, esta etapa combinou os estudos primários e, em seguida, propõe um referencial teórico inicial sob a representação de estruturas teóricas. Ao todo, são descritos 34 fenômenos referentes aos fatores, entre si inclusive, e a produtividade do desenvolvimento de software. Para cada um dos fenômenos também é apontado quanta confiança é possível ter no efeito resultante obtido a partir da combinação das evidências. Também são discutidos como estes novos achados se interceptam com outras proposições já estabelecidas no campo da Engenharia de Sistemas e Software.
4. ***Avaliar os Resultados Obtidos em Campo***: Com base na representação obtida na etapa anterior, um *survey* foi elaborado para capturar a percepção dos profissionais da prática em relação aos

fenômenos descritos. Em seguida, ambos os resultados são confrontados a fim de avaliar a convergência e concordância entre a pesquisa (estruturas teóricas) e prática (percepções dos praticantes). Os resultados mostram que, aparentemente, mesmo com as limitações dos diferentes estudos, a academia tem tido relativo sucesso em identificar e estudar questões relativas à medição de fatores. Deste modo, se há questões que impedem o avanço na adesão dos métodos de medição e predição da produtividade no processo de desenvolvimento, há motivos ainda não bem esclarecidos que impedem a obtenção de um consenso em parte significante da Comunidade de ES.

Cada passo será descrito, apresentando o raciocínio adotado durante as atividades, aspectos e resultados significativos. Deste modo, o restante deste manuscrito relata a metodologia de pesquisa empregada na condução das atividades, apresentando passos, decisões e diferentes questões exploradas em diferentes momentos, estudos realizados e os resultados alcançados. Além disso, serão discutidos a convergência com os resultados expressos pelo corpo de conhecimento proposto e como utilizá-lo para apoiar profissionais e pesquisadores a tomarem decisões sobre seus estudos e iniciativas de melhoria de processos.

1.5. Organização do Manuscrito

Além desta introdução, este texto é composto por mais cinco capítulos: (1) o Capítulo 2 define os conceitos em torno do tema produtividade no contexto do desenvolvimento de software, além de apresentar os detalhes, a repetição e evolução de um protocolo de revisão sistemática utilizado no mapeamento da literatura técnica; (2) o Capítulo 3 apresenta a análise qualitativa realizada para identificar os fatores encontrados na literatura, bem como se estes fatores podem potencialmente ser combinados através da definição de fatores mais gerais, com maior potencial teórico; (3) no Capítulo 4 é apresentada a construção do referencial teórico proposto, com definições, detalhes sobre a abordagem para síntese das evidências, a interpretação dos resultados obtidos e as possibilidades de uso por outros pesquisadores e praticantes; (4) o Capítulo 5 apresenta uma avaliação por comparação do modelo de estruturas teóricas com os resultados obtidos em uma pesquisa de opinião realizada com gerentes e líderes de projetos de TIC em organizações que desenvolvem software na indústria brasileira, e; (5) o Capítulo 6 apresenta as conclusões, abordando e discutindo as contribuições e perspectivas futuras de pesquisa.

Deste modo, esta tese mostra as etapas realizadas para alcançar as respostas sobre a viabilidade da construção do corpo de conhecimento proposto e sua disponibilização à Comunidade de ES, sob a forma de uma representação sintética do estado da arte quando relacionado à questão de fatores que influenciam a produtividade do desenvolvimento de software.

Capítulo 2: Revisão da Literatura

Este capítulo resume as revisões sistemáticas da literatura realizadas para capturar um panorama no tema produtividade do desenvolvimento de software e subsidiar a definição desta proposta de tese.

2.1. Definições e Considerações Iniciais

Diferentes revisões de literatura identificam que os estudos primários citam produtividade, eficiência, desempenho e, muitas vezes, eficácia de forma intercambiável (Aquino & Meira, 2009; Petersen, 2011; Hernández-López *et al.*, 2015; Monteiro & de Oliveira, 2011). Assim, para efeitos deste trabalho, serão adotadas as seguintes definições:

- Produtividade é a relação que descreve a relação entre os resultados e os recursos utilizados para produzi-los (Aquino & Meira, 2009; Monteiro & de Oliveira, 2011; Petersen, 2011; Hernández-López *et al.*, 2015).
- Eficiência é um conceito relativo. Ele compara o que foi produzido, considerando os recursos disponíveis, e o que poderia ser realmente produzido com os mesmos recursos. Assim, quando se comparam duas medidas de produtividade, é possível encontrar as entidades mais eficientes (Soares De Mello, Angulo-Meza, Gomes, & Biondi Neto, 2005).
- Eficácia é a capacidade de alcançar algo ou para produzir o resultado pretendido, ou seja, a extensão na qual as atividades são realizadas e os resultados são alcançados de acordo com um planejamento (ABNT, 2015a). Está relacionado aos objetivos e interesses de uma organização, projeto ou equipe.
- Medição de Desempenho, de acordo com a ISO 9001 (ABNT,2015b) e a ISO 9004 (ABNT, 2010), leva em consideração a eficácia e eficiência e o possível potencial de melhoria.

Produtividade e eficiência podem ser considerados termos intercambiáveis sem perda de significado, mas isto não é necessariamente verdade quando a eficácia é comparada com os outros dois termos (Petersen, 2011). Pois a eficácia está ligada apenas ao que é produzido, sem levar em conta os recursos usados para a produção (Soares De Mello, Angulo-Meza, Gomes, & Biondi Neto, 2005).

2.2. Referencial Teórico Inicial

Algumas revisões da literatura em ES têm discutido aspectos relevantes a respeito do tema produtividade do desenvolvimento de software. Deste modo, resumidamente, algumas das principais questões abordadas são trazidas pelos trabalhos apresentados a seguir:

- A revisão de literatura executada por Scacchi (1991) aponta uma série de ameaças à validade de estudos em ES sobre produtividade do desenvolvimento de software, tais como: pobre definição de medidas (por exemplo, medida de linhas de código e esforço relacionado a todas as atividades do desenvolvimento do ciclo de vida); ausência de relações claras de causa-efeito de fatores, e; desconhecimento sobre a combinação de efeitos entre fatores afetando a produtividade (por exemplo, aspectos gerenciais e técnicos).
- Putnam e Myers (1992) observaram em seu trabalho que perfis da força de trabalho de organizações de software seguiam uma distribuição Rayleigh, propondo uma métrica chamada “produtividade de processo”. Esta medida foi baseada em anos de experiência na gestão de projetos de P&D nas forças armadas dos Estados Unidos, e mais tarde na General Electric Co., envolvendo projetos desenvolvidos nos Estados Unidos, Grã-Bretanha, Austrália e Japão. No entanto, Maxwell *et al.* (1996 apud Aquino & Meira, 2009) demonstraram que uma medida de produtividade simples e baseada em linhas de código descrevia melhor fatos do que a proposta, em um estudo usando um banco de dados de projetos da Agência Espacial Europeia com 99 projetos de oito países europeus.
- Dale & van der Zee (1992) discutiram medidas de produtividade a partir de diferentes perspectivas (desenvolvimento, usuário e gestão). Na perspectiva de desenvolvimento (inclui atividades de requisitos, implementação e verificação e validação), a produtividade é expressa em termos de linhas de código. A respeito desta medida, os autores salientam que linhas de código são somente uma parte das saídas valoráveis produzidas no desenvolvimento de software. O mesmo se aplica ao esforço. Ao definir esforço é necessário decidir que recursos incluir na análise (por exemplo, *staffing* de manutenção, desenvolvedores, testadores, dentre outros). Da perspectiva do usuário, pontos de função, como uma representação do valor entregue ao usuário. Em relação ao uso de pontos de função, os autores apontam que os custos relativos a

interações com os usuários também devem ser levados em consideração. Do ponto de vista gerencial, a análise da produtividade deve se concentrar principalmente em aspectos monetários, ou seja, capturar o valor criado pelo montante de investimentos relacionados a Tecnologia da Informação e Comunicação (TIC). Além disso, os autores alertam que quando comparamos projetos é importante ter uma definição clara de tais medidas, ou seja, o que buscam capturar e como medi-las.

- Em de Aquino & Meira (2009), foi apresentada uma revisão sobre os estudos de medidas e medição de produtividade publicados na literatura técnica. Os autores classificam as medidas identificadas de acordo com alguns critérios e lições aprendidas. Neste artigo, os autores apontam que a medida de saída mais comumente utilizada é linhas de código, e apresentam críticas e paradoxos sobre o assunto publicados em outros trabalhos. Além disso, algumas direções são apresentadas de modo a discutir meios de evoluir o estado da arte na medição da produtividade do desenvolvimento de software.
- Petersen (2011) identificou um total de 38 estudos primários, por meio de uma revisão sistemática da literatura sobre abordagens de medição e predição de produtividade do desenvolvimento de software. Além das evidências encontradas, uma contribuição deste trabalho é propor um esquema de classificação para a pesquisa experimental no tema. O autor também relata ter tido dificuldade de identificar um modelo genérico de proposições comuns ou uma convergência na investigação de fatores nos estudos selecionados, até mesmo quando levados em conta somente estudos de simulação, onde, segundo o autor, estes elementos seriam mais facilmente identificados.
- Hernández-López *et al.* (2012) apresentaram desafios, conceitos e questões gerais na pesquisa em produtividade do desenvolvimento de software que precisam de mais atenção por parte da comunidade científica. Os autores apontam para a necessidade de um consenso sobre os fatores, entradas e saídas úteis e conhecidos para o uso nas medições.
- Hernández-López *et al.* (2013) dão uma visão geral do estado da arte na medição da produtividade em ES, focando em entradas e saídas utilizados para a medição no nível de tarefas executadas por desenvolvedores. Usando um protocolo de revisão sistemática da literatura, o objetivo foi avaliar as entradas e saídas presentes na literatura técnica, a fim de explorar novas medidas de produtividade para profissionais de software.

Este trabalho revelou que são usadas duas medidas diferentes para avaliar os profissionais de engenharia de software: linhas de código por hora e unidades de planejamento de projeto por unidade de tempo.

- Hernández-López *et al.* (2015) executaram uma revisão sistemática da literatura sobre medidas de entradas e saídas utilizadas para a investigação de medidas de produtividade em diferentes perspectivas de processos de software (atividades, stakeholders, dentre outros) e atentaram para uma falta de pesquisas sobre medidas de produtividade para os níveis mais baixos do processo de desenvolvimento (por exemplo, equipe e individual). Além disso, seu estudo destaca que há medidas sendo utilizadas para mensurar a produtividade em diferentes níveis de abstração de um processo ou etapa do desenvolvimento de software. Assim, sua conclusão é que a utilização de mesmas medidas em diferentes níveis de tarefas e etapas deveria ser visto, a princípio, como algo possivelmente controverso.

Estes trabalhos de alguma forma fazem reflexões sobre a literatura técnica disponível, mas abordam a questão da produtividade do desenvolvimento de software sob diferentes perspectivas, sem uma preocupação de discutir de maneira uniforme as principais questões no tema.

A medição de produtividade é construída a partir de três elementos: entradas, saídas e fatores (Hernández-López *et al.*, 2015). De acordo com as revisões encontradas na literatura técnica, sistemáticas ou não, podemos claramente identificar duas frentes de pesquisa no tema: uma relativa à definição de medidas, outra à identificação de fatores. Ambas apoiadas por variados estudos primários.

Quanto às medidas, mesmo com críticas e problemas, linhas de código (*lines of code* - LOC), pontos de função (*function points* - FP) e esforço são as medidas de entrada e saída mais utilizadas em estudos e métodos relacionados à produtividade do desenvolvimento de software (Scacchi, 1991; Aquino & Meira, 2009; Dale & van der Zee, 1992; Hernández-López *et al.*, 2012; Hernández-López *et al.*, 2015).

Por outro lado, em revisões como as de Wagner & Ruhe (2018), Trendowicz & Münch(2009), de Barros Sampaio *et al.* (2010), Paiva *et al.* (2010) e Ramírez-Mora & Oktaba (2017) são encontradas discussões relevantes sobre fatores influenciando a produtividade do desenvolvimento de software. Apesar da variedade de fatores encontrados e a extensão de estudos relacionados, nestas revisões não é possível observar diretamente os efeitos atribuídos a estes fatores, bem como que medidas são

utilizadas para operacionalizá-los e sua correlação com a produtividade do desenvolvimento de software.

A ausência de convergência sobre a investigação de fenômenos a respeito do assunto pode ser atribuída à falta de uma visão homogênea destas duas frentes. Há uma lacuna sobre como relacionar os resultados destas duas frentes e produzir uma visão mais realista dos desafios envolvidos na produtividade do desenvolvimento de software. Para tentar reduzir esta lacuna e subsidiar esta tese, optamos por uma investigação baseada, também, na execução de uma *quasi*-revisão sistemática da literatura e posterior agregação das evidências obtidas.

2.3. Protocolo de pesquisa de Petersen (2011)

Em geral, revisões de literatura tem o intuito de trazer uma melhor compreensão a respeito do tema em investigação. Deste modo, Petersen *et al.* (2015) diferenciam revisões sistemáticas realizadas em ES entre revisões e mapeamentos sistemáticos. Resumidamente, as revisões sistemáticas teriam papel de encontrar evidências com propósito de comparação e síntese, os mapeamentos teriam o objetivo de descrever o cenário da pesquisa dentro de um tema.

Entretanto, Travassos *et al.* (2008) identificam que grande parte das revisões sistemáticas da literatura não apresentam elementos de comparação e, dificilmente, aplicam meta-análise. Desta forma, estudos secundários desta natureza são classificados como *quasi*-revisões sistemáticas da literatura. Esta classificação conceitual e terminológica é derivada de que tal avaliação deve explorar o mesmo rigor e formalismo metodológico de revisões sistemáticas clássicas (como executadas na Medicina por exemplo) para as fases de elaboração e execução do protocolo, mas nenhuma meta-análise, *a priori*, pode ser aplicada devido à ausência de grupos de controle na maior parte dos estudos primários em ES.

Pela abrangência de seus resultados, o protocolo de Petersen (2011) foi tomado como base para a repetição e realização das revisões da literatura contidas nesta tese, pois este trabalho propõe um esquema de classificação de estudos primários e um panorama da pesquisa experimental sobre métodos de medição e predição encontrados na literatura técnica sobre produtividade do desenvolvimento de software. Embora suas questões de pesquisa não contemplem responder sobre medidas, fatores e fenômenos, o autor consegue resultados semelhantes a outros autores no que tange à investigação de medidas.

Os resultados desta revisão se limitam a artigos publicados antes de 2009, são avaliados aproximadamente 1000 entradas de artigos retornados e a extração e análise dos resultados foi finalizada em 2010.

Já as revisões executadas no escopo deste trabalho ocorreram em duas etapas. A primeira etapa consistiu da repetição seguida de uma evolução do protocolo de Petersen (2011), contemplando trabalhos publicados até 2013. A segunda etapa consistiu de uma atualização por *forward snowballing*, para incluir trabalhos publicados ou indexados entre final de 2013 e o final de 2016. Os detalhes da execução destas etapas são apresentados nas seções a seguir.

2.4. Repetição e Evolução do Protocolo de Petersen (2011)

As revisões aqui realizadas centram, também, sua investigação na identificação de métodos de medição e predição usadas para observar a produtividade do desenvolvimento de software com alguma avaliação experimental, mas novas questões de pesquisa foram adicionadas para identificar as medidas e fatores de influência utilizados em tais métodos. Na Tabela 1 são apresentadas as questões propostas para as revisões executadas.

Inicialmente uma repetição do estudo de Petersen (2011) foi planejada, visando obter e atualizar seus resultados, além de possibilitar a verificação da necessidade de ajustes a fim de acomodar a extração das informações relativas às novas questões de pesquisa e, então, garantir a generalização e comparabilidade com os resultados da SLR original.

Todos os artigos selecionados do protocolo original foram utilizados como artigos de controle, ou seja, estes deveriam ser encontrados e incluídos em execuções subsequentes do protocolo definido. Este arranjo foi adotado para garantir a qualidade do procedimento de busca e para minimizar o viés de pesquisadores, considerando a cobertura e escopo de uma revisão sistemática da literatura já conhecida.

Tabela 1. Questões de Pesquisa da Revisão de Literatura (adaptada de Petersen, 2011).

Questão Principal:	RQ1: Existe alguma evidência sobre a precisão/utilidade dos métodos de medição e previsão de produtividade ou eficiência de software?	
	Questões Secundárias:	RQ1.1: Existe alguma medida que possa apoiar os métodos de medição e previsão? Se sim, quais são aplicadas em processos de software específicos?[1]
		RQ1.2: Quais fatores (ou *drivers* ou construtos) foram usados para definir/compor/descrever tais métodos de medição e predição?[1]
Questões Alternativas:	RQ2: Que recomendações podem ser dadas para melhorar metodologicamente os estudos de produtividade (ou eficiência) de software e o empacotamento e apresentação de tais estudos?	
	RQ3: Existem proposições, hipóteses ou teorias comuns que têm sido usadas para justificar estudos de produtividade (ou eficiência) do desenvolvimento de software realizados por pesquisadores de Engenharia de Software? [1]	
Questões de Mapeamento:	RQ4: Como a frequência de abordagens relacionadas a medir e prever a produtividade do desenvolvimento de software mudou ao longo tempo?	
	RQ5: Como a frequência de pesquisas publicadas está distribuída dentro da estrutura de pesquisa de produtividade do desenvolvimento de software?	

[1] questões adicionadas em relação ao protocolo original

Durante o refinamento da repetição, o autor de Petersen (2011) foi contatado em busca de informações adicionais sobre o seu protocolo, que gentil e prontamente compartilhou todos os *dumps* de artigos retornados, *strings* usadas nos motores de busca, formulários e resultados do processo de extração. Com todas estas informações em mãos, ficou decidido não analisar todos os itens que já haviam sido avaliados na pesquisa original, ou seja, as ações de remoção e inclusão dos artigos já avaliados foram simplesmente reproduzidas. Certamente, o esforço de identificar estas ações sobre estes itens foi menor do que para realizar o procedimento de extração novamente.

A *string* de busca do protocolo original foi reexecutada com o objetivo de encontrar artigos não previamente identificados, limitando a observação até o ano de 2013. Também foi incluído o motor de busca da base Scopus (ver Tabela 2), pois dois dos artigos selecionados no protocolo original não estavam sendo mais retornados. Assim, a execução da busca nas bases de periódicos selecionadas foi realizada no dia

14 de Abril de 2014. Seus resultados foram mantidos, mas um movimento de artigos entre as bases de periódicos foi observado.

Tabela 2. Motores de Busca.

Nome	URL
Web of Science	http://www.webofknowledge.com
IeeeXplore	http://ieeexplore.ieee.org/
EngineeringVillage	http://www.engineeringvillage.com
ACM Digital Library	http://dl.acm.org/
Scopus	http://www.scopus.com/

Os resultados obtidos motivaram uma série de rodadas posteriores para avaliar a possibilidade de evolução do protocolo original, através de verificações da repetibilidade dos resultados, a ampliação do espaço amostral e alterações na *string* de busca. Por exemplo, Petersen (2011) relata ter tido dificuldades com os motores de busca incapazes de lidar com *strings* complexas, dividindo sua *string* de busca em *strings* menores combinadas com operadores lógicos apropriados para cada motor de busca. No entanto, essas rodadas foram úteis para verificar que este procedimento não era mais necessário devido à evolução e versões atuais dos motores de busca.

A versão final do protocolo do estudo secundário executado nesta pesquisa é baseada em Biolchini *et al.* (2005). Para organizar a *string* de busca, foi usada a abordagem PICO (Pai *et al.*, 2004), cujo a versão final é:

Title or Abstract: *(("software process" OR "software development" OR "software engineering processes" OR "business information system" OR "software maintenance" OR "software project" OR "open source project" OR "OSS project") AND (productivity OR efficiency OR "process performance" OR "development performance" OR "software performance" OR "project performance" OR "prediction method" OR "measurement method") AND (measure OR metric OR model OR predict OR estimate OR measurement OR estimation OR prediction) AND (empirical OR validation OR evaluation OR experiment OR example OR simulation OR analysis OR study OR interview))*

Após as rodadas de avaliações, todos os critérios e procedimentos de seleção foram explicitados de acordo com Biolchini *et al.* (2005). Os critérios de exclusão, inclusão e aceitação do protocolo original foram utilizados sem modificações. Os critérios de inclusão, mais permissivos, foram aplicados durante a fase de leitura de

títulos e resumos. Os critérios de exclusão foram aplicados durante a fase de leitura completa, na seleção final dos artigos.

O protocolo resultante deste aperfeiçoamento preservou e estendeu os resultados no período de observação de Petersen (2011) e da repetição executada, representando uma evolução. A Tabela 3 mostra os números de artigos retornados pelos motores de busca após a remoção de duplicatas no protocolo evoluído, comparando-os com o conjunto obtido protocolo original sob as mesmas condições.

Tabela 3. Artigos retornados não avaliados até 2008, entre 2009 e 2013 (no protocolo evoluído) e Petersen (2011).

Motor de Busca	Até 2008	Entre 2009 e 2013	Petersen (2011)
Engineering Village	260	279	448
Web of Science	478	269	299
IeeeXplore	163	116	102
ACM Digital Library	58	46	50
Scopus	440	400	N.A

No total foram manipuladas 7087 entradas de artigos, lidos 3209 títulos e resumos de artigos, para se chegar à leitura completa de 318 artigos. A revisão foi finalizada com a seleção de 89 novos artigos, totalizando 127 artigos selecionados quando somados aos 38 anteriormente selecionados no protocolo original. A extração de dados respeita a estrutura do protocolo original a fim de garantir que o novo material selecionado possa ser adequadamente agregado aos resultados anteriormente obtidos. A Figura 1 apresenta resumidamente os passos até a obtenção da lista final de artigos selecionados durante as duas revisões executadas neste trabalho.

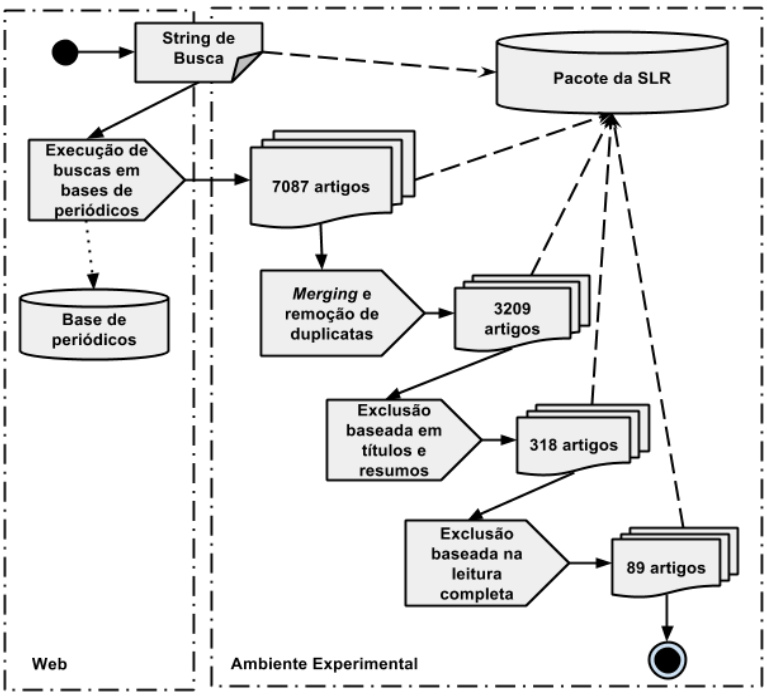

Figura 1. Procedimentos de Execução da Repetição e Evolução do Protocolo de SLR.

Os artigos selecionados como evidências dos métodos encontrados na literatura são listados nos APÊNDICES I, II e III. No primeiro, são listados os artigos selecionados no protocolo original. No segundo, são listados os novos artigos selecionados na repetição. No último, são listados os artigos selecionados no protocolo evoluído. Em um processo incremental, cada etapa de revisão engloba os artigos retornados de seu antecessor.

2.5. Atualização da Revisão por *Forward Snowballing*

Para os estudos primários publicados após 2013, a pesquisa foi realizada de 14 de março de 2017 a 28 de março de 2017, realizando um *forward snowballing*, ou seja, pesquisando os artigos que citavam trabalhos selecionados pelo protocolo SLR. Segundo Felizardo *et al.* (2016), *snowballing* é uma abordagem válida para atualizar resultados de SLR previamente executados de forma iterativa.

O número total de artigos devolvidos pelos motores de busca neste passo foi 813. Após a leitura de títulos e resumos, restaram um conjunto de 133 artigos para a leitura completa. Destes, somente 26 artigos foram incluídos nos artigos selecionados. A lista final de artigos selecionados via *snowballing* pode ser vista no APÊNDICE IV.

Ao final de todo o processo de revisão, o número total de novos artigos selecionados (além daqueles selecionados no protocolo original) é 115. Levando em

conta os artigos do primeiro protocolo, o número total de artigos selecionados é 153. Além disso, os artigos não acessíveis nas bases de períodicos ou cujos autores não puderam responder ou serem localizados foram removidos e corretamente relatados no pacote do estudo.

2.6. Apresentação de Resultados Agregados das Revisões

2.6.1. Considerações Gerais dos Resultados

2.6.1.1. Evidências de Métodos na Pesquisa de Produtividade do Desenvolvimento de Software (RQ1)

Os resultados apresentados nas seções a seguir são relativos ao conjunto total das evidências encontradas, os 153 artigos finais.

Cada artigo selecionado foi classificado quanto à sua abordagem, finalidade, abstração, tipo de avaliação, critérios de avaliação, método de pesquisa e rigor experimental, conforme definido por Petersen (2011). As informações detalhadas sobre a classificação e qualidade (rigor experimental) dos novos artigos selecionados se encontram em Chapetta (2018).

2.6.1.1.1. Principais Medidas Utilizadas Encontradas na Revisão (RQ1.1)

Observando o conjunto de 153 estudos, é possível perceber que a maioria das medidas utilizadas são baseadas em linhas de código (34 citações) e em pontos de função (31 citações). A medida de entrada mais comum continua a ser o esforço (homem-hora ou derivações) (42 citações).

Mesmo considerando que aparentemente não há divergências sobre a utilização de esforço como uma medida de entrada para os processos de software, o mesmo não pode ser dito se observamos as medidas de saída. Medidas baseadas em LOC e FP são utilizadas praticamente na mesma proporção, quando observado o conjunto de estudos primários selecionados.

Embora os números mudem, tais observações já haviam sido apresentadas em Petersen (2011) e permanecem válidas, mesmo considerando que o número final de estudos selecionados em seu estudo seja consideravelmente menor.

Deste modo, pode-se afirmar que as questões relacionadas às entradas e saídas de processos de software utilizadas para medir produtividade no contexto do processo de desenvolvimento talvez não tenham alcançado o consenso necessário

entre pesquisadores e profissionais, já que todas as três medidas são referenciadas em menos de 28% do total selecionado.

2.6.1.1.2. Fatores Observados nos Métodos Encontrados na Revisão (RQ1.2)

Petersen (2011) reporta ter sido difícil capturar os fatores que afetam a produtividade do desenvolvimento de software devido ao pequeno número de estudos encontrados em sua revisão. No entanto, todos os 153 estudos selecionados ao final da revisão realizada foram revisitados em busca de fatores com os níveis de confiança explicitamente expressos ou que demonstram diferença significativa entre tratamentos em estudos experimentais, quando aplicáveis.

Com base em uma investigação inicial dos estudos selecionados, foram identificados relatos de 83 diferentes fatores que afetam a produtividade do desenvolvimento de software (ver Figura 2), o que pode ser consequência de uma possível falta de uma taxonomia dos termos utilizados nos estudos primários neste tema.

Figura 2. Fatores relatados nos estudos primários.

Por estes motivos, esta pesquisa optou por incluir uma etapa seguinte para investigar qualitativamente estes relatos com o intuito de identificar congruências e construtos comuns que poderiam representar um mesmo fenômeno, de modo que os seus respectivos estudos primários pudessem ser conjuntamente avaliados. O resultado da análise qualitativa destes relatos será discutido detalhadamente no próximo capítulo.

2.6.1.2. Recomendações para Pesquisa em Produtividade do Desenvolvimento de software (RQ2)

Não foi encontrada nenhuma evidência sobre o empacotamento de estudos primários selecionados nesta revisão, portanto podemos concluir que a repetição de qualquer um destes estudos primários não será facilmente realizada, se possível. Assim, a necessidade de empacotamento adequado de estudos primários ainda é pertinente no tema, merecendo atenção sob o ponto de vista metodológico.

Como em Petersen (2011), nenhum modelo de previsão geral pode, *a priori*, ser recomendado já que os estudos não exploram com clareza que os fatores afetam a produtividade do desenvolvimento de software. Enquanto os estudos forem observados individualmente, eles darão uma visão restrita sobre fenômenos que dificilmente tenderão a convergir quando observados em diferentes contextos. Por este motivo, atualmente, as organizações de software ficam restritas a identificar e testar efeitos de tais preditores quase que exclusivamente dentro de seu contexto ou de seus projetos.

Deste modo, os pesquisadores têm que se perguntar que melhorias metodológicas devem ser implementadas em seus estudos primários e secundários e que direção as suas investigações devem tomar. Neste tema de pesquisa, isto inclui discutir: (1) medidas de entrada e saída de processos de software e sua relevância para as partes interessadas em diferentes atividades ou processos de software; (2) agregar as evidências disponíveis para a disponibilização de um conhecimento ou modelo mais geral sob os fenômenos aparentemente mais relevantes, e; (3) estabelecer critérios claros para justificar a investigação de um fator, visando facilitar a comparação e generalização de resultados entre estudos.

Outro passo importante a ser tomado seria no relato de estudos primários. É preciso torná-los mais consistentes nas descrições de contexto (Petersen, 2011). Em geral, todos os estudos apresentam alguma informação de contexto, mas com diferentes ênfases. Além disso, a extração de informação relevante nesta revisão foi difícil devido à forma como muitos dos estudos foram relatados. As informações sobre contexto, análise, validade, dentre outros foram distribuídas em diferentes seções. Portanto, uma certa padronização no modo de relatar estudos sobre o tema, bem como uma pré-classificação de acordo com uma taxonomia, como a proposta por Petersen (2011), pode melhorar e tornar mais eficiente para os pesquisadores a extração dos dados em estudos primários e secundários subsequentes.

Ainda na questão do relato de estudos, Petersen (2011) destaca que para entender o retorno do investimento no que diz respeito à introdução de programas de medição com foco em produtividade do desenvolvimento de software é importante para relatar o esforço e/ou custo necessário para implementá-las na prática. O esforço e custo são importantes como critérios de decisão para as empresas. Nenhum dos 153 estudos analisados traz informações a este respeito.

Quanto à qualidade dos estudos selecionados, estes foram avaliados e ranqueados pela presença ou não de oito características baseadas em critérios de rigor experimental, conforme propostos por Petersen (2011). Estas características e que tipo de informação buscam capturar são resumidamente:

1. Contexto (***Co***): O estudo descreve minimamente o contexto onde foi executado?
2. Coleta de Dados (***DC***): Os métodos de coleta de dados foram descritos e justificados?
3. Validade (***V***): As ameaças à validade do estudo foram discutidas?
4. Construção (***Cr***): Houve uma descrição dos passos de como o modelo de medição proposto e sua estrutura foram construídas?
5. Grupo de Controle (***CG***): Há um grupo de controle ou comparação com outro modelo?
6. Análise (***A***): Há uma explicação de como a análise e interpretação dos dados coletados foi realizada?
7. Achados (***F***): Houve uma reflexão/discussão sobre a utilização dos resultados obtidos?
8. Variáveis (***Va***): As variáveis do modelo ou de medição foram definidas?

A Figura 3 mostra de modo cumulativo, ao longo das diferentes revisões executadas, como os 153 estudos selecionados se distribuem atendendo os diferentes critérios de rigor experimental.

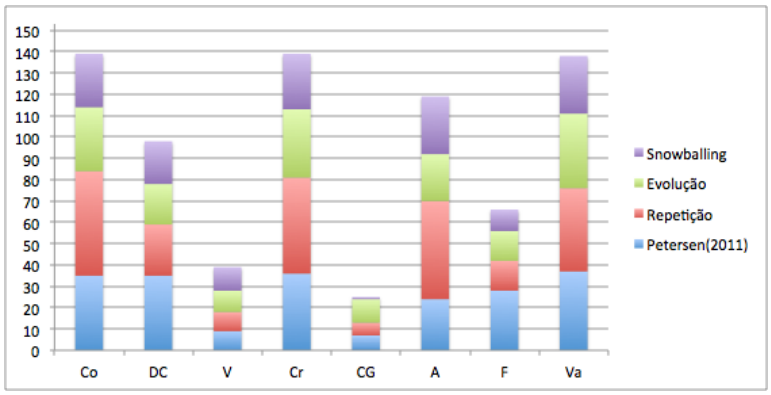

Figura 3. Distribuição dos 153 estudos de acordo com os diferentes critérios de Qualidade.

Assumindo cada conjunto de artigos selecionados como um subconjunto dos artigos selecionados em uma etapa de revisão subsequente e levando em conta o aspecto temporal de cada revisão, é possível verificar que, ao longo do tempo, os estudos executados não têm melhorado significativamente quanto ao rigor experimental (Figura 3).

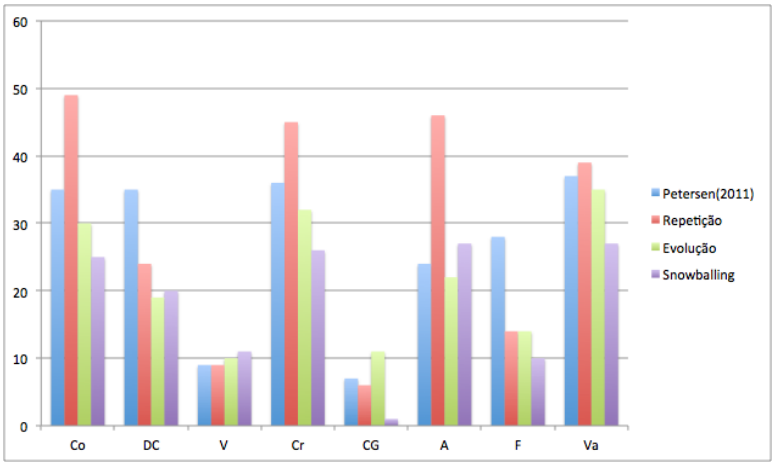

Figura 4. Distribuição de cada subconjunto de artigos de acordo com os diferentes critérios de Qualidade.

Assim como em Petersen (2011), as Figuras 3 e 4 mostram duas grandes áreas com potencial de aperfeiçoamento na pesquisa experimental sobre produtividade do desenvolvimento de software, na questão das discussões da validade e dos grupos de controle ou comparação nos estudos executados.

A respeito da validade de estudos experimentais em ES fica evidente a necessidade de melhorar a forma de capturar as limitações dos resultados, de modo que seja fácil para outros pesquisadores julgar a força da evidência.

Grupos de controle ou de comparação são importantes para sabermos qual método utilizar em quais condições. Em contextos semelhantes a presença de grupos de controle viabiliza, em muitos casos, a realização de meta-análise das evidências disponíveis a respeito de um determinado fenômeno. Pode-se atribuir a ausência de grupos de controle em parte pelo fato dos estudos em medição de produtividade serem estudos observacionais, também denominados longitudinais, onde em geral se avalia a mudança ou inserção de uma tecnologia e espera-se observar os efeitos em determinados contextos, ou seja, no desenvolvimento como um todo ou em processos de software específicos. Uma alternativa viável que vem sendo utilizada em alguns casos nas evidências encontradas para tratar esta ausência de comparação são os estudos baseados em simulação. Estes estudos demonstram estarem aptos a modelar diferentes cenários de observação com diferentes grupos de controle.

2.6.1.3. Proposições Gerais na Pesquisa em Produtividade do Desenvolvimento de Software (RQ3)

Pode-se observar que há a execução contínua de estudos primários em investigações ao longo dos anos, pois os pesquisadores anseiam demonstrar a utilidade de suas propostas. Entretanto, estes estudos são sempre executados em contextos muito restritos, o que compromete a capacidade da generalização de seus resultados.

Desta forma, mesmo que haja uma quantidade considerável de evidências e relatos sobre fatores, nenhuma proposição geral ou teoria pôde ser diretamente extraída a partir desses estudos. Isto indica que há uma ausência de consenso sobre quais fenômenos devem ser observados e melhor compreendidos, ou ainda, que não existe uma definição de questões básicas para orientar pesquisadores em estudos específicos sobre métodos de medição e predição de produtividade do desenvolvimento de software.

2.6.1.4. Distribuição de Métodos ao longo do Tempo (RQ4)

Entre os métodos preditivos, aqueles baseados em ponderação de fatores (*weighting factors*) são os mais frequentes. Métodos preditivos tiveram 49 ocorrências, enquanto os reativos 43 ocorrências em estudos primários. Entre os reativos, a medição (análise) baseada em medidas de razão simples continuam a ser o mais estudado.

A ponderação de fatores busca encontrar e explicar relações entre variáveis independentes e a dependente, através da identificação de pesos para cada uma

dessas relações. Em geral, os métodos relacionados utilizam algum método de regressão.

Em geral, a medição baseada em razão simples são modelos clássicos, utilizando a divisão de saídas por entradas, sem nenhum juízo de valor ou atribuição de pesos às variáveis medidas.

Entretanto, pode-se observar um interesse significativo no uso de Análise Envoltória de Dados (*Data Envelopment Analysis* – DEA) como uma abordagem para medição, com ocorrências em 25 estudos. A análise envoltória de dados é capaz de lidar com múltiplas entradas e saídas, que são ponderadas de modo que a produtividade seja maximizada para cada unidade de tomada de decisão (organização, projeto, indivíduo, dentre outros). Com base nas melhores produtividades encontradas num conjunto de unidades de tomada de decisão, o método determina uma fronteira de produção. Mas estes estudos, *a priori*, não permitem a comparação porque há diferentes perspectivas e medidas primárias (ou primitivas) usadas entre eles. O período entre 2001 e 2008 concentra a maioria dos estudos (oito) sobre DEA. Entre 2013 e 2016 foram encontradas cinco ocorrências de estudos primários utilizando este método, indicando um crescimento do interesse neste tipo de análise mais recente.

Com o objetivo de possibilitar a comparação entre os resultados obtidos em Petersen (2011), a seguir são destacadas as principais observações de acordo com a seguinte divisão de períodos: 1º) anteriores a 2001; 2º) entre 2001 e 2008; 3º) entre 2009 e 2013, e; 4º) entre 2013 e 2016.

Resumidamente, quanto aos principais métodos de medição e predição, o resultado da revisão de literatura alcançada nesta tese estende Petersen (2011), apresentando as seguintes informações: (1) até 2001, inclui 18 estudos, sendo três ocorrências sobre métodos baseados em ponderação de fatores, oito sobre a medição baseada em razão simples e duas sobre simulação; (2) entre 2001 e 2008, há nove referências sobre medição baseada em razão, cinco sobre ponderação de fatores e um sobre simulação; (3) entre 2009 e 2013, existem 19 estudos sobre ponderação e oito sobre simulação, e; (4) entre 2013 e 2016 foram selecionados cinco estudos sobre a ponderação de fatores, nove sobre a avaliação baseada em razão simples e cinco estudos de simulação. Isto evidencia que, a partir de 2001, há um crescente interesse em estudos de simulação.

Existem 77 estudos selecionados no período até 2008. Consequentemente, há 50 novos estudos no intervalo entre 2009 e 2013, e 26 estudos selecionados a partir entre 2013 e 2016.

2.6.1.5. Pesquisa Publicada no Tema Produtividade do Desenvolvimento de Software (RQ5)

Os métodos investigados com maior número de ocorrências são apresentados na Tabela 4, de acordo com as revisões observadas no escopo deste trabalho.

Tabela 4. Métodos mais estudados na Literatura e suas ocorrências ao longo das execuções do protocolo da revisão sistemática.

Método	Petersen (2011)	Repetição	Evolução	*Snowballing*
Ponderação de Fatores	7	18	15	5
Medição Baseada em Razão	6	6	21	9
Análise Envoltória de Dados	7	1	12	5
Simulação Baseada em Eventos	5	4	0	2
Simulação Contínua	4	1	5	1

Em Petersen (2011), as pontuações mais elevadas para o rigor experimental foram encontradas em estudos que utilizaram ponderação de fatores e análise envoltória de dados, ambos com ao menos cinco estudos alcançando uma pontuação igual ou superior a seis. Estudos baseados em simulação atingem uma pontuação entre três a seis cinco (Figura 5). A pontuação máxima que pode ser obtida por um estudo primário é oito.

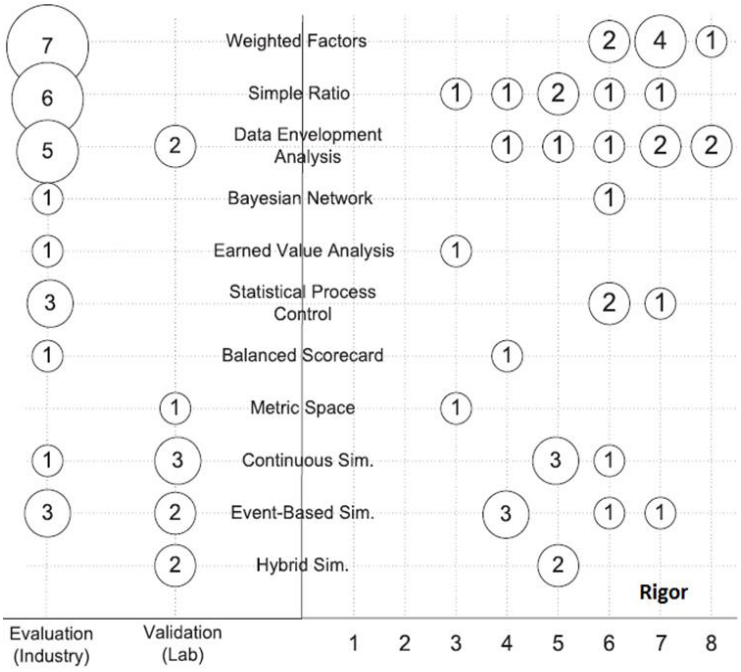

Figura 5. Obtido de Petersen (2011).

Na repetição, o número de novos estudos selecionados com pontuação de rigor acima de seis é onze, com sete novos estudos baseados em ponderação de fatores e dois novos estudos baseados em simulação. Nenhum novo estudo alcança pontuação máxima. A Figura 6 destaca as mudanças ocasionadas pela inserção dos estudos na repetição em relação a (Petersen, 2011).

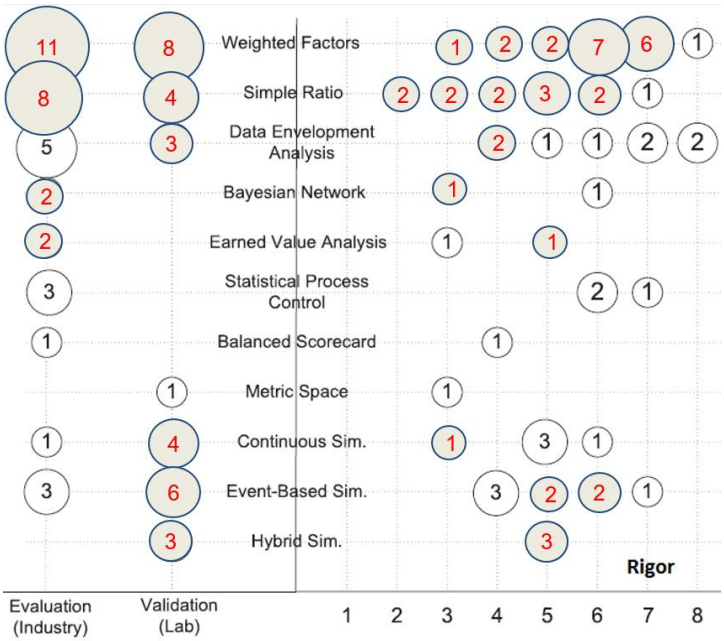

Figura 6. Mudanças com a repetição, adaptado de Petersen (2011).

Na evolução, o número de novos estudos com pontuação de rigor acima de seis é sete, onde os estudos de ponderação de fatores e análise baseada em razão simples agrupam três estudos cada. Somente um estudo alcança pontuação máxima no protocolo evoluído. Somente foram acrescentados um estudo utilizando Redes Bayesianas e outro em Controle Estatístico do Processo. A Figura 7 destaca as mudanças ocasionadas pela inserção dos estudos selecionados na evolução em relação a Petersen (2011).

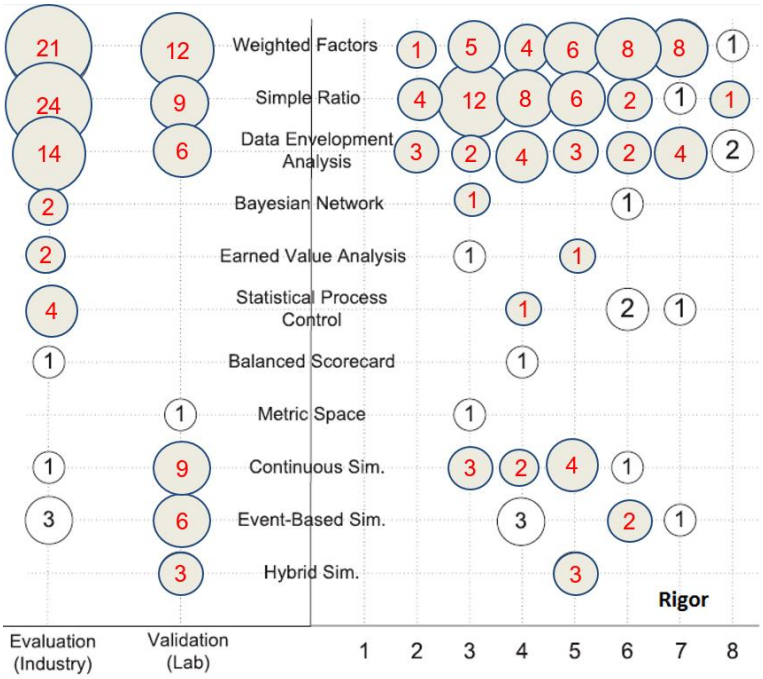

Figura 7. Mudanças da evolução, adaptado de Petersen (2011).

Observando os artigos selecionados no *snowballing* (Figura 8), a mudança mais significativa do cenário diz respeito ao número de novos estudos com pontuação de rigor acima de seis, com a adição de três novos trabalhos.

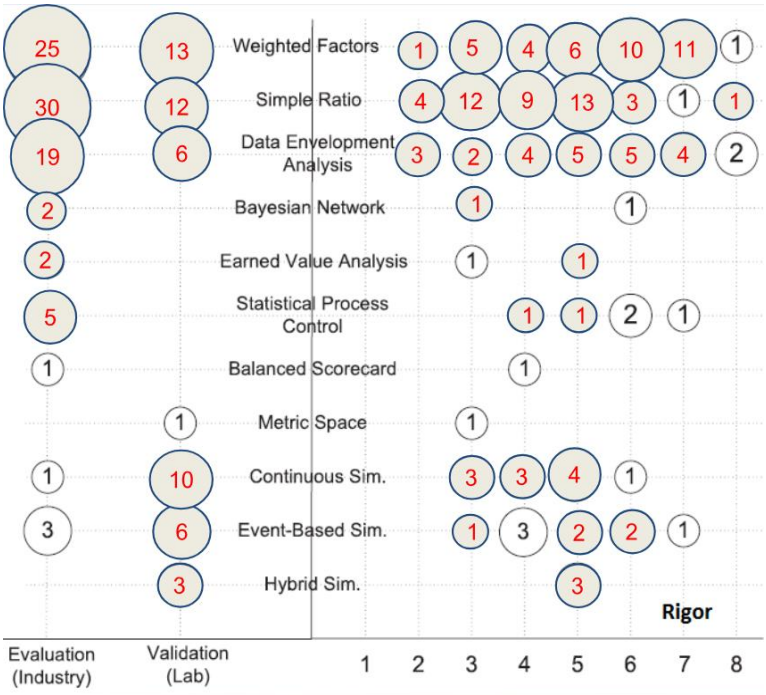

Figura 8. Mudanças pelo *snowballing*, adaptado de Petersen (2011).

Observando os resultados da aplicação de todo o protocolo, novos métodos foram encontrados em oito novos estudos e serão discutidos na próxima seção. Seus impactos na pesquisa publicada podem ser observados na Figura 9. Isto mostra que todos estes métodos ainda foram pouco explorados ao longo dos anos.

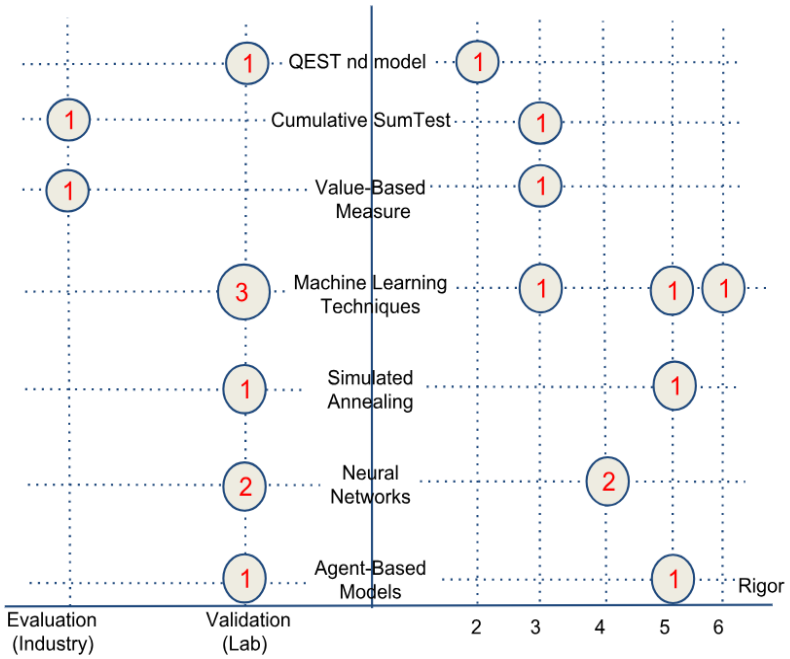

Figura 9. Estudos apresentando novos métodos.

De qualquer forma, as informações apresentadas anteriormente indicam um aumento na investigação experimental sobre medição e predição de produtividade do desenvolvimento de software nos últimos sete anos. As classificações dos novos estudos e pontuações quanto ao rigor experimental estão disponíveis em Chapetta (2018).

2.6.2. Caracterização de Métodos de Medição e Predição

As definições iniciais e informações de métodos já encontrados na literatura técnica podem ser obtidos em Petersen (2011).

No resultado final desta investigação foram identificados sete novos métodos para medir e prever a produtividade software: Recozimento Simulado (*Simulated Annealing*), Medição Baseada em Valor, Modelo baseado em Agentes, Técnicas de Aprendizado de Máquina, Redes Neurais, *QEST nD Model* e *Cumulative Sum Test*. Os novos métodos são sucintamente descritos a seguir:

- Medição baseada em Valor: assume que a métrica deve tentar medir as variações do valor das partes interessadas relevantes, e assume que se uma organização for bem-sucedida na captura deste valor para suas respectivas partes interessadas, em tese, seria capaz de perceber e reagir a conflitos, sugerindo que o valor pode ser mais eficientemente entregue. Nesta revisão, apenas o trabalho de Aquino Júnior & Meira (2009) emprega a medição baseada em valor, apresentando uma prova de conceito e propõe uma metodologia para implementar e executar um processo de medição que, teoricamente, poderia apoiar os profissionais a definir uma medida de produtividade com base no valor percebido a partir de partes interessadas pré-definidas. Os valores atribuídos nada mais são do que pesos, mas neste método estes pesos são geralmente atribuídos por alguns profissionais responsáveis por capturar, diferenciar e representar as perspectivas das partes interessadas.
- Técnicas de Aprendizado de Máquina: engloba um grande número de métodos. No entanto, um dos estudos selecionados nesta categoria apresenta uma combinação de dois destes métodos (regras de associação e árvores de classificação e regressão) para prever o aumento ou diminuição de produtividade. Tal fato nos motivou a caracterizar este estudo através de uma classificação mais ampla. Por exemplo, em Bibi *et al.* 2008, estes métodos são combinados para modelar relações desconhecidas e fornecer uma estimativa da

produtividade. Assim, o método proposto pode revelar a maneira em que determinados atributos e fatos podem aumentar ou diminuir a produtividade do desenvolvimento de software. Já Sharpe & Cangussu (2005) usam reconhecimento de padrão para identificar faixas de produtividade e observar os integrantes de projetos de software.

- Recozimento Simulado (*Simulated Annealing*): é um algoritmo baseado na analogia entre um processo de recozimento de sólidos e resolução de grandes problemas de otimização combinatória. O método funciona emulando o processo físico em que um sólido é lentamente resfriado de modo que, eventualmente, a sua estrutura seja congelada, que é a configuração mínima de energia. Então, Kang, Jung, & Bae (2011) empregaram esse método para propor uma solução para a alocação de recursos humanos em projetos de software a nível individual, proporcionando uma orientação de acordo com vários fatores para predizer a produtividade dos desenvolvedores de software com base em COCOMO II (Boehm, Madachy, & Steece, 2000).
- Redes Neurais: são técnicas de computação e processamento de sinal inspirado em uma rede de neurônios biológicos. Lopez-Martin, Chavoya, & Meda-Campaña (2014) modelaram e treinaram uma rede neural com um conjunto de dados de 140 projetos de software desenvolvidos entre os anos de 2005 e 2008 com as práticas com base no *Personal Software Process* (PSP). Em seguida, o estudo compara a precisão obtida pela rede neural com um método usando lógica *fuzzy* e modelos de regressão estatística para o mesmo conjunto de projetos.
- Modelos baseados em Agentes: Celik *et al*. (2011) definem um *framework* baseado em um método de emulação para propor uma solução para a atribuição da força de trabalho, a aplicação de um modelo baseado em agentes para selecionar as melhores alocações de força de trabalho possíveis e estimar a produtividade, bem como estimar o desempenho organizacional a longo prazo.
- *QEST nD model*: Este modelo descreve o processo global de medição como a combinação de qualquer dimensão considerada, calculado como a soma ponderada das métricas aplicadas. De acordo com Ardagna *et al*. (2010), o método proposto é dissociado de processos de desenvolvimento específicos, permitindo uma análise multi-processo, e multi-projeto.

- *Cumulative Sum Test*: consiste de um teste sequencial de alterações, acumulando evidências à medida que cada nova amostra é retirada. O método tem sido utilizado para a detecção de alterações em variáveis estocásticas, tais como o valor médio e a variância de um processo Gaussiano. Ramil & Lehman (2001) realizam este teste que permitiu examinar se a produtividade em um projeto mudava ao longo de um período de onze anos, usando dados empíricos de módulos de software.

2.7. Ameaças à Validade

A principal ameaça à validade das revisões executadas para definição desta proposta é o viés dos pesquisadores, que foi minimizado através de *peer-reviews* em que os pesquisadores avaliaram as questões de pesquisa, os critérios utilizados, a extração e análise das revisões executadas. Além disso, os resultados de Petersen (2011) foram utilizados como artigos de controle, para evitar que as alterações propostas no protocolo de pesquisa pudessem distorcer resultados obtidos anteriormente. O risco de exclusão de artigos relevantes foi reduzido por ser inclusivo, ou seja, os pesquisadores incluíram artigos para revisão detalhada em caso de dúvida de sua pertinência. Além disso, mesmo estudos com classificação de baixa qualidade foram incluídos, como no protocolo original.

Quanto à execução do protocolo, Petersen (2011) afirma que os critérios utilizados para avaliar o rigor experimental dos estudos reduzem o viés da revisão devido à sua simplicidade. Realmente, estes critérios são simples e servem de orientação para a extração de dados de estudos primários, mas o poder de discriminação das evidências fica visivelmente comprometido. Outro ponto a destacar é que a string de busca executada nesta pesquisa não inclui termos relativos a métodos de pesquisa como *survey* e *action research*. Alguns *surveys* foram selecionados e incluídos desde o trabalho do protocolo original, em 2011. Entretanto, não podemos afirmar que não há estudos que precisariam ser incluídos caso o termo *survey* fosse incluído na string de busca. Por outro lado, o termo *action research* também poderia ser utilizado para selecionar estudos, mesmo considerando que a utilização deste método em ES é recente. O impacto da inclusão destes termos e de outros métodos de pesquisa em revisões subsequentes ainda precisa ser avaliado.

É necessário ressaltar que as revisões executadas para esta tese estão limitadas a 2016. Há atualmente uma lacuna de dois anos entre os resultados das revisões e os resultados apresentados nesta tese. Como se trata de um estudo que,

entre outros propósitos, visa atualizar e fornecer o status do conhecimento no tema, este ponto também é considerado uma limitação nesta pesquisa. Este é somente o primeiro passo da metodologia de pesquisa proposta e, como será visto a seguir, pela quantidade de material que precisou ser manipulado, não é viável manter esta lacuna a menor possível. Entretanto, os resultados não se tornam menos acionáveis no curto prazo, pois estes não deixam de ser válidos para outros pesquisadores devido a esta lacuna temporal. De qualquer forma, a disponibilização do protocolo permite a sua atualização e evolução em estudos futuros.

Pesquisas subsequentes têm de levar em conta a possibilidade de estudos relevantes não terem sido ainda selecionados. A viabilidade de repetir um protocolo de revisão sistemática de modo incremental ficou demonstrada ao longo deste capítulo, de modo que não há motivos para acreditar que sua repetição deixará de ser viável em um futuro próximo.

2.8. Questões em Aberto

Com base nos achados das revisões executadas, podemos destacar como questões em aberto as listadas a seguir:

- Como em Petersen (2011), nenhum conjunto ou modelo de proposições teóricas gerais, ou seja, nenhum conjunto de proposições (hipóteses, leis e teorias) comuns derivadas de questões básicas, em produtividade do desenvolvimento de software foi identificado.
- Os resultados das revisões executadas podem sinalizar que alguns métodos e fenômenos são “bem” conhecidos, e apontam aqueles que precisariam ser mais ou melhor observados e com que rigor sob o ponto de vista experimental. Por exemplo: Por que métodos como Controle Estatístico de Processos e Redes Bayesianas foram tão pouco explorados?
- As evidências mostram que, apesar de haver diferentes níveis de observação (organização, projetos, indivíduos, dentre outros), em geral os estudos executados consideram a produtividade do ponto de vista de todo o processo de desenvolvimento de software. Não foram encontrados estudos observando a produtividade de processos de software de uma forma específica. O que reflete uma visão de observar o processo de desenvolvimento como uma linha de produção ou manufatura. Assim, os estudos não têm como, por exemplo, identificar gargalos entre processos de software específicos (gerência de

configuração, VV&T, gerência de requisitos, e outros) e como estes impactam a produtividade global do processo de desenvolvimento.

- Por fim, nos estudos primários selecionados não foram encontradas preocupações com métodos de medição e predição com o propósito de capturar a componente de impacto de uma mudança tecnológica ou organizacional, bem como uma preocupação em avaliar o impacto de tais mudanças sob um ponto de vista temporal, ou seja, ao longo de um período.

De todas estas questões somente a primeira será tratada nesta proposta de tese. A segunda já poderia ser endereçada com os resultados obtidos, destacando somente a necessidade de avaliar a atualização do protocolo para contemplar artigos posteriores a 2016. Os outros dois pontos poderão ser facilmente endereçados a partir dos resultados ao final da tese proposta.

2.9. Conclusão

Controlar e melhorar a produtividade do desenvolvimento de software é reconhecidamente decisivo para projetos de software bem-sucedidos e organizações competitivas (Cataldo & Herbsleb, 2013; de O. Melo *et al*., 2013; Colomo-Palacios *et al.*, 2014; Meyer *et al.*, 2014; Hernández-López *et al.*, 2015; Ramírez-Mora & Oktaba, 2017). No entanto, algumas questões elementares (consenso conceitual e uma agenda de investigação comum) devem ser discutidas pela comunidade de ES para permitir o aumento no ritmo de evolução do conhecimento face aos desafios de projetos de software contemporâneos.

Este capítulo apresentou uma abordagem *bottom-up* para identificar métodos de medição e predição além de relatos de fatores influenciando a produtividade do desenvolvimento de software através de *quasi*-revisões sistemáticas, replicando e evoluindo um protocolo encontrado na literatura e contribuindo para tornar explícito um maior número de estudos não previamente identificados. O protocolo de pesquisa evoluído foi capaz de selecionar estudos mesmo no período contemplado pelo protocolo original, devido, provavelmente, à indexação e atualização de material nas bases de periódicos.

Aparentemente, os pesquisadores têm investido na investigação de métodos baseados em ponderação de fatores e medições baseadas em razão por questão de conveniência, ajustando os dados de projetos de software disponíveis para testar seus modelos e encontrar correlações sem definir uma agenda de investigação comum. Entretanto, como pesquisadores, temos de nos perguntar se tal fato não impede a evolução do conhecimento sobre preditores da produtividade e o aumento da força de evidências por falta de possibilidade de agregação, acarretando na necessidade contínua de realizar estudos tão singulares que seus resultados nunca podem ser utilizados em diferentes projetos e organizações.

No próximo capítulo serão tratados o uso dos 153 artigos selecionados nas revisões executadas para a identificação de fatores de influência na produtividade de software.

Capítulo 3: Fatores que Afetam a Produtividade do Desenvolvimento de Software

Este capítulo apresenta a análise qualitativa para identificar e descrever conceitos mais estudados na literatura técnica sobre produtividade em desenvolvimento de software.

3.1. Considerações Iniciais

Conforme discutido anteriormente, na literatura técnica foram encontrados relatos de fatores que experimentalmente demonstraram ter algum efeito na produtividade do desenvolvimento de software, ou seja, fatores com níveis de confiança explicitamente expressos ou cujos estudos demonstraram diferença significativa entre tratamentos. Com base em uma investigação inicial, foram identificados relatos de 83 diferentes fatores afetando a produtividade do desenvolvimento de software.

Neste capítulo será abordada a análise qualitativa realizada buscando, nas evidências, identificar conceitos semelhantes com denominações diferentes ou aqueles que pudessem ser agregados em um construto mais geral, capaz de capturar uma perspectiva mais ampla a respeito das relações envolvidas.

Assim, serão apresentados os passos utilizados para a obtenção dos fatores que serviram para a construção do modelo proposto nesta tese. Para tal, tomou-se por base uma das vertentes da Teoria Fundamentada em Dados (*Grounded Theory* - GT) (Strauss & Corbin, 2015), a Clássica.

A GT tem sido utilizada para a análise qualitativa de estudos primários com sucesso em diferentes temas da ES (Barke & Prechelt, 2018; Gralha, Damian, Wasserman, Goulão, & Araújo, 2018; Holmes, Allen, & Craig, 2018; Lindman & Hammouda, 2018). Por exemplo, Rahman *et al.* (2018) utilizam o método com o intuito de auxiliar na identificação de métricas a serem utilizadas em atividades de inspeção no contexto de “Infraestrutura como Serviço” (IaaS).

Assim, as seções seguintes destacam os principais elementos e nuances deste método, bem como os resultados encontrados durante a análise para identificação dos fatores.

3.2. Teoria Fundamentada em Dados

O método da Teoria Fundamentada em Dados ou Grounded Theory (GT) foi desenvolvido como uma alternativa à tradição hipotético-dedutiva da pesquisa qualitativa da época, ou seja, propunha o desenvolvimento de teorias a partir de dados obtidos por meio da pesquisa, e não a dedução de hipóteses por meio de teorias existentes (J. L. G. dos Santos *et al.*, 2016). A GT baseia-se na tendência natural de pesquisadores em teorizar e na ideia de haver um padrão capaz de descrever o comportamento observado (Glaser, 1978; Kenny & Fourie, 2015; Strauss & Corbin, 2015).

Princípios fundamentais do método da GT incluem: (1) minimizar ideias preconcebidas sobre o problema de pesquisa e os dados, (2) usar coleta e análise de dados simultâneos para informar uns aos outros, (3) permanecer aberto a variadas explicações e/ou entendimentos dos dados e (4) focar a análise de dados para construir teorias de médio alcance (Charmaz, 2008).

Tais princípios têm o objetivo de traduzir e identificar conceitos, características e relacionamentos para extrair uma representação dos elementos de um assunto ou questão, a partir de fontes de dados (relatos, entrevistas, textos técnicos, arquivos de áudio, etc.). Como exemplo, França (2015) usa a GT para identificar diretrizes para planejamento de estudos baseados em simulação, além da identificação e mitigação de potenciais ameaças à validade relacionadas a esse tipo de estudo.

Há outras opções para capturar conceitos. Santos (2015) propõe mecanismos onde a extração e tradução de conceitos e relacionamentos são executadas ao longo de um método de síntese, mas de forma pouco ampla. Seu método foca em aspectos estruturais das evidências que nos levam a concentrar nos elementos causais e contextuais, sem levar em consideração a ideia de extração e rotulagem (catálogo) das informações para revelar os principais temas ou o registro de linhas de argumentação na identificação e agrupamento dos conceitos. Isso significa que não é possível identificar com os mecanismos propostos por Santos (2015), por exemplo, quais características tem um conceito ou que conceitos englobam outros conceitos. Então se buscamos identificar diretrizes e os tipos de ameaça à validade de um tipo de estudo, por exemplo, os mecanismos de Santos (2015) não nos auxiliarão. Deste modo, França (2015) nunca poderia chegar aos seus resultados utilizando a proposta de Santos (2015).

Outro ponto que merece destaque é que Santos (2015) nos restringe a realizar a síntese de um elemento causal, um fator, por vez. Tal fato torna razoável adotar

seus mecanismos de codificação, uma vez que em ES na maior parte das sínteses, assim como numa possível meta-análise, se busca qualificar/quantificar o possível efeito do uso/introdução de uma tecnologia/método/ferramenta em uma ou mais características de um processo/produto de software. Em outras palavras, em geral, quando tomamos a decisão de sintetizar evidências com o método proposto por Santos (2015), de alguma forma foi decidido *a priori*: (1) qual o elemento causal capaz de combiná-los, e; (2) quais estudos serão combinados. Tais premissas conduzem à uma situação em que a quantidade de estudos e conceitos a manusear é restrita e, em tese, relativamente fáceis de gerenciar.

No entanto, isto não se aplica ao caso dos fatores identificados pelas revisões executadas no âmbito desta tese. O número de relatos é grande e a *priori* não temos certeza nem podemos assumir que tais relatos são unívocos, requerendo uma análise mais ampla e profunda das informações contidas nos estudos. Assim, avaliar o potencial de combinação de estudos e conceitos necessitou mecanismos elaborados de controle e registro das informações.

Por este motivo, este trabalho adota a GT, que ao longo dos anos tem sido explorada sob algumas vertentes de trabalho diferenciadas principalmente sob a perspectiva dos procedimentos metodológicos a serem possivelmente adotados. As vertentes da GT são: a Clássica, a Straussiana e a Construtivista (Santos *et al.*, 2018).

Apesar das divergências, estas três vertentes compartilham dos mesmos elementos inerentes ao método, que são: (1) amostragem teórica; (2) análise comparativa constante de dados; (3) elaboração de memorandos (memos); e (4) diferenças entre teoria substantiva e teoria formal. Resumidamente, de acordo com Santos *et al*. (2018), os princípios são:

- A amostragem teórica: que se refere à identificação e coleta de dados de participantes e/ou fontes de dados consideradas relevantes para responder à questão de pesquisa. Ciclicamente, à medida que os dados coletados são analisados, novos assuntos ou dados podem ser usados de acordo com a necessidade específica de aprofundar ou preencher lacunas do conhecimento. Os dados são coletados e analisados concomitantemente até atingir saturação teórica, ou seja, até que nenhum novo conceito emerja a partir de sua análise.
- A análise comparativa constante: onde, primeiro, os dados coletados são analisados incidente por incidente, a fim de gerar códigos conceituais. Esses códigos são agrupados, denotando conceitos de

nível superior, também às vezes denominados de categorias. Através de comparações constantes, os códigos e categorias são continuamente comparados entre si.

- A elaboração de memorandos: à medida que os conceitos (códigos) começam a emergir estes são registrados junto às reflexões que os trouxeram à tona e são todos registrados na forma de memorandos, visando capturar os códigos e desenvolvimento das ideias que os originaram.
- A diferença entre teoria substantiva e teoria formal: uma teoria substantiva é aquela que só se aplica ao contexto investigado. Por sua vez, uma teoria formal requer um refinamento envolvendo a geração de conceitos mais abstratos que possam ser aplicados de forma generalizada a uma realidade mais ampla.

Quanto às diferenças entre as vertentes, a literatura técnica aponta que as diferenças entre elas surgem essencialmente de três pontos (Kenny & Fourie, 2015): (1) no procedimento de codificação; (2) posições filosóficas, e; (3) no uso da literatura técnica.

Na vertente Clássica, o procedimento de codificação é marcado pelo princípio do surgimento de uma teoria como consequência direta do conteúdo dos dados. Do ponto de vista filosófico, a perspectiva clássica propõe que a pesquisa implica simplesmente em “um processo de revelar ou descobrir fenômenos pré-existentes e a relação entre eles” (Madill, Jordan, & Shirley, 2000), adotando uma visão pré-positivista (Santos *et al.*, 2018). Sob a perspectiva Clássica, o uso da literatura técnica deve ser empregado ao final da codificação teórica (*theoretical coding*), como objeto para a comparação e mapeamento (Giske & Artinian, 2007) com os resultados obtidos ao final.

Na vertente Straussiana, o procedimento de codificação é composto por uma estrutura altamente sistemática para criar (descrever, ao invés de simplesmente descobrir) uma teoria rigorosamente correspondente aos dados. Composta de quatro estágios de codificação, cada estágio é rigorosamente qualificado de modo a delimitar as regiões limítrofes entre as sucessivas fases, que são: codificação aberta (*open coding*), codificação axial (*axial coding*), codificação seletiva (*selective coding*) e matriz condicional (*conditional matrix*). O detalhamento de cada uma destas fases está fora do escopo deste trabalho, e mais detalhes podem ser obtidos em Strauss & Corbin (2015). Mas, resumidamente, a ideia geral é que além de identificar e definir os conceitos, é preciso capturar propriedades e dimensões de cada um deles, bem como

definir através de paradigmas a relação entre conceitos e subconceitos. Assim, do ponto de vista filosófico, a perspectiva da GT Straussiana assume uma posição positivista, caracterizado aqui pela imposição da estrutura mais rígida de regras para a condução dos procedimentos de codificação. Do ponto de vista de uso da literatura técnica, aqui ela é utilizada apropriadamente ao longo de todos os estágios da codificação nesta perspectiva.

A partir de 2000, surge a vertente construtivista do método, que define diretrizes de codificação altamente adaptáveis que endossam a liberdade interpretativa dos dados (Charmaz, 2008), permitindo uma certa imprecisão subjetiva quanto à correspondência aos dados. Sob esta perspectiva, há somente dois procedimentos iterativos durante a codificação: (1) codificação aberta (*open coding*), e; (2) codificação "refocada" (*refocused coding*). Estes procedimentos orientam na identificação de conceitos recorrentes, elevados a categorias teóricas provisórias, com foco na elaboração dos memorandos. Deste modo, os códigos e categorias são interpretados a fim de destacar condições, consequências, dentre outras características. Um estudo de análise de GT adotando uma perspectiva construtivista tipicamente tende a encontrar resultados com a compreensão interpretativa do pesquisador (Hallberg, 2006). Do ponto de vista filosófico, a denominação desta perspectiva se dá por admitir possíveis interações e ambiguidades devido a imersão e/ou interação entre o objeto de estudo e o pesquisador. Quanto ao uso da literatura técnica, esta perspectiva endossa seu uso ao longo de toda a análise, mas vai além ao recomendar que ao final seja feita uma compilação da literatura técnica específica.

3.3. Definição e Execução da Análise Qualitativa

Nesta etapa do método de pesquisa, o objetivo é realizar uma análise qualitativa com base nos estudos selecionados na revisão da literatura. Tem como objetivo comparar iterativamente cada ocorrência de relato dos fatores que afetam a produtividade do desenvolvimento e identificá-las com conceitos já identificados ou não, até que não surja nenhum novo conceito ou outro intercambiável com potencial de definição mais geral (teórico), quando observado um conjunto pré-estabelecido de evidências.

Diferentemente da codificação segundo a vertente Straussiana, que adota uma estrutura mais rígida para descrever o contexto e as relações entre conceitos, e da Construtivista, que busca dar uma interpretação às observações, neste momento, buscou-se somente adotar uma posição mais alinhada à captura das informações existentes nos estudos.

Assim, a análise executada para este trabalho atenta para a integração conceitual de conceitos explícitos e relacionados para identificar conceitos mais gerais, tentando tornar visíveis os construtos latentes como uma base para novas observações, o que está mais alinhado ao conceito da codificação teórica da GT Clássica (Hernandez, 2009; Thornberg & Charmaz, 2014).

A codificação teórica (Clássica) fratura os dados em códigos (substantivos), e compara-os uns com os outros, buscando por semelhanças e diferenças, até que haja a saturação teórica (Hernandez, 2009). Quando os pesquisadores estão realizando a identificação de códigos teóricos, eles devem procurar por uma teoria substantiva integrada e explicativa (isto é, conceitos que explicam os fenômenos observados). Códigos teóricos conceitualizam como os códigos substantivos podem relacionar-se entre si como hipóteses a serem integradas na teoria. Em suma, os códigos substantivos fracionam os dados, enquanto os códigos teóricos recuperam a "história fragmentada" em uma teoria organizada como um todo (Glaser, 1978).

Sem códigos substantivos, os códigos teóricos são abstrações vazias (Glaser, 1978). Desta forma, a identificação de códigos substantivos e teóricos não são processos isolados ou desconectados. Ambos os tipos de identificação coincidem, até certo ponto, mas o pesquisador "focalizará relativamente mais na codificação substantiva ao descobrir códigos dentro dos dados, e mais na codificação teórica quando teoricamente ordenando e integrando seus memorandos" (Hernandez, 2009).

Buscar alinhar o estudo qualitativo realizado nesta tese a uma das perspectivas apresentadas quanto à questão do uso da literatura técnica para apoiar as observações não faz diferença, ou mesmo sentido, já que a própria literatura foi utilizada como objeto de estudo. Deste modo, esta não será usada para comparação ou consulta ao longo do processo de codificação, conforme preconizado na perspectiva Clássica. Tampouco será feita uma compilação interpretativa dela ao final da análise, como na perspectiva Construtivista.

Entretanto, para a elaboração dos memorandos, este trabalho desenvolveu um modelo baseado em Chidamber & Kemerer (1991). O modelo proposto ajuda na captura de conceitos (fatores) através de informações relativas à definição, pontos de vista e raciocínio (lógica) em que tais conceitos podem teoricamente combinar estudos primários, bem como mostra como diferentes medidas têm sido empregadas para operacionalizá-los nos estudos primários selecionados.

Por exemplo, tamanho de software baseado em linhas de código (LOC-based Size) e tamanho funcional (FP-based Size) foram considerados códigos

independentes e relatados separadamente em seus respectivos memorandos. A interpretação aqui decorre que, neste momento, é necessário considerar que ambos capturam diferentes aspectos do software, mesmo que estes tenham uma denominação comum, neste caso, a ideia de tamanho. Se fossem a mesma característica ou intercambiáveis entre si, poderíamos encontrar, com algum erro, uma constante *Ki* que representa a razão LOC/PF para uma linguagem de programação *i*, por organização, etc. Embora haja alguns resultados neste sentido (Rubin, 1993; Port & MacArthur,1999;), há outras evidências mostrando que mesmo dentro de projetos de uma organização nem sempre isto poderia ser considerado verdade (Kemerer, 1987, 1993; Low & Jeffery, 1990).

Outra analogia que pode ser feita é que projetos de software querem entregar uma certa quantidade de pontos de função a seus clientes, mas entregá-los com a melhor solução na menor quantidade possível de linhas de código, pois há um esforço inerente à concepção da solução durante a codificação que dificilmente, desta forma, será rentabilizada. Na prática, não ficaríamos surpresos se diferentes desenvolvedores implementassem uma mesma funcionalidade com diferentes quantidades de LOC, utilizando uma mesma linguagem de programação. Aliás, esperaríamos que, quanto mais complexa a funcionalidade a ser implementada, maior seja a disparidade entre as quantidades de LOC produzidas neste cenário. Deste modo, estes códigos foram mantidos separados para ser possível verificar mais adiante se são equivalentes ou intercambiáveis. Se considerarmos somente o paradoxo apresentado no parágrafo anterior, esperaríamos que não.

Um exemplo resumindo o preenchimento do modelo de memorando para o conceito Tamanho Funcional (Baseado em Pontos de Função - *Functional (FP-based) Size*) pode ser visto na Figura 10. Todos os memorandos estão disponíveis na internet em http://goo.gl/y8ueKL, bem como no anexo VI desta tese.

Variable: Functional (FP-based) Size

Definition:

It represents the idea of observing the software functional size measured through counting function-points (FP).

Theoretical Basis:

Some researchers investigate whether there is some relationship between software functional size and software productivity. They have tried to observe this relationship by using FP-based measures, assuming that this kind of measure may theoretically represent the functional size of software.

Viewpoints:

Increasing the software functional size may influence software productivity over time.

Measures found in studies:

Name	Specification
FP-based size	Ratio scale, measured as software functional size measured in function-points.

Articles:

References
FOULDS, Les R.; QUADDUS, Mohammed; WEST, Martin. Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. In: **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. IEEE, 2007. p. 724-731.
MORASCA, Sandro; RUSSO, Giuliano. An empirical study of software productivity. In: **Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International**. IEEE, 2001. p. 317-322
TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.
TSUNODA, Masateru; ONO, Kenichi. Pitfalls of analyzing a cross-company dataset of software maintenance and support. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.
WANG, Hao; WANG, Haiqing; ZHANG, Hefei. Software productivity analysis with CSBSG data set. In: **Computer Science and Software Engineering, 2008 International Conference on**. IEEE, 2008. p. 587-593.

Figura 10. Exemplo do modelo de memorando e seu preenchimento.

Do ponto de vista metodológico, pode-se dizer que o modelo de memorando proposto nesta tese incorpora, em parte, elementos da GT Construtivista. Não para efetivamente interpretar os fenômenos que estão sendo observados, mas como uma interpretação sobre qual ponto de vista os estudos podem ser combinados. Entretanto, a efetiva interpretação dos resultados e a compilação da literatura pertinente ficaram a cargo das sínteses das evidências, que serão devidamente apontadas e discutidas nos próximos capítulos.

Pelos motivos acima expostos, o estudo qualitativo realizado nesta tese foi considerado predominantemente relacionado à vertente da GT Clássica. Entretanto,

devemos ter em mente que a abordagem para alcançarmos o *framework* teórico utilizada na metodologia de pesquisa desta tese contempla, primeiro, identificar e extrair os códigos (conceitos) utilizando a GT Clássica e, em seguida, realizar um procedimento de síntese para obter as estruturas teóricas tomando por base esses códigos. A ilustração desse cenário pode ser vista na Figura 11.

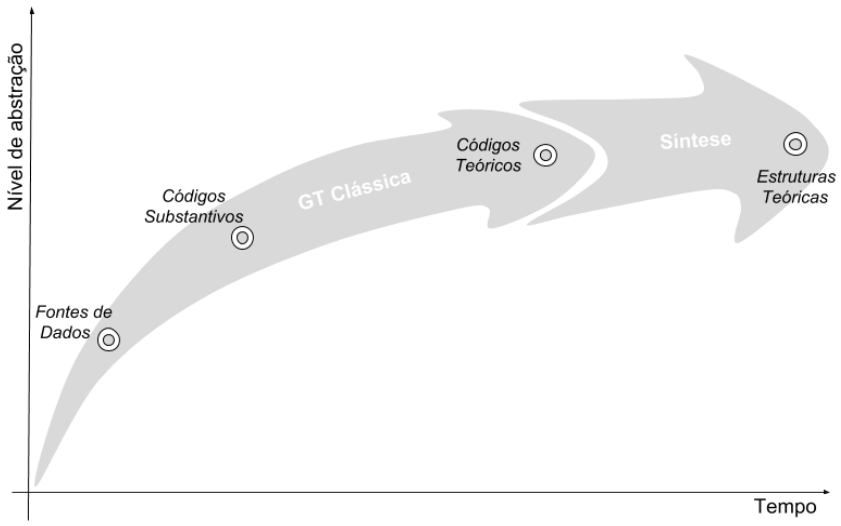

Figura 11. Análise Qualitativa e Síntese (adaptado de Santos, 2015).

Durante a identificação dos códigos substantivos, os artigos selecionados foram revisitados em busca dos fatores que experimentalmente foram avaliados em estudos primários. Além de trechos de texto contendo a definição de conceitos nos artigos, também foram identificadas as medidas usadas para operacionalizar tais fatores em seus respectivos estudos. As informações foram devidamente registradas nos memorandos. Em um processo iterativo, novos códigos substantivos foram identificados a medida que não se adequavam a nenhum outro identificado anteriormente. A Figura 12 ilustra a identificação de dois códigos substantivos em Otero *et al.* (2012).

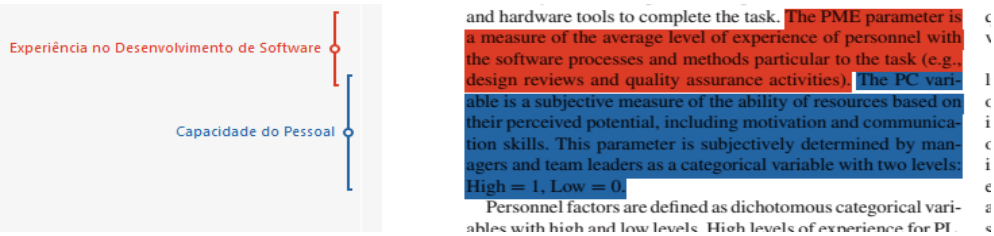

Figura 12. Exemplos de códigos substantivos em Otero *et al.* (2012).

Na identificação de códigos teóricos, uma análise das definições e da operacionalização dos fatores permitia verificar se existia um conceito mais geral que pudesse agrupar estudos em um conceito mais geral. Por exemplo, na Figura 13

foram identificados dois códigos distintos, pois diferem pela definição e pela sua operacionalização. Na parte superior da figura o código *Capital Social Relacional* (Hsu & Hung, 2013), na inferior o código *Envolvimento de Usuários* (Ramasubbu *et al.*, 2015).

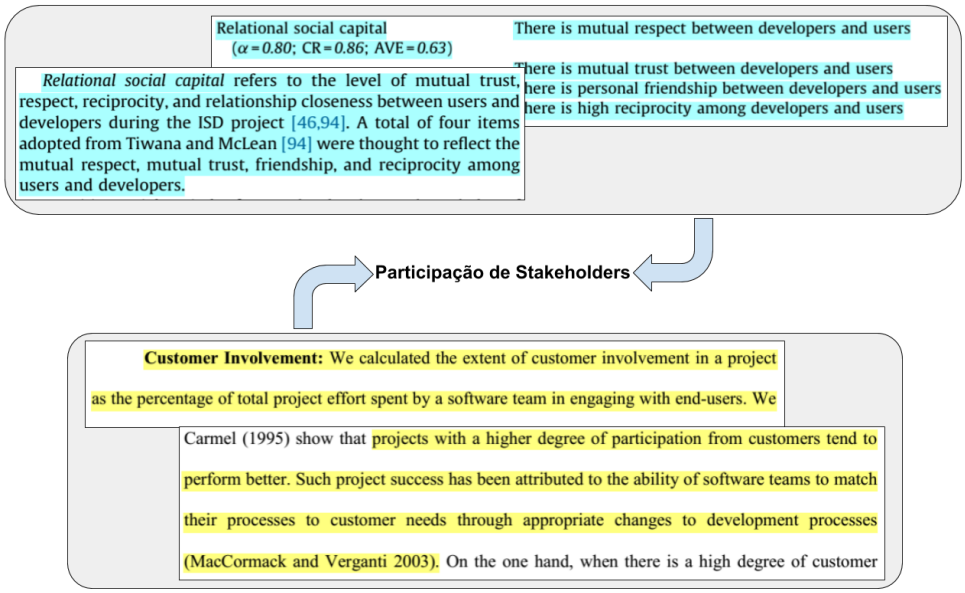

Figura 13. Identificando um código teórico a partir de Hsu & Hung (2013) e Ramasubbu *et al.* (2015).

A partir desses códigos substantivos se chegou a um conceito mais geral que se considerou ser capaz de combiná-los, Participação de Stakeholders. Assim, mesmo que não representem exatamente um mesmo conceito, são de alguma forma combináveis, pois foi interpretado haver um conceito mais amplo, que pode representar os dois (uma generalização). Isso é equivalente à criação de uma categoria na perspectiva Straussiana, mas aqui o foco é somente nos fenômenos descritos.

Com um código teórico identificado, o seu registro é feito em seu respectivo memorando. Isto é realizado iterativamente até que não seja possível identificar um novo código teórico. Ao final, todos os códigos com três ou mais estudos foram considerados suficientemente gerais e, assim, considerados teóricos.

3.4. Resultados da Análise Qualitativa

A partir de 83 relatos de fatores encontrados nos diferentes estudos primários, já identificados como códigos substantivos, foram encontrados 25 fatores definidos como códigos teóricos. Estes códigos emergiram da análise estando estritamente relacionados a hipóteses empiricamente avaliadas nos estudos selecionados.

Na Tabela 5 podem ser vistos os fatores a serem levados em conta para as etapas restantes do método de pesquisa deste trabalho, definidos a partir de seus respectivos códigos teóricos.

Tabela 5. Fatores Encontrados definidos a partir dos Códigos Teóricos

Código Teórico	*Definição*	*# de estudos*	*Estudos*
Tamanho baseado em LOC	Representa a observação do tamanho do software medido através da contagem de linhas de código fonte.	9	(D. R. Jeffery, 1987); (Maxwell et al., 1996); (Banker & Slaughter, 1997); (MacCormack, Kemerer, Cusumano, & Crandall, 2003); (Foulds, Quaddus, & West, 2007); (Rothenberger, Kao, & Van Wassenhove, 2010); (Otero, Centeno, Otero, & Reeves, 2012); (Cuauhtemoc Lopez-Martin, Chavoya, & Meda-Campana, 2014); (Moazeni, Link, & Boehm, 2014)
Tamanho da Equipe	Representa o número de desenvolvedores envolvidos no projeto de software.	9	(D. R. Jeffery, 1987); (Maxwell et al., 1996); (Morasca & Russo, 2001); (Wang, Wang, & Zhang, 2008); (Ahmed, Ahmad, Ehsan, Mirza, & Sarwar, 2010); (Narayan Ramasubbu, Cataldo, Balan, & Herbsleb, 2011); (Farshchi, Jusoh, & Murad, 2012); (D. Rodríguez, Sicilia, García, & Harrison, 2012); (Scholtes et al., 2016)
Experiência no Desenvolvimento de Software	Representa a percepção subjetiva da experiência do desenvolvedor no desenvolvimento de software. Pode ser observado através da aquisição de conhecimento de treinamento, *coaching*, realização de diferentes tarefas ou uso de métodos e técnicas executados em diferentes projetos de software ao longo do tempo.	8	(Hanakawa, Morisaki, & Matsumoto, 1998); (Maxwell & Forselius, 2000); (Munch & Armbrust, 2003); (Chang & Jiang, 2006/2006); (Mohapatra, 2011); (Nguyen, Huang, & Boehm, 2011); (Otero et al., 2012); (Cuauhtemoc Lopez-Martin et al., 2014);
Capacidade do Pessoal	Representa o nível de habilidade em que os membros da equipe executam com eficiência ou precisão suas tarefas em funções específicas durante o desenvolvimento do software.	8	(Card, 1987); (Krishnan, Kriebel, Kekre, & Mukhopadhyay, 2000); (Munch & Armbrust, 2003); (Chang & Jiang, 2006/2006); (Tan et al., 2009); (Farshchi et al., 2012); (Otero et al., 2012); (Kuchar & Vondrak, 2013);

Flexibilidade de Reuso	Seu objetivo é representar subjetivamente como o nível de flexibilidade na reutilização de requisitos, análise e design pode beneficiar projetos de software e aumentar sua produtividade.	8	(Maxwell & Forselius, 2000); (Morisio, Romano, & Stamelos, 2002); (Foulds et al., 2007); (Ramasubbu & Balan, 2007); (Tan et al., 2009); (Ahmed et al., 2010); (Nguyen et al., 2011); (Carrillo de Gea et al., 2016);
Nível de (Capacidade) Maturidade	Representa uma indicação da capacidade da organização em relação ao processo de desenvolvimento de software.	7	(Premraj, Shepperd, Kitchenham, & Forselius, 2005); (de Oliveira, Valle, & Mahler, 2009); (Gleison Santos et al., 2010); (Mohapatra, 2011); (Nguyen et al., 2011); (Di Tullio & Bahli, 2013); (Narayan Ramasubbu, Bharadwaj, & Tayi, 2015)
Metodologia de Desenvolvimento de Software	Indica o modo de desenvolver soluções de software adotadas nos projetos, por exemplo, desenvolvimento ágil, em cascata, baseado em componentes e assim por diante.	7	(Mizuno, Kikuno, Inagaki, Takagi, & Sakamoto, 2000); (Masateru Tsunoda, Monden, Yadohisa, Kikuchi, & Matsumoto, 2009); (de Souza Carvalho, Rosa, dos Santos Soares, da Cunha Junior, & Buiatte, 2011); (Narayan Ramasubbu et al., 2011); (Xu & Yao, 2013); (Kamma & G, 2014); (Narayan Ramasubbu et al., 2015)
Participação de *Stakeholders*	Representa a percepção subjetiva da participação do *stakeholder* no processo de desenvolvimento de software.	6	(Maxwell & Forselius, 2000); (Foulds et al., 2007); (Maxwell & Forselius, 2000)(Mohapatra, 2011); (Hsu & Hung, 2013); (Xu & Yao, 2013); (Narayan Ramasubbu et al., 2015);
Tamanho Funcional (baseado em Pontos de Função)	Representa a ideia de observar o tamanho funcional do software medido pela contagem de pontos de função (FP).	5	(Morasca & Russo, 2001); (Foulds et al., 2007); (Wang et al., 2008); (Masateru Tsunoda et al., 2009); (Masateru Tsunoda & Ono, 2014)
Complexidade dos Artefatos	Pretende-se observar empiricamente a complexidade dos artefatos de software (documentos, modelos, recursos, funcionalidades).	5	(Munch & Armbrust, 2003); (Colazo, 2008); (Rothenberger et al., 2010); (Mohapatra, 2011); (Nguyen et al., 2011);
Área de Negócios	Indica o domínio da aplicação dentro da organização/indústria que o projeto/sistema/software estará suportando.	5	(Myrtveit & Stensrud, 1999); (He, Yang, Wang, & Li, 2008); (Wang et al., 2008); (Masateru Tsunoda et al., 2009); (Masateru Tsunoda & Ono, 2014);

Linguagem de Programação	Indica a plataforma de desenvolvimento (relativa à codificação) utilizada para produzir o código fonte.	5	(He et al., 2008); (Masateru Tsunoda et al., 2009); (D. Rodríguez et al., 2012); (Masateru Tsunoda & Ono, 2014); (Huang, Sun, & Li, 2015);
Disponibilidade e Alocação de Desenvolvedores	Pretende representar a percepção de ausência/presença de recursos humanos no momento de realizar tarefas e atividades ao longo do desenvolvimento de software.	4	(Maxwell & Forselius, 2000); (Dongwon Kang, Jung, & Bae, 2011); (Schluter & Birkholzer, 2012); (Kuchar & Vondrak, 2013)
Volatilidade de Requisitos	Representa a percepção de como as mudanças ambientais que resultam em requisitos instáveis podem afetar a produtividade do software.	4	(Maxwell & Forselius, 2000); (J. Y.-C. Liu, Chen, Chen, & Sheu, 2011); (Otero et al., 2012); (Narayan Ramasubbu et al., 2015)
Exposição a Riscos de Software	Representa o nível de incerteza do projeto em relação a fatores organizacionais, ambientais, usuários, requisitos, planejamento e controle, com um impacto perceptível na medida em que o software pode suportar novas ou mudanças de funções em resposta às necessidades de negócios.	4	(Anderson & Ghavami, 1999); (Jeet, Bhatia, & Minhas, 2011); (Nguyen et al., 2011); (Di Tullio & Bahli, 2013);
Dispersão da Equipe	Pretende representar a relação entre a localização de equipes, capturando diferenças de fuso horário e diferenças geográficas.	4	(N. Ramasubbu & Balan, 2007); (Avritzer & Lima, 2009); (Nguyen et al., 2011); (Narayan Ramasubbu et al., 2011)
Uso de Ferramentas	Representa a indicação das tecnologias de software usadas que suportam o processo de desenvolvimento de software.	4	(Maxwell et al., 1996); (Maxwell & Forselius, 2000); (Mohapatra, 2011); (Nguyen et al., 2011)
Comunicação entre Desenvolvedores	Pretende apoiar a observação de como e se atrasos e interrupções nos canais de comunicação entre os desenvolvedores podem afetar a produtividade do software.	3	(Teasley, Covi, Krishnan, & Olson, 2002); (Avritzer & Lima, 2009); (Liang, Wu, Jiang, & Klein, 2012)

Conhecimento de Domínio dos Desenvolvedores	Pretende representar o nível de conhecimento dos desenvolvedores sobre o problema, incluindo operações organizacionais, o funcionamento dos departamentos de usuários, experiência e habilidade administrativa e experiência na área de aplicação do software em desenvolvimento.	3	(Munch & Armbrust, 2003); (Avritzer & Lima, 2009); (Hsu & Hung, 2013)
Qualidade do Produto	Representa uma indicação de se e como a ausência/presença de defeitos pode afetar a produtividade do desenvolvimento de software.	3	(Krishnan et al., 2000); (N. Ramasubbu & Balan, 2007); (Narayan Ramasubbu et al., 2011)
Implementação de Qualidade de Software	Representa uma indicação de se, e como, as práticas de qualidade de software (como inspeções, verificação, validação, gerenciamento de configuração, etc.) são aplicadas ou combinadas durante todo o processo de desenvolvimento de software.	3	(Krishnan et al., 2000); (N. Ramasubbu & Balan, 2007); (Rothenberger et al., 2010)
Diversidade da Equipe	Representa subjetivamente a heterogeneidade dentro da equipe em relação aos atributos individuais dos membros da equipe, como idade, sexo, origem étnica, educação, formação funcional, posse e habilidades técnicas.	3	(Lee, Lee, & Xia, 2010); (Günsel & Açikgöz, 2011); (P.-C. Chen, Chern, & Chen, 2012)
Autonomia da Equipe	Subjetivamente representa a medida em que a equipe de software é empoderada com a autoridade e controle na tomada de decisões para realizar o projeto.	3	(Lee et al., 2010) (Günsel & Açikgöz, 2011); (P.-C. Chen et al., 2012);
Duração do Projeto	Representa o tempo gasto para construir/excutar o projeto de software.	3	(Maxwell et al., 1996); (Masateru Tsunoda et al., 2009); (L. Liu, Kong, & Chen, 2015)

3.5. Ameaças à Validade

A codificação e a utilização de um critério para selecionar fatores que influenciam a produtividade do desenvolvimento de software também são consideradas ameaças à validade. A codificação é uma tarefa estritamente subjetiva e

foi minimizada pelo uso da GT, através de revisões por pares e apresentações ao Grupo ESE.

Para aumentar a confiança nos resultados encontrados, somente códigos com pelo menos três estudos relacionados foram levados em conta para uso como fatores do modelo proposto. Este critério visa minimizar a subjetividade, baseando-se na relação de um fator com a sua avaliação experimental em ao menos três estudos primários. A definição deste número de estudos primários visa eliminar fatores pouco investigados ou que coincidentemente foram investigados em apenas dois estudos diferentes, como um indício de percepção coletiva sobre a relevância ou importância do impacto de um fator.

Por outro lado, analisando o fato de que eventualmente alguns estudos não estão sendo considerados por causa da lacuna temporal entre o *snowballing* da revisão da literatura técnica e a confecção deste manuscrito, possivelmente as frequências de alguns fatores precisariam ser alteradas. Se estas alterações ocorrerem em fatores que já estão sendo levados em consideração para investigação, a lista final de fatores não será alterada. Se estas alterações ocorrerem naqueles fatores que não foram incluídos na lista final, então alguns fatores passariam a ser incluídos e, de acordo com o critério proposto, se referenciados em três estudos seriam também considerados. Assim, a probabilidade de precisar incluí-los à lista final seria a probabilidade de que tais fatores passassem a ser foco em pesquisas recentes (últimos 3 anos) e pode ser considerada baixa, até mesmo pela ausência de consenso sobre a direção das pesquisas no tema, conforme já citado.

De qualquer forma, o número de fatores considerado (25) já representa um avanço significativo em termos de conhecimento obtido através da pesquisa experimental. Nenhum dos estudos primários ou secundários apresentados nesta tese reúne tantos fatores com evidências experimentalmente avaliadas. Então, neste contexto assume-se que o acréscimo de mais dois ou três fatores nesta lista não mudará significativamente a abordagem nem os resultados desta tese. Neste momento, ter um ponto de partida em um referencial teórico no tema é mais importante e necessário para a organização do conhecimento e a evolução da pesquisa em produtividade do desenvolvimento de software do que determinar fatores adicionais, que podem ser obtidos em revisões subsequentes.

3.6. Conclusão

A perspectiva Clássica da GT se mostrou adequada para o intuito pretendido nesta tese: encontrar fenômenos, descrevê-los e registrar informações que levassem ao raciocínio de como combinar evidências quando levados à síntese.

De fato, a aplicação da GT ajudou na organização das informações. Ao longo da execução da análise o próprio template de memorando foi modificado algumas vezes, a medida que havia um amadurecimento e organização das ideias. O que levou sua versão final a se basear em conceitos presentes em Chidamber & Kemerer (1991).

Iniciando esta etapa da metodologia de pesquisa com 83 relatos de fatores, ao final da análise qualitativa se chegou a 25 códigos teóricos, que foram os fatores utilizados para a composição do *framework* proposto.

Boa parte destes códigos teóricos se encontram de alguma forma relatados em outras revisões (Trendowicz & Münch, 2009; de Barros Sampaio *et al.*, 2010; Paiva *et al.*, 2010; Ramírez-Mora & Oktaba, 2017). Entretanto, tais revisões buscam simplesmente identificar fatores sem uma preocupação em mensurar a intensidade do efeito das relações causais encontradas, bem como explicitar que características são utilizadas para operacionalizar medidas destes fatores, não explorando os fenômenos sob investigação em suas evidências. Estes aspectos serão contemplados nesta tese.

No próximo capítulo abordaremos a síntese das evidências com base nos fatores e fenômenos correlatos a partir dos códigos teóricos aqui descritos. Além disso, serão discutidos como os resultados alcançados se complementam a ideais já conhecidas e implicações de seu uso pela Comunidade de ES.

Capítulo 4: Modelo de Estruturas Teóricas em Produtividade do Desenvolvimento de Software

Este capítulo descreve os passos e resultados decorrentes das atividades realizadas para alcançar um modelo teórico representativo sobre fatores que influenciam a produtividade do desenvolvimento de software.

4.1. Definições e Considerações Iniciais

4.1.1. Definindo Teorias, Leis e Hipóteses

A prática deve ser a verdadeira medida da utilidade de um método porque este é o local onde os problemas diários são resolvidos. O conhecimento que nos ajuda a dominar tarefas práticas deveria ter a nossa maior atenção. Em computação é, portanto, necessária uma nova abordagem se quisermos fechar a lacuna entre teoria e prática (Endres & Rombach, 2003; Shull & Feldmann, 2008; Sjøberg, Dybå, Anda, & Hannay, 2008; Stol & Fitzgerald, 2015; Stol et al., 2016). Entretanto, há expectativas discrepantes em relação a descoberta e desenvolvimento de conhecimento na engenharia e na ciência.

Na ciência espera-se chegar a uma proposição teórica, que possa ser derivada de modelos de hipóteses testadas, provadas matemática ou experimentalmente aceitas (Poser, 1998). Caso contrário o conhecimento teórico precisa ser revisto e/ou arranjo experimental mudado. Em última análise, uma teoria ou lei precisa ser mudada ou provisoriamente rejeitada até que o fenômeno possa ser novamente descrito de acordo com uma mudança sobre seu entendimento (Shull & Feldmann, 2008; Sjøberg, Dybå, Anda, & Hannay, 2008).

Na engenharia, o conhecimento parte da definição de um problema a ser resolvido, um objetivo (Poser, 1998). A partir deste objetivo, estratégias para resolvê-lo são desenvolvidas através da criação, adaptação, correção de artefatos ou perseguindo um objetivo através de métodos. A avaliação do sucesso ou falha no alcance de um objetivo é realizada por um programa ou iniciativa de medição com medidas apropriadas, que reflitam o resultado esperado (Briand, Morasca, & Basili, 2002; Florac & Carleton, 1999; Humphrey, 1988). Se o método não alcança o objetivo esperado, ações corretivas ou adaptações devem ser tomadas. Caso contrário, novos problemas são atacados ou soluções melhores ou similares a custos menores são buscadas.

Em ES, pesquisadores têm pregado que os profissionais devem ser reeducados (Johnson, Ekstedt, & Jacobson, 2012), e fazer melhor uso dos resultados teóricos disponíveis (Wohlin, Šmite, & Moe, 2015). Endres & Rombach (2003) creditam o problema aos teoristas (aqueles que desenvolvem ou conhecem cientificamente certas ordens de fatos, ou seja, princípios fundamentais de uma ciência). Em vez de desenvolverem teorias, pesquisadores em ES frequentemente argumentam em favor de métodos e ferramentas específicas, sem acordo suficiente (consenso) de sua utilidade na prática nem após validá-las por avaliações realmente significativas. Entretanto, alguns pesquisadores (Endres & Rombach, 2003; Shull & Feldmann, 2008; Sjøberg, Dybå, Anda, & Hannay, 2008; Stol & Fitzgerald, 2015; Stol et al., 2016) argumentam ainda que o desenvolvimento de métodos (práticas e técnicas) deveria ser regido por princípios básicos baseados na formulação de leis ou teorias. A partir disto, é que novas ferramentas deveriam ser projetadas para apoiar estes métodos.

Teorias provêm meios para retratar de modo organizado um fenômeno do mundo real onde alguma complexidade do fenômeno real é reduzida na sua construção (Santos, 2015), e tendem a facilitar a comunicação de ideias e conhecimento ao oferecerem um referencial conceitual comum para estruturá-los de forma concisa e precisa (Reynolds, 2015).

Além disso, teorias são construídas e evoluem ao longo do tempo (Suppe, 1977). Novos conhecimentos ou observações podem acarretar em mudanças sobre o entendimento de um determinado fenômeno, e que podem precisar ser explicadas através de novas teorias. Assim, mesmo uma lei, algo tomado como verdade, pode e deve ser contestada, quer porque o problema que aborda tornou-se irrelevante, ou porque as circunstâncias que conduzem a uma certa conclusão mudaram ao longo do tempo (Endres & Rombach, 2003).

Neste contexto, quando nos referimos a um modelo de estruturas teóricas ou um *framework* teórico, estamos falando da representação de um conjunto de proposições (teoria, leis, hipóteses, conjecturas, dentre outras) que visam expressar o conhecimento sobre um tema, explicando o acontecimento de alguns fenômenos com algum nível de certeza associada às intervenções realizadas. Assim, o termo "estrutura teórica" refere-se mais sobre a representação do que sobre a concepção teórica "completa" (os elementos interpretativos e explicativos), isto é, trata somente dos elementos necessários para explicitar os conceitos e relações de uma proposição.

Apesar disto, ainda precisamos distinguir aqui diferenças entre teoria, leis, hipóteses e conjecturas. Para uniformizar o entendimento sobre os termos discutidos

ao longo deste capítulo, adotaremos as seguintes definições (Endres & Rombach, 2003):

- Observações podem ser fatos ou impressões simplesmente subjetivas. Observações são realizadas durante todo o dia usando todos os nossos sentidos. Atualmente, podemos até mesmo fazer observações de sinais que não são detectáveis com nossos sentidos, tais como ondas elétricas ou partículas radioativas. No entanto, não podemos sempre encontrar regularidades ou padrões recorrentes nas observações e muitas vezes observações não-repetíveis são consideradas como acidentes ou colocadas de lado, como erros de observação.
- A conjectura é apenas uma proposição que sugere uma suposição. Algumas pessoas podem acreditar nela, mas (ainda) não se encontram muitas que compartilham da mesma visão.
- Uma hipótese é uma proposição que só é provisoriamente aceita, sobre qual se pretende realizar algum tipo de teste capaz de confirmá-la ou refutá-la. Se chamamos algo de uma hipótese, isso não significa necessariamente que estamos de acordo; provavelmente temos dúvidas sobre a sua validade. Caso contrário, poderíamos ter olhado mais rigorosamente para obter uma evidência com alto nível de confiança, para que pudéssemos transformá-la em uma lei.
- Citando uma determinada proposição como uma lei, queremos dizer que há evidência experimental forte para apoiá-la, e que representa uma visão importante regida por determinados princípios. Se elevamos algo a uma lei, estamos sugerindo que ela seja aceita como verdade, pelo menos por enquanto ou para um dado conjunto de situações, e que estudos futuros devem ser dirigidos a outras observações, talvez em novas hipóteses e conjecturas. Sempre que são realizadas observações recorrentes, pode-se querer reagir a elas, se ajustando ou tirando proveito delas. Por exemplo, as *Leis de Newton* comumente se aplicam para descrever fenômenos dentro de um limite de dimensões e velocidades, aqueles facilmente observáveis pelo ser humano. Já em escalas atômicas e sub-atômicas e velocidades bem abaixo da velocidade da luz, os conceitos de força, posição e momento linear não podem ser totalmente explicadas pela Mecânica Clássica, e daí o desenvolvimento da Mecânica Quantica. Já para os casos de velocidades próximas à da luz em dimensões muito grandes, desenvolveu-se a Teoria da Relatividade. Em geral, observações

repetíveis com seus resultados baseados em diversas verificações experimentais, ou seja, com reconhecida habilidade em predizer tais resultados, podem ser declaradas muitas vezes como uma lei. Usando leis, podemos então fazer previsões sobre novas observações. As leis são generalizações de uma situação para a outra, ou a partir de um período de tempo para outro. A lei nos diz como as coisas ocorrem, mas não porque ocorrem. Por exemplo, as *Leis de Kepler* descrevem como, mas não porque os planetas se movem.

- Teoria: Teorias explicam e orquestram as nossas observações. Para obtê-las temos de nos perguntar: “Por que” um determinado fenômeno ocorre. Por exemplo, se gostaríamos de ter uma argumentação que explique a existência de uma lei e, consequentemente, porque esta lei pode ser tida como verdade, nós gostaríamos de ter uma teoria.

Um entendimento resumido das relações entre os conceitos apresentados pode ser visto na Figura 14. Nesta figura podemos verificar que conforme o conhecimento sobre um fenômeno vai sendo aumentado a força experimental das evidências também. Este aumento se dá através do processo experimental, com a formulação de hipóteses e a aceitação por parte considerável de um grupo de indivíduos, até que seja formulada uma lei. Quando se encontra uma explicação para a ocorrência do fenômeno temos uma teoria.

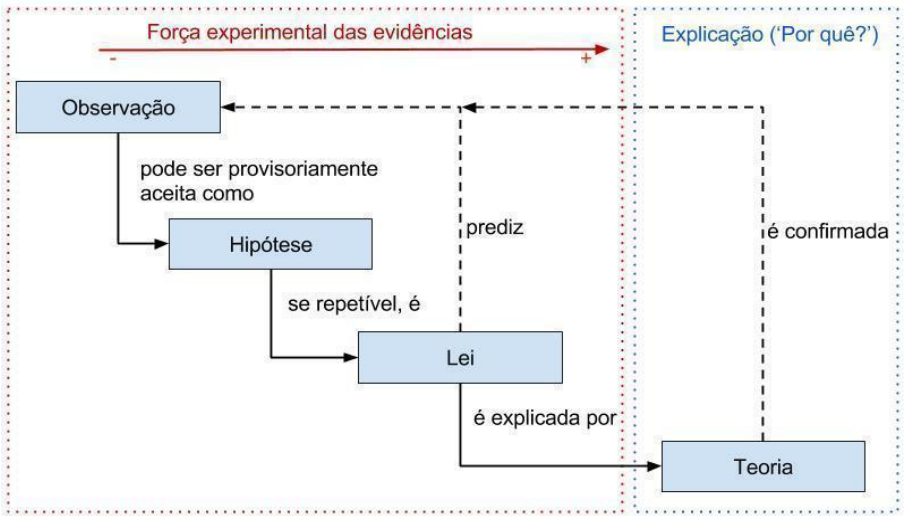

Figura 14. Relações entre conceitos (adaptado de Endres & Rombach, 2003).

Entretanto nada impede que uma teoria seja postulada sem uma evidência ou observação concreta. Por exemplo, em 1916 Einstein predisse a existência teórica de ondas gravitacionais como consequência da teoria da relatividade. O primeiro resultado considerado uma evidência da existência de tais ondas só pode ser

mensurado indiretamente mais de 50 anos depois, com o experimento proposto por Hulse & Taylor (1975). Mas somente em 2015 a detecção direta de ondas gravitacionais e a confirmação da predição de Einstein foi possível com o advento do LIGO (*Laser Interferometer Gravitational-Wave Observatory*) (Castelvecchi & Witze, 2016) e, consequentemente, a teoria pôde ser confirmada.

Em Engenharia de Software, a pesquisa experimental tem basicamente evoluído através de estudos primários (Stol & Fitzgerald, 2015). No contexto dos estudos primários em ES, em geral, busca-se capturar o impacto de introdução ou a variação de efeitos em processos, atividades, tarefas ou indivíduos no uso de tecnologias propostas em situações bem limitadas. O cerne das pesquisas, na maioria das vezes, tem focado na proposição de hipóteses, que pouco levam em consideração estudos já realizados, a interseção destas hipóteses com outras já testadas, ou como conceitos gerais ou o referencial teórico são compartilhados. Parte disso pode ser constatado se considerarmos a quantidade de repetições em relação à quantidade total de estudos primários encontrados na literatura técnica. Repetimos pouco.

Em ES, repetições de estudos primários são importantes como meio de aumentar o nível de confiança de resultados previamente conhecidos quando há convergência de resultados. Mesmo quando um conjunto de evidências “congêneres” possuem resultados divergentes, repetições permitem que pesquisadores reflitam sobre as deficiências dos estudos executados ou identifiquem a necessidade de reformular hipóteses, permitindo que um entendimento sobre determinado fenômeno evolua.

De qualquer forma, como pesquisadores e profissionais da prática em ES, sempre temos de avaliar se há uma oportunidade para transformar hipóteses ou observações em leis ou teorias para substituí-las por novas. Se uma lei pode ser razoavelmente explicada e provavelmente aceita por um nicho de pesquisadores, pode-se propor uma teoria.

Neste capítulo, o intuito é discutir e demonstrar o modelo de estruturas teóricas, capazes de capturar proposições relacionadas a produtividade do desenvolvimento de software. Explicitando as relações entre fatores e a produtividade do desenvolvimento de software por meio da descrição dos tamanhos de efeito (*effect size*) e confianças resultantes das sínteses das evidências. Além disso, os resultados obtidos serão interpretados, sempre que possível, de acordo com um outro conjunto de proposições, onde há uma concordância significativa entre os especialistas no campo da Engenharia de Sistemas e Software (Endres & Rombach, 2003). Assim,

espera-se que este segundo conjunto sirva, de alguma forma, para avaliar sua convergência com os resultados alcançados na síntese das evidências.

4.2. Abordagem de Construção do Modelo de Estruturas Teóricas

De acordo com Santos (2015), o processo de síntese de evidências não só é dependente de algum tipo de representação para determinar as suas capacidades de combinação, mas também de um formalismo da incerteza associada à evidência para descrever as tendências relevantes considerando os estudos analisados. Ao selecionarmos uma abordagem capaz de lidar com a incerteza, podemos representar o quão provável será, de fato, observar um determinado fenômeno dado um ambiente ou contexto.

Isto não significa que necessariamente conhecemos a distribuição das diferentes manifestações daquele fenômeno, mas que o conhecimento que temos das ocorrências de um conjunto de fatos nos levam a acreditar em determinadas chances de ocorrência de certos efeitos.

Assim, a incerteza nos possibilita expressar o nível de desconhecimento ou ignorância com relação aos fatos sobre os quais não tenha conhecimento da ocorrência ou, tão pouco, a manifestação (efeito) de um fenômeno. Assim, também é possível lidar com a ignorância sobre um fato, indicando a falta de conhecimento sobre resultados advinda normalmente de observações não sistêmicas.

Particularmente importante em ES, em que existe uma considerável heterogeneidade de estudos primários, a utilização de probabilidades para lidar com incertezas é crítica. Pois a incerteza nos dá uma ideia do risco em se utilizar os resultados de uma evidência.

Como grande parte dos estudos primários em ES não permite a comparação e agregação de evidências através de meta-análise, tanto pela falta de uso de medidas comuns e bem definidas, como pela pouca repetição de estudos primários ou falta de grupos de controle ou comparação (Kitchenham, Dyba, & Jorgensen, 2004; Miller, 2000, 2005; Travassos *et al*., 2008), optamos por utilizar um método de síntese denominado Método de Síntese Estruturada (Structured Synthesis Method, SSM) (Santos, 2015), que pode representar indistintamente resultados de estudos experimentais tanto quantitativos quanto qualitativos através de um mecanismo único de representação, permitindo a combinação de diferentes tipos de evidência.

O método define um conjunto de mecanismos de descrição de estruturas teóricas relacionados à representação dos fatos observados (em termos de variável dependente, variáveis independentes, variáveis de contexto) e atribuição das probabilidades de certezas de acordo com o rigor experimental de uma determinada evidência. Após a representação e atribuição das certezas a um conjunto de evidências, o pesquisador pode realizar as sínteses desejadas, desde que respeitados alguns mecanismos semânticos, que visam garantir a consistência da representação de conceitos após uma agregação. O SSM oferece uma mistura interessante de síntese integrativa e interpretativa (Cruzes & Dybå, 2010).

Primeiramente, SSM confronta um conjunto de evidências aos aspectos interpretativos da síntese, ou seja, os aspectos da representação de evidências. Tais aspectos se referem à organização e descrição de construtos e as suas relações para descrever variáveis (independentes ou dependentes), os efeitos causais observados (as hipóteses testadas, por exemplo, A afeta B ou C modera o efeito de A sobre B) e as características contextuais de evidência.

Um construto define um conceito. No contexto desta tese, um fator é definido como um construto mensurável direta ou indiretamente por medidas. As relações entre diferentes construtos é que representam as proposições teóricas do modelo, ou seja, são as descrições de fenômenos.

Para isso, a SSM usa uma representação diagramática para colocar todas as evidências no mesmo formato de descrição e tornar sua combinação mais objetiva. Nesta tese, isso implica que os fatores encontrados na revisão sistemática da literatura (Capítulo 2) foram representados como conceitos, e os efeitos causais entre eles ou a produtividade do desenvolvimento de software foram capturados e expressos em relação a sua intensidade (módulo e direção).

Usando uma sintaxe similar a UML e um conjunto de construtos pré-definidos chamados arquétipos (Atividade, Ator, Tecnologia e Sistema), as evidências em ES são representadas de modo a descrever os fenômenos observados de acordo com a seguinte perspectiva: *"Um **Ator** (**Actor**) executa uma **Atividade** (**Activity**) com uma **Tecnologia** (**Technology**) para desenvolver um tipo de **Sistema** (**System**)"*. São estas representações gráficas de cada evidência, sob uma perspectiva, que são utilizadas para auxiliar um pesquisador a realizar a agregação de evidências. O resultado de uma agregação é também a representação gráfica de um fenômeno, uma nova evidência, que pode ser utilizada no futuro para novas agregações. Um exemplo de representação de uma evidência de acordo com o SSM pode ser visto na Figura

15, que é uma parte da representação de Middleton & Joyce (2012) de acordo com dos Santos *et al*. (2018).

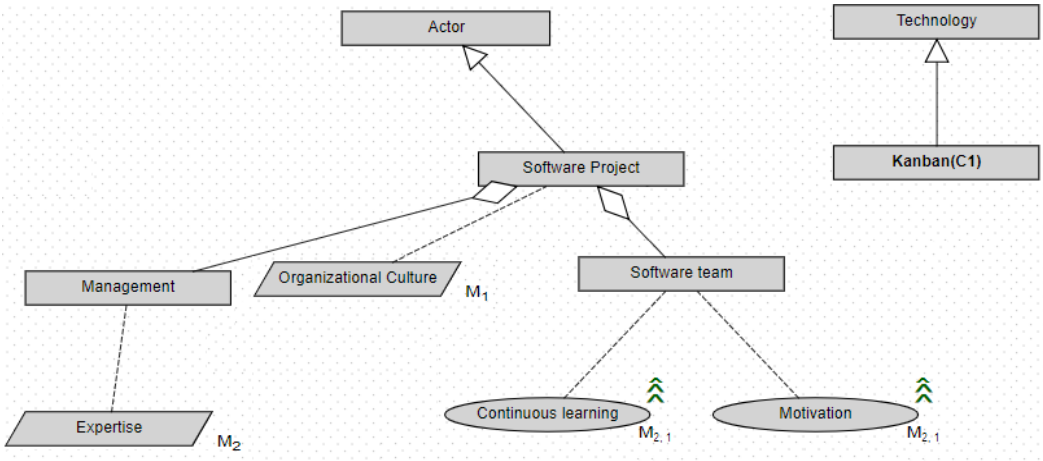

Figura 15. Parte da representação de Middleton & Joyce (2012) através de SSM, retirada de dos Santos (2018).

Os elementos que requerem mais atenção na interpretação do exemplo acima são:

- As elipses, que representam efeitos observados ou a variável dependente. A barra de preenchimento com um valor percentual abaixo representa a probabilidade de certeza atribuída àquele efeito, e o símbolo no canto superior direito representa o efeito mais provável de ser observado, utilizando uma escala de *Likert* de sete valores, que serão detalhados mais adiante.
- Os retângulos na parte superior da figura, os elementos *Activity*, *Actor*, *System* e *Tecnology*, são os arquétipos anteriormente citados. Estes sempre são elementos raízes dos quais a representação de qualquer outro conceito é derivada. São utilizados para garantir a consistência durante o procedimento de agregação de evidências.
- Os demais retângulos, que representam os demais construtos, sejam variáveis independentes ou variáveis de contexto. A sintaxe da SSM permite que mais de um efeito seja descrito numa representação gráfica de evidências, mas somente haverá uma única causa. Por este motivo é que não são necessárias representações das relações de causa-efeito Santos (2015).

- Os paralelogramos são variáveis moderadoras, ou seja, uma mudança na manifestação destas variáveis é capaz de amplificar ou diminuir o efeito da variável independente na dependente.

Para fins de interpretação, as folhas de cada árvore de arquétipos são os elementos que necessitam de maior atenção. O retângulo com rótulo em negrito indica o elemento causa, ou seja, um fator, variável independente ou preditor. Já os outros retângulos são variáveis de contexto, ou seja, ajudam a interpretação da descrevendo particularidades contextuais consideradas relevantes na representação da evidência. Assim, conforme o exemplo da Figura 15, a evidência em questão nos fornece o seguinte *insight*:

*"**Kanban** tem efeitos **positivos ou fortemente positivos** (⇈) na **Motivação (Motivation)** e no **Aprendizado Contínuo (Continuous Learning)** de uma **Equipe de Software (Software Team)**, e estes efeitos são moderados pela **Cultura Organizacional (Organiztional Culture)** e o **Expertise da Gestão (Management)** do **Projeto de Software (Software Project)**".*

Após representar cada uma das evidências, os aspectos integrativos do SSM induzem os pesquisadores a se concentrarem na agregação dos efeitos causais ou das relações de moderação. Para tal, um pesquisador deve decidir se ou quais conceitos, efeitos causais e características contextuais entre as diferentes representações se correspondem. Tendo elementos que não correspondem, existem as opções de distintamente mantê-los separados nos resultados agregados ou removê-los da síntese. O resultado da combinação ainda será uma representação dentro da sintaxe definida.

No entanto, depois de determinar como e quais evidências podem ser combinadas conceitualmente, a SSM usa a Teoria de *Dempster-Shafer* (TDS) (G. Shafer, 1976, 1990) como o formalismo matemático para expressar eventos ou fenômenos incertos, permitindo a definição da incerteza da combinação de evidências a partir das incertezas de cada evidência. Aplicar a TDS no SSM requer duas informações necessárias para a síntese de evidências.

A primeira é a intensidade do efeito causal, o fato, apontado por cada elemento de evidência, que é representado sob a forma de subconjuntos de valores em um frame de discernimento. No caso do SSM, o frame de discernimento é composto por valores em uma escala de *Likert* de sete pontos, apresentada abaixo:

Θ = {Fortemente Negativo (↓↓↓), Negativo (↓↓), Fracamente Negativa (↓), Indiferente (↔), Fracamente Positivo (↑), Positivo (↑↑), Fortemente Positivo (↑↑↑)}

Note que a intensidade de um efeito causal pode assumir valores únicos (por exemplo, {↑↑}), assim como pode assumir conjuntos compostos (por exemplo, {↑, ↑↑} para fracamente positivo ou positivo, {↓, ↔} para fracamente negativo ou indiferente). Esta é uma alternativa para as medidas de tamanho de efeito das técnicas tradicionais de meta-análise, permitindo: (1) expressar uma imprecisão sobre o efeito notado, e; (2) combinar resultados qualitativos e quantitativos para que os estudos a serem combinados não precisem compartilhar as mesmas medidas.

A segunda informação é o valor de certeza relacionado a cada evidência, ou seja, o nível de certeza atribuído aos resultados de cada estudo primário, variando de 0% a 100%. Em outras palavras, quanta confiança os pesquisadores e profissionais podem atribuir aos resultados de cada uma das evidências? As certezas dos estudos primários são estimadas com base no seu tipo de estudo e avaliação de qualidade. Com base na hierarquia de evidências GRADE (Atkins *et al.*, 2004), os valores de certeza são limitados em quatro subescalas de acordo com o tipo de estudo primário: (1) de 0% a 25% para observações não sistemáticas; (2) de 25% a 50% para estudos observacionais; (3) de 50% a 75% para *quasi*-experimentos e; (4) de 75% a 100% para estudos controlados aleatorizados. O procedimento de avaliação de qualidade proposto por SSM usa formulários de avaliação para variar em 25% de acordo com o subintervalo do tipo de estudo.

A partir do momento em que as intensidades e os valores das certezas dos estudos primários são definidos, as evidências são confrontadas com os aspectos integrativos da SSM. Deste modo, a TDS atribui a cada uma das evidências uma função básica de atribuição de probabilidade, denominada função de certeza (*belief function*), que expressa valores de certeza atribuídos a diferentes fatos (neste caso, os efeitos observáveis das relações causais expressas em hipóteses, leis, dentre outros). Para combinar duas evidências, a TDS encontra uma nova função de certeza ($m(X)$), calculando a probabilidade da interseção dos fatos para determinar qual as probabilidades de todos os possíveis fatos dado a observação conjunta das evidências. Assim, a função de certeza resultante da combinação de evidências utilizando a TDS é uma função de certeza que, por definição, também pode ser utilizada na combinação de outras evidências.

As evidências são agregadas aplicando a Regra de Dempster (ver Equação (1)), que relaxa a inferência bayesiana para acomodar um componente de incerteza

m(Θ) e, então, pega as evidências e suas respectivas funções de certeza $m_1(B)$ e $m_2(C)$ e produz uma nova evidência que representa o consenso das peças originais, estabelecendo uma nova função de certeza ($m_3(A)$) (Shafer, 1976, 1990). Onde A, B e C são os possíveis subconjuntos do frame de discernimento, e m(Θ) representa a certeza associada a qualquer outra observação possível não contemplada pela evidência em questão.

$$m_3(\emptyset) = m_{1,2}(\emptyset) = 0$$

$$m_3(A) = m_{1,2}(A) \ = (m_1 \oplus m_2)\, A = \frac{1}{1-K} \sum_{B \cap C = A \neq \emptyset} m_1(B)\,.\,m_2(C)$$

$$K = \sum_{B \cap C = \emptyset} m_1(B)\,.\,m_2(C) \qquad (1)$$

Desta forma, quão maior a interseção entre os efeitos observados nas duas evidências maior a probabilidade da certeza da interseção. Consequentemente, quão maior a disparidade entre os efeitos observados, menor a certeza que se tem sobre o efeito esperado com a combinação das evidências, o que aumenta a probabilidade associada à ignorância sobre o fenômeno em questão. Assim, quanto maior o número de evidências e/ou maior a convergência entre seus resultados maior a chance de que um determinado efeito ou fato seja observável.

Uma vez que as funções de certeza para todos os efeitos causais são determinadas, o método apresenta o maior valor de certeza na respectiva função, que é denominado valor de certeza imediato, e sua respectiva intensidade levada em conta os resultados de síntese, como o resultado esperado mais imediato e provável. Em outras palavras, olhando para todos os possíveis efeitos de um fator, o método apresenta a intensidade do efeito mais provável de ocorrer a qualquer momento, ou seja, uma visão de curto prazo.

Por exemplo, vamos considerar a síntese de duas evidências que observaram experimentalmente o efeito da diversidade de equipes na produtividade do desenvolvimento de software. O primeiro trabalho (Günsel & Açikgöz, 2011) reporta um efeito fracamente positivo ({↑}) na produtividade do desenvolvimento de software. O segundo trabalho (Chen *et al.*, 2012) reporta um efeito negativo ou fracamente negativo ({↓↓, ↓}). Seja m_1(b) e m_2(c) as funções de certeza obtidas para cada estudo, respectivamente. Usando o SSM, os valores de certeza imediatos atribuídos a cada um são, respectivamente, m_1({↑}) = 31,7% e m_2({↓↓, ↓}) = 31,8%. Consequentemente, as respectivas incertezas, ou seja, a chance relacionada ao desconhecimento sobre a

ocorrências de efeitos não atrelados às evidências, são $m_1(\Theta)$ = 1- $m_1(\{\uparrow\})$ = 68,3% e $m_2(\Theta)$ = 1- $m_2(\{\downarrow\downarrow, \downarrow\})$ = 68,2%.

Como pode ser visto na Tabela 6, a Regra de Dempster é aplicada iterativamente a cada combinação possível de intersecções entre $m_1(b)$ e $m_2(c)$ para calcular a função de certeza da nova evidência ($m_{1,2}(a)$). Note que $\{\downarrow\downarrow, \downarrow\} \cap \{\uparrow\} = \emptyset$, $\{\downarrow\downarrow, \downarrow\} \cap \Theta = \{\downarrow\downarrow, \downarrow\}$ e assim por diante. Onde *K* é o nível de conflito, calculado como a soma das ausências de intersecções entre os efeitos relatados nos estudos primários e está associado ao nível de contradição entre as evidências. Observe que nesse exemplo K é composto de uma única parcela possível, na ausência de interseção $m_1(\{\uparrow\}) \cap m_2(\{\downarrow\downarrow,\downarrow\})$. Então, $m_{1,2}(\{\downarrow\downarrow,\downarrow\})$ = $m_1(\Theta)$. $m_2(\{\downarrow\downarrow,\downarrow\})$ / 1-K = 68,3% . 31,8% / 1-K = 21,7% / 1-K = 21,7% / (100%-10,07%) = 24,1%, e assim por diante.

Tabela 6. Exemplo da Síntese de Günsel & Açikgöz (2011) and Chen *et al.*, (2012)

	$m_2(\{\downarrow\downarrow,\downarrow\})$ = 31.8%	$m_2(\Theta)$ = 68,2%
$m_1(\{\uparrow\})$ = 31.7%	$m_{1,2}(\emptyset)$ = 0.0% K= $m_1(\{\uparrow\})$. $m_2(\{\downarrow\downarrow,\downarrow\})$ = 10,07%	$m_{1,2}(\{\uparrow\})$ = 24.0%
$m_1(\Theta)$ = 68.3%	$m_{1,2}(\{\downarrow\downarrow,\downarrow\})$ = 24.1%	$m_{1,2}(\Theta)$ = 51.8%

Então, quanto maior o nível de conflito, mais divergentes são os resultados individuais dos estudos selecionados para a síntese. Por outro lado, conforme diminui o nível de conflito mais convergentes os resultados individuais dos estudos.

O SSM e a TDS foram satisfatoriamente usados para agregar evidências sobre arquitetura de software (Martinez-Fernandez, Santos, Ayala, Franch, & Travassos, 2015) , Usage-Based Reading (Santos & Travassos, 2017) e os benefícios do uso de Kanban em ES (dos Santos *et al.*, 2018), além de já contar com uma ferramenta CASE para apoiar a representação gráfica, pesquisa e comparação de evidências. A ferramenta, chamada ***Evidence Factory*** (dos Santos & Travassos, 2017), foi utilizada durante a realização deste trabalho, e está disponível em http://evidencefactory.lens-ese.cos.ufrj.br/, onde os resultados de todas as sínteses estão descritos e disponíveis. Mais detalhes sobre o método podem ser vistos em Santos (2015).

Esta etapa da metodologia de pesquisa propôs utilizar TDS e *Evidence Factory* como abordagem para representar, agregar as evidências e fatores encontrados nas

etapas anteriores e, então, usar as representações das sínteses para encontrar um modelo causal obtido através das estruturas teóricas dos fatores que influenciam a produtividade do desenvolvimento de software, de acordo com o evidenciado pela literatura científica no tema.

A Tabela 7 apresenta o resultado da síntese de cada fator ordenado pelos maiores valores de certeza imediata, em termos de efeito mais provável, a certeza associada (imediata) e o nível de conflito observado. Talvez seja contra intuitivo, mas o nível de conflito pode ser pensado como a ausência de conhecimento sobre como medir ou prever um fenômeno e servir como um vislumbre de quão abstrato e desafiador é trabalhar com um fator ou construto atualmente. Assim, este trabalho fornece uma primeira indicação de quais fatores devem ser melhor compreendidos quando se olha para a produtividade do desenvolvimento de software. A coluna *Ganho de Certeza* refere-se à diferença entre os valores de certeza agregados e o valor de certeza mais alto atribuído a um único estudo selecionado, representando o ganho de confiança após a respectiva síntese.

Tabela 7. Resultado das Sínteses.

Código/Fator	*# de Estudos*	*Nível de Conflito*	*Intensidade*	*Certeza Imediata*	*Ganho de Certeza*
Tamanho de Software baseado em LOC	9	0,2%	{↑, ↑↑}	96%	18%
Volatilidade de Requisitos	4	0,0%	{↓↓, ↓}	85%	14%
Capacidade do Pessoal	8	0,0%	{↑ , ↑↑}	84%	23%
Experiência no Desenvolvimento de Software	8	0,0%	{↑}	75%	34%
Conhecimento de Domínio dos Desenvolvedores	3	0,0%	{↑,↑↑}	74%	14%
Autonomia da Equipe	3	0,0%	{↑, ↑↑}	72%	7%
Linguagem de Programação	5	16,9%	{↑, ↑↑}	72%	4%
Disponibilidade e Alocação dos Desenvolvedores	4	0,0%	{↑,↑↑}	71%	5%
Comunicação entre Desenvolvedores	3	0,0%	{↑↑}	70%	36%
Nível de Maturidade (Capacidade)	7	5,2%	{↑, ↑↑}	70%	11%

Área de Negócios	5	4,0%	{↓, ↔}	68%	1%
Complexidade dos Artefatos	5	6,0%	{↓, ↔}	67%	29%
Flexibilidade de Reuso	8	6,9%	{↑↑}	65%	3%
Implementação de Qualidade de Software	3	0,0%	{↑,↑↑}	62%	20%
Duração do Projeto	3	0,0%	{↓↓ , ↓}	59%	-8%
Metodologia do Desenvolvimento de Software	7	34,1%	{↓, ↔}	56%	-11%
Tamanho da Equipe	9	16,5%	{↓↓ , ↓}	56%	-22%
Uso de Ferramentas	4	28,5%	{↑,↑↑}	49%	10%
Participação de Stakeholders	6	28,2%	{↑,↑↑}	46%	-19%
Dispersão da Equipe	4	15,2%	{↓↓↓, ↓↓}	40%	-25%
Resposta à Mudança	2	0,0%	{↓↓↓, ↓↓}	35%	0%
Tamanho Funcional (baseado em PF)	5	20,0%	{↑↑}	33%	-37%
Qualidade de Produto	3	20,1%	{↑, ↑↑}	32%	-9,3%
Exposição a Riscos de Software	4	17,2%	{↓↓}	28%	-42%
Diversidade da Equipe de Software	3	16,9%	{↔}	22%	-46%

4.3. Achados e Interpretação dos Resultados

4.3.1. Modelo de Estruturas Teóricas da Influência de Fatores na Produtividade do Desenvolvimento de Software

Os resultados das sínteses e a utilização da ferramenta *Evidence Factory* permitiram a análise e construção do modelo proposto nesta tese através da extração e representação dos efeitos causais entre os fatores encontrados e a produtividade do desenvolvimento de software, bem como entre eles. Com base nisso, as discussões dos resultados serão abordadas ao longo deste capítulo.

Ao contrário de Martinez-Fernandez *et al.* (2015), que utilizou a *Evidence Factory* e discutiu os resultados de suas sínteses em relação à presença ou à falta de ganho de certeza, e dos Santos *et al.* (2018), que discute as sínteses em relação aos

benefícios observados pelo uso de Kanban em ES, esta tese propôs uma abordagem diferente. Dividir todos os efeitos causais das sínteses em quartis por valores de certezas imediatas. Assim, pretende-se igualmente distribuir os resultados das sínteses através de uma possível falta de conhecimento sobre estes fenómenos, com base unicamente naqueles que foram identificados como significativos para esta discussão.

Esses quartis foram organizados de acordo com suas respectivas metades e discutidas através delas, construídos a partir da conjunção de sínteses, explicitando todos os efeitos causais. A primeira metade contém os efeitos causais a serem cautelosamente empregados em um ambiente industrial e somente se necessário, porque contém os fenômenos que devem ser melhor compreendidos e estudados em investigações futuras. O segundo é o conjunto de fenômenos os quais podem ser empregados diretamente e terem seus resultados reforçados ou contestados por investigações experimentais adicionais, sem qualquer impedimento ou alto risco ao usá-lo na prática. Com isso, estamos advogando que a utilização dos valores imediatos das certezas pode servir como um indicativo da confiança (ou risco) que podem ser atribuídos aos resultados relatados pelas sínteses.

A representação final do modelo mantém os elementos da sintaxe de representação do SSM (retângulos, elipses e paralelogramos). As relações causais agora precisam ser representadas (setas), uma vez que não há somente um construto causal. Além disso, as informações referentes às relações causais estão explicitamente expressas por meio da tupla (<efeito>, <certeza imediata>).

4.3.2. Efeitos Causais com Menores Níveis de Confiança: A Primeira Metade do Modelo

Os efeitos causais com os menores níveis de confiança representam o conjunto de fenômenos que os pesquisadores parecem ter pouco conhecimento e, consequentemente, sobre os quais a capacidade de previsão de nosso modelo é, sem dúvida, limitada. A Figura 16 mostra esta metade do modelo obtido.

Os fatores destacados em cinza são aqueles nos quais os efeitos causais estão no primeiro quartil (certezas <= 35%). Como pode ser visto na Figura 16, a produtividade e qualidade do produto são representadas como elipses porque foram reconhecidas na literatura técnica como medidas de desempenho no conjunto de evidências. Produtividade como uma medida de eficiência, e a qualidade do produto como uma medida da eficácia de projetos de desenvolvimento de software. Os fatores representados como paralelogramos são moderadores. Note também que o Tamanho Funcional aparece nesta metade e o Tamanho baseado em LOC não. Embora os efeitos causais sejam próximos em relação à intensidade, não está claro se eles representam e medem a mesma característica do produto.

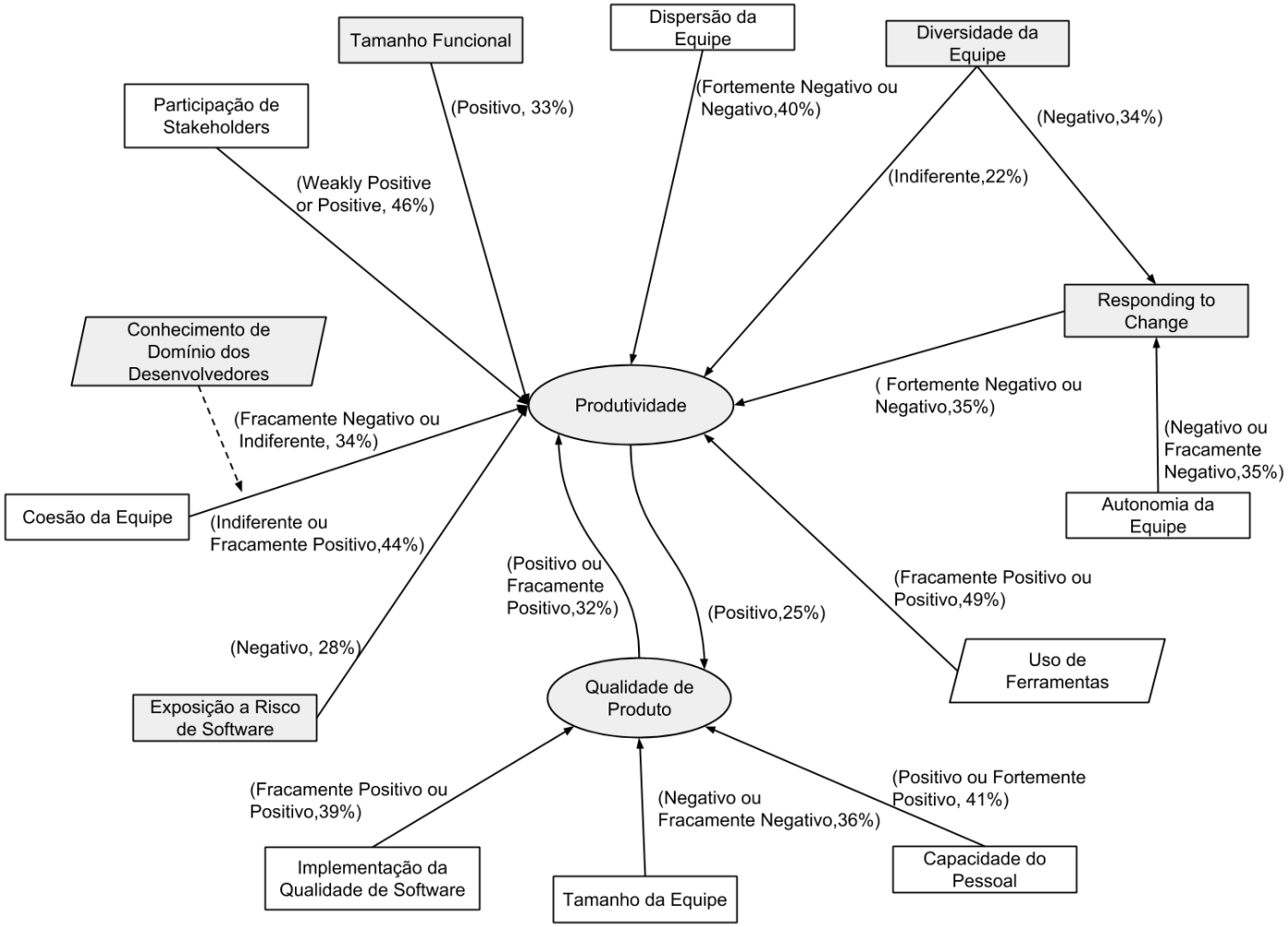

Figura 16. Primeira Metade do Modelo de Estruturas Teóricas.

No segundo quartil (certezas > 35% e <=49%), pode-se destacar, por exemplo, que o fator Uso de Ferramentas possui efeitos nas duas metades do modelo, sendo o efeito causal abaixo da mediana sua relação direta com a produtividade do desenvolvimento de software. Consequentemente, há uma relação de moderação pertencente à segunda metade dos resultados. O Nível de Conhecimento de Domínio dos Desenvolvedores modera a relação entre a Coesão da Equipe e a produtividade do desenvolvimento de software, além de ter um efeito causal direto na produtividade do desenvolvimento de software na outra metade.

Além disso, existe uma diferença significativa entre as confianças das diversas sínteses, e isto não pode ser desprezado. Talvez o fato do maior número de estudos se concentrar na segunda metade e ser quase duas vezes maior que a primeira explique esta diferença observada, mas, de qualquer forma, tal fato merece mais atenção no tema.

Os efeitos causais nesta metade são considerados tão incipientes que esses fenômenos não adquiriram confiança suficiente por enquanto. Isso significa que mesmo esses efeitos causais possam ser medidos, no curto prazo eles não serão interpretados corretamente. Em parte por causa da variação amplamente esperada em tais observações, em parte devido à falta de compreensão sobre se as medidas atualmente empregadas para sua observação no ambiente organizacional são apropriadas. Por outro lado, ao incorporar tais resultados em iniciativas de melhoria e mensuração, os profissionais provavelmente não serão capazes de perceber qualquer ganho/perda de produtividade de desenvolvimento de software em relação a esses fatores. Portanto, os pesquisadores e profissionais devem estudá-los e observá-los melhor. Isto sugere que esses resultados devem ser inicialmente declarados como conjecturas ou simples questões de pesquisa (hipóteses iniciais), e são elas:

- A participação dos stakeholders afeta positivamente a produtividade do desenvolvimento de software?
- O tamanho funcional (baseado em pontos de função) afeta positivamente a produtividade de desenvolvimento de software?
- A dispersão da equipe afeta negativamente a produtividade do desenvolvimento de software?
- A diversidade da equipe afeta a produtividade do desenvolvimento de software?
- A autonomia da equipe afeta negativamente a produtividade do desenvolvimento de software?

- O uso de ferramentas afeta positivamente a produtividade do desenvolvimento de software?
- A produtividade do desenvolvimento de software afeta positivamente a qualidade do produto?
- A qualidade do produto afeta positivamente a produtividade do desenvolvimento de software?
- A capacidade do pessoal afeta positivamente a qualidade do produto?
- O tamanho da equipe afeta negativamente a qualidade do produto?
- O nível de implementação da qualidade do software afeta positivamente a qualidade do produto?
- O nível de exposição ao risco de software afeta negativamente a produtividade de desenvolvimento de software?
- A coesão da equipe afeta positivamente a produtividade do desenvolvimento de software?
- O nível de conhecimento do domínio dos desenvolvedores diminui negativamente o efeito da coesão da equipe na produtividade do desenvolvimento de software?

Pelo do uso do SSM e da *Evidence Factory*, se os pesquisadores realizarem mais estudos observando essas hipóteses, o arcabouço teórico proposto pode ser continuamente atualizado até que seus efeitos causais sejam melhor classificados quanto à confiança (valor de certeza imediata). Assim, o conhecimento sobre o tema pode ser continuamente avaliado e evoluído.

4.3.3. Efeitos Causais com Maiores Níveis de Confiança: A Segunda Metade do Modelo

A segunda metade dos efeitos causais inclui o conjunto de fenômenos em que os resultados da evidência convergem e, consequentemente, podem fornecer *insights* mais gerais em níveis de confiança mais altos (Figura 17).

Como um *framework* teórico inicial, esta parte do modelo indica apenas que, quando os resultados obtidos das evidências são convergentes, há poucas razões para persistir em observar os mesmos fenômenos quando vistos sob as mesmas perspectivas. Por outro lado, não se deve impedir que quem desconfia desses resultados tente refutá-los ou propor novas discussões sobre como medir e quais medidas seriam mais apropriadas nos estudos em andamento. Seria um movimento natural da Engenharia de Software Experimental como uma disciplina científica em busca de sua evolução contínua.

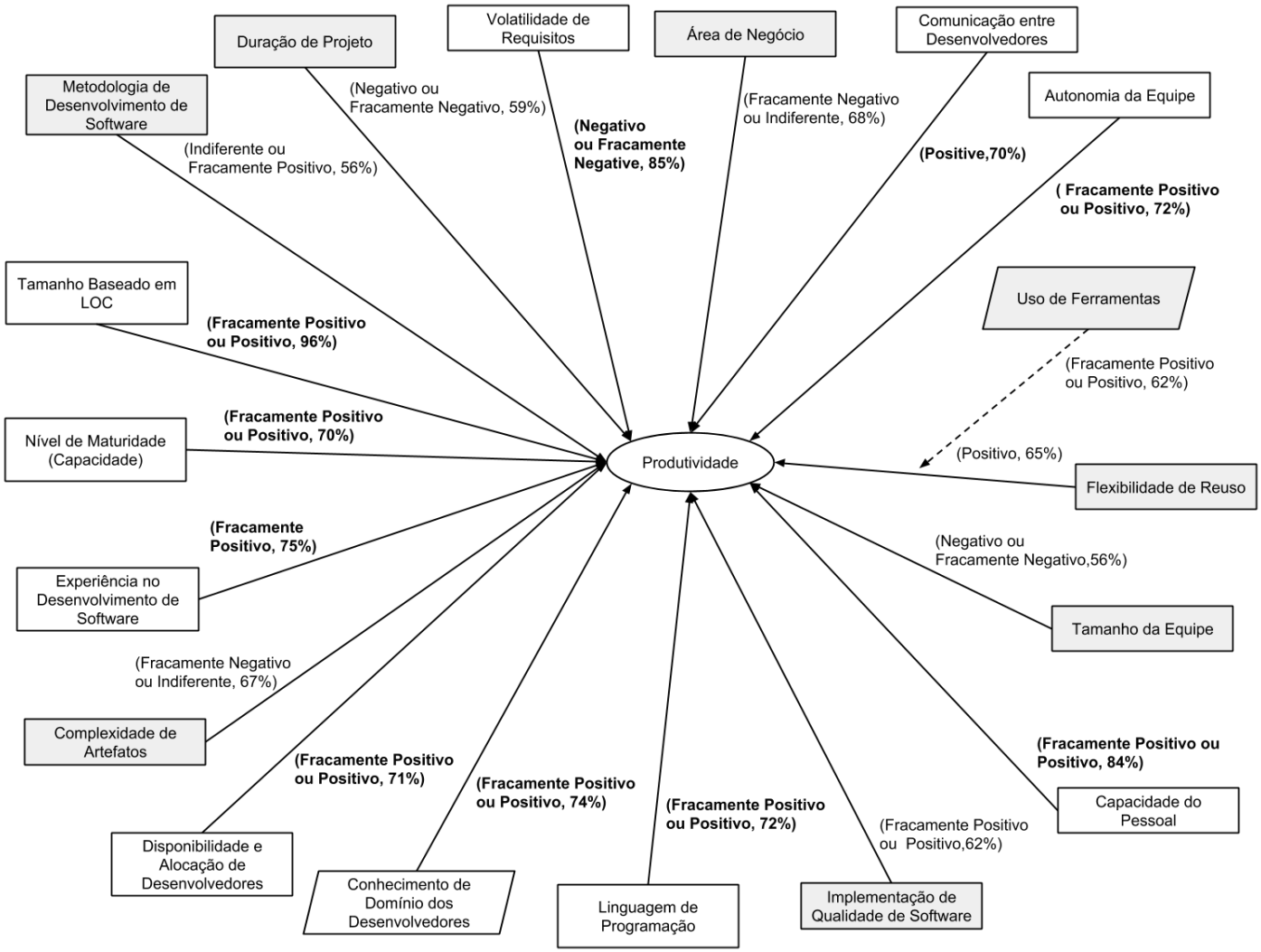

Figura 17. Segunda Metade do Modelo de Estruturas Teóricas

No terceiro *quartil* (certezas > 49% e <= 65%), observe, por exemplo, que o *Uso de Ferramentas* modera significativamente o efeito *Flexibilidade de Reuso* nesta metade dos resultados. No quarto *quartil* (certezas > 65%), o *Conhecimento de Domínio dos Desenvolvedores* mantém um alto nível de confiança. E ambos os fatores estão presentes nas duas metades do modelo, permeando-as por diferentes fenômenos.

4.3.4. Interpretação dos Resultados Mais Significativos

Nesta seção, os resultados com os níveis de confiança mais altos serão discutidos com relação a que tipo de *insights* podem ser extraídos deles. Vale destacar que esses *insights* se referem única e exclusivamente às evidências selecionadas e então analisadas ao longo da metodologia de pesquisa aplicada. Deste modo, as interpretações apresentadas ao longo desta seção não têm caráter exaustivo, nem confirmatório. Somente pretende-se interpretar sintética e sistematicamente os resultados das sínteses através de uma argumentação minimamente geral que possa se manter sustentada a partir do conjunto de evidências obtidas.

Em seu livro, Endres & Rombach (2003) selecionam um conjunto de *insights*, os quais os autores argumentam ver um nível significativo de concordância entre especialistas no campo de Engenharia de Sistemas e Software. Segundo eles, essas percepções permanecerão verdade por um tempo. O que representa que muitas pessoas compartilham suas ideias centrais, mesmo que não tenham sido suficientemente ou sistematicamente avaliadas.

Sempre que aplicável, os resultados de Endres & Rombach (2003) são confrontados com os resultados mais significativos do modelo encontrado, servindo como uma compilação baseada em evidências do *mindset* encontrado na Comunidade de ES. Seus achados serão usados para destacar diferenças ou semelhanças com nossos resultados. Espera-se, assim, complementar a análise qualitativa apresentada no capítulo anterior, quanto à questão de uma revisitação da literatura, mas sem um ponto de contraposição adequado naquele momento.

4.3.4.1. Em geral, quanto mais significativos forem os incrementos do tamanho de software baseado em LOC ao longo do tempo, maior será o impacto positivo na produtividade do desenvolvimento de software.

Tendo o número mais significativo de estudos e um maior nível de confiança nas sínteses, o tamanho do software medido através de linhas de código é o preditor mais perceptível da produtividade do desenvolvimento de software. Devido ao ganho de 18% do valor agregado das certezas sobre os estudos individuais melhor

classificados, o resultado mostra que os efeitos esperados podem variar, mas não muito. Todas as evidências da respectiva síntese demonstraram um efeito fracamente positivo ou positivo, exceto o estudo de Moazeni, Link & Boehm (2014), com efeito negativo e certeza imediata igual a 19%. E por este motivo obteve-se o resultado alcançado. Essa exceção é um *quasi*-experimento com estudantes, de modo que sua certeza imediata pouco contribuiu para contrapor a certeza imediata da combinação das outras evidências, nem pesa o suficiente para tornar o nível de conflito entre as evidências detectável usando uma casa decimal (0,0%). Mas sua importância é que esse estudo forneceu o *insight* para comparar todas as evidências levando em consideração a componente de tempo. Independente da duração, há um intervalo temporal de observação em todas as evidências do conjunto, seguido ou não de observações subsequentes. Assim, foi possível observar na combinação das evidências o efeito do crescimento do tamanho em LOC sempre sob uma mesma perspectiva.

Uma possível explicação é que inserções, remoções ou atualizações maiores de linhas de códigos em um produto ou *release* são mais naturais e, consequentemente, mais produtivas que as menores. No entanto, quais operações sobre código-fonte são mais benéficas para a produtividade do desenvolvimento de software não está claro (por exemplo, inserções são mais produtivas do que remoções e considerando atualizações como remoções antes de inserções), bem como se o tamanho baseado em LOC impacta diferentemente a produtividade de desenvolvimento de software de acordo com as diferentes características dos projetos (por exemplo, se o produto está em desenvolvimento ou manutenção, ou a duração do projeto - ambos relacionados ao ciclo de vida do produto). Então, entender como o tamanho baseado em LOC influencia a produtividade do desenvolvimento de software ao longo do tempo faz sentido, embora isso não tenha sido bem explorado em muitos casos.

De acordo com Endres & Rombach (2003), a *Lei de Corbató* afirma que *"há uma relação entre o comprimento do texto do programa e o nível da linguagem de programação"*, e isto contribui positivamente para a produtividade do desenvolvimento de software. Dependendo do nível das linguagens de programação, os desenvolvedores podem pular detalhes específicos, como camadas de armazenamento e otimizações de código e, assim, projetar e produzir software escrevendo menos linhas de código. Isso significa que quanto mais linhas de código são escritas, mais produtivo é o desenvolvimento do software. Analisando esse fator isoladamente, nossos resultados reforçam essa lei no sentido de que um aumento no

tamanho com base em linhas de código aumenta a produtividade do desenvolvimento de software. Embora, em nossa opinião, a argumentação relacionada a essa lei no livro pareça estar mais relacionada aos efeitos do uso de linguagens de programação de alto nível (por exemplo, Prechelt (2000) appud Endres & Rombach (2003)).

Com base nessas considerações e nos resultados das sínteses, sugerimos que o número de alterações relacionadas a linhas de código ao longo do tempo possa ser usado para explicar a flutuação na produtividade do desenvolvimento de software. Pode ser contra intuitivo, mas tal observação pode ser encarada como uma consequência da *Lei de Basili-Möller* (*"mudanças menores têm uma densidade de erro maior do que as grandes"*). Se mudanças menores resultarem em maior densidade de erro, o esforço necessário para entregar os mesmos resultados esperados aumenta (retrabalho), o que diminui a produtividade em um determinado período.

Por outro lado, duas questões suplementares surgem a partir deste tópico: A diferença do tamanho entre liberações de um produto em relação às operações de inserção, atualização e remoção ao longo do tempo pode ser mais apropriada para explicar os fenômenos que os pesquisadores pretendem observar? Existe um valor para a diferença de tamanho entre lançamentos que tendem a maximizar a produtividade do desenvolvimento de software de acordo com o produto/projeto/organização?

A compreensão de tais aspectos poderá merecer atenção em estudos futuros, embora a observação acima não possa ser refutada com base no nível de confiança obtido a partir dos estudos produzidos, pelo menos nos últimos 20 anos.

4.3.4.2. A volatilidade dos requisitos tem efeito fracamente negativo ou negativo na produtividade do desenvolvimento de software.

Apesar do número limitado de evidências (quatro) levados em conta para as sínteses neste fator, a certeza imediata (85%) e a ausência de conflito mostram que a percepção do seu efeito é notória. Este resultado demonstra que a volatilidade dos requisitos tem sido principalmente medida através do relato da percepção dos profissionais da prática, que coleta os dados de uma forma meramente qualitativa e/ou altamente dependente de pessoas. Por exemplo, Otero et *al*. (2012) usaram uma variável dicotômica para categorizar em altas ou baixas os níveis de mudanças em requisitos a partir da percepção de gerentes de projetos. Liu et *al.* (2011) utilizam quatro questões submetidas aos gerentes de projetos a serem respondidas em uma escala de *Likert* de cinco valores, de "discordo toltamente" a "concordo totalmente". Já Ramasubbu, Bharadwaj & Tayi et *al.* (2015) a medem indiretamente através do

esforço gasto com retrabalho, o que é questionável pois na prática está se assumindo que toda mudança e, consequentemente, retrabalho são, completa ou quase completamente decorrentes de uma mudança em requisitos. Todas estas variações nas maneiras de medir esse fator explicam a imprecisão da intensidade expressa e, ainda assim, obteve o segundo maior nível de confiança de todas as evidências combinadas através das sínteses.

Aqui, uma provável explicação é apresentada: há uma mentalidade compartilhada de que a Comunidade de ES observa que a volatilidade dos requisitos afeta negativamente a produtividade do desenvolvimento de software. A menos que novas evidências possam refutar esses resultados, elas podem ser empregadas nas iniciativas de melhoria. Como medir este fator quantitativamente e de forma independente da pessoa, permanece uma questão em aberto. A sugestão é que os projetos de software possam combinar informações de rastreabilidade de requisitos e práticas/sistemas de controle de versão, fornecendo alguma medida quantitativa.

Neste caso, os autores Endres & Rombach (2003) não apontam nenhuma afirmação sobre a influência da volatilidade de requisitos na produtividade do desenvolvimento de software.

A compreensão de tais aspectos poderá merecer atenção em estudos futuros, embora a conjectura acima apresentada não possa ser refutada com base no nível de confiança obtido a partir dos estudos produzidos, pelo menos nos últimos 20 anos.

4.3.4.3. Níveis mais altos de capacidade do pessoal afetam positivamente a produtividade de desenvolvimento de software.

Os estudos nesta síntese também indicam que a capacidade do pessoal tem sido mais medida através do relato de profissionais da prática, usando uma escala *Likert* de cinco pontos (KRISHNAN et *al*. 2000; FARSHCHI et *al*., 2012; Tan et *al*., 2009) de acordo com Boehm (1981). Otero et al. (2012) utiliza uma escala binária para indicar níveis altos ou baixos de capacidade do pessoal de uma equipe de acordo com a percepção de gestores de projetos. Já Munch & Armbrust (2003) utilizam a taxa de detecção de defeitos (em número de defeitos encontrados por um inspetor dividido pelo número total de defeitos conhecidos) como medida para avaliar a capacidade de inspetores em atividades de revisão.

Embora o raciocínio não seja direto, com execção do estudo de Munch & Armbrust (2003), pode-se concluir que os estudos contidos na síntese deste fator tentam observar o efeito desse fator sem caracterizar cada desenvolvedor

individualmente. Em primeiro lugar, seria difícil listar um conjunto de características capazes de operacionalizar uma medida quantitativa. Em segundo lugar, as implicações éticas podem afetar a aplicação de tal avaliação.

O resultado das sínteses e o ganho no valor da certeza sintetizada mostram que se os desenvolvedores têm as capacidades necessárias (competências, habilidades, comportamentos, entre outros) sobre como usar os recursos necessários para produzir os resultados esperados, então eles podem ser mais produtivos no desenvolvimento de software.

Embora o efeito da capacidade do pessoal na produtividade de desenvolvimento de software possa permear o *mindset* da maioria dos participantes da Comunidade de ES, a afirmação mais próxima a respeito em Endres & Rombach (2003) refere-se à Segunda *Lei de Sackman* (*"desempenho individual do desenvolvedor varia consideravelmente"*). Nossos resultados reforçam essa lei, fortalecendo evidências e, além disso, definem valores para o módulo e a direção do efeito esperado.

4.3.4.4. A experiência dos desenvolvedores no processo de desenvolvimento de software afeta fraca e positivamente a produtividade do desenvolvimento de software.

De maneira geral, os estudos que fundamentam o resultado da síntese deste fator indicam que uma equipe de projeto com desenvolvedores mais experientes nas tecnologias aplicadas no processo de desenvolvimento de software pode levar a um desenvolvimento de software ligeiramente mais produtivo do que outro com uma equipe sem essa experiência. E esta experiência geralmente refere-se à experiência média da equipe, sendo geralmente medido através de uma escala *Likert* de cinco pontos, baseado em Boehm (1981) (e.g., Nguyen et *al*., 2011, Maxwell & Forselius, 2000), ou através de períodos ou homem-horas de imersão no processo de desenvolvimento de software (e.g., Otero et *al*., 2012; Mohapatra, 2011, e; Lopez-Martinn et *al*., 2014).

Por outro lado, quando comparado com o resultado anterior, note que a capacidade pessoal de empregar seu conhecimento para resolver problemas poderia compensar parcialmente a sua falta de experiência. Isto evidencia que, seguindo a perspectiva de Boehm (1981), também encontrada em Boehm, Madachy & Steece (2000), a experiência e a capacidade dos desenvolvedores devem realmente ser tratadas independentemente como fatores pessoais distintos. Então, dado que a capacidade do pessoal média das equipes não seja muito afetada, a adição de um

desenvolvedor não experiente não deveria afetar drasticamente a produtividade do desenvolvimento de software, considerando esses dois fatores de forma exclusiva. O que, em tese, poderia ter implicações no sentido de minimizar ou atenuar os efeitos da *Lei de Brooks* (*"adicionar mão de obra a um projeto atrasado o torna mais atrasado"*). Tal conjectura não foi observada em nenhum estudo, necessitando uma investigação até mesmo da existência de uma relação de moderação ou mediação. Desta maneira, essa colocação fica no campo das hipóteses.

Nenhuma proposição sobre a experiência de um desenvolvedor foi identificada por Endres & Rombach (2003). No entanto, o número de estudos selecionados (oito) e o respectivo nível de conflito (0,0%) demonstram que tal fator tem sido amplamente estudado, sem grandes contradições. Isto pode indicar que tal fenômeno estaria possivelmente presente no *mindset* da Comunidade SE. No entanto, até onde pode-se observar, não foi explicitamente conjecturado ainda.

4.3.4.5. O conhecimento do domínio do desenvolvedor afeta positivamente a produtividade do desenvolvimento de software.

O conhecimento dos desenvolvedores sobre o problema, incluindo operações organizacionais, o funcionamento dos departamentos de usuários, experiência e habilidade administrativa e conhecimento na área de aplicação do software em desenvolvimento contribui positivamente para a produtividade do processo de desenvolvimento. A explicação é que quando os desenvolvedores se colocam no lugar de seus usuários, eles podem contribuir mais eficientemente para o desenvolvimento do software, sendo mais precisos, evitando retrabalho. Em Hsu & Hung (2013) foi medido por meio de uma escala *Likert* de cinco pontos, e medido por períodos ou pessoa-hora de imersão em atividades de desenvolvimento de software para um domínio específico nos outros estudos da respectiva síntese.

Aqui, outra proposição não vista diretamente em Endres & Rombach (2003). Entretanto, a proposição acima pode obter alguma fundamentação se explicitamente apresentada à Comunidade de ES a considerando como consequência da *Lei de Curtis* (*"bons projetos requerem profundo conhecimento do domínio de aplicação"*), ou seja, produzir o software requer um certo nível de conhecimento de domínio, independentemente sobre quão bom é o seu design. Consequentemente, a taxa na qual os projetos de software obtêm entradas e produzem saídas está relacionada ao conhecimento do domínio de seus desenvolvedores.

4.3.4.6. A autonomia da equipe afeta positivamente a produtividade do desenvolvimento de software.

Quando os projetos de software descentralizam a tomada de decisão para aqueles que executam o trabalho, está empoderando a equipe de software com autoridade e controle na tomada de decisões. De acordo com nossos resultados, isso é benéfico para a produtividade do desenvolvimento de software.

No entanto, não há medida quantitativa para capturar o grau de discrição e independência concedidos à equipe na programação, gerenciamento e controle do próprio trabalho. Esse fator geralmente é medido usando escalas *Likert*, através do relatório de profissionais da prática (Lia & Xia, 2010; Chen et *al*., 2012; Gunsel & Açikgöz, 2013). Além disso, não pudemos observar nenhuma proposta em Endres & Rombach (2003) sobre a influência da autonomia da equipe no desenvolvimento de software.

4.3.4.7. Linguagens de programação mais recentes contribuem positivamente mais para a produtividade de desenvolvimento de software do que as mais antigas.

A escolha de linguagens de programação para desenvolver o todo ou partes de um produto de software pode afetar a produtividade do software.

De maneira geral, os estudos que abordaram esse fator objetivaram comparar diferentes linguagens de programação nas quais puderam se diferenciar quanto a algumas características dos produtos de software e, assim, poder observar seus efeitos na produtividade do desenvolvimento de software. Os estudos considerados para a síntese deste fator mostram que a linguagem de programação é medida usando escalas nominais, e em alguns casos indicando uma escala ordinal indicando sua geração (3GL, 4GL, etc.).

Os resultados da síntese mostram que linguagens de programação mais novas e, possivelmente de mais alto nível, podem afetar de forma discreta e positiva a produtividade do desenvolvimento de software. O único estudo na respectiva síntese que contradiz em parte este achado é Tsunoda & Ono (2014), maior responsável pelo nível de conflito alcançado (16,9%) e afirmando que o efeito da linguagem de programação na produtividade do desenvolvimento de software pode ser indiferente em tarefas de alterações no produto relacionadas ao suporte de usuários.

De acordo com a *Lei de Corbató* (Endres & Rombach, 2003), este resultado é naturalmente esperado, considerando que as novas linguagens de programação

tentam incorporar os benefícios e remover as desvantagens das mais antigas. Como mencionado anteriormente, em conjunto com um aumento no software baseado em LOC, a adoção de linguagens de programação específicas de alto nível contribui para perceber o fenômeno por trás desta lei.

4.3.4.8. Ter a disponibilidade e estratégias para a alocação adequada de desenvolvedores afeta positivamente a produtividade do software.

O desenvolvimento de software é uma atividade intensiva altamente dependente do capital humano, a quantidade de fatores relacionados demonstra isto. Assim, sua produtividade pode ser afetada pela familiaridade dos desenvolvedores com as tarefas a serem executadas, pela disponibilidade da pessoa correta no momento certo ou quando os gerentes e líderes tomam decisões para concentrar ou dividir o esforço de pessoal.

A disponibilidade e a alocação de desenvolvedores só foram medidas em projetos por meio de um questionário no estudo de Maxwell & Forselius (2000). Nos outros estudos foi modelada com simulações (Kuchar & Vondrak, 2013) ou como problemas de otimização (KANG, Jung, & Bae, 2011; Schluter & Birkholzer, 2012), com base em informações empíricas de projetos. De fato, medir esse fator parece ser ainda um desafio. Em estudos futuros, recomenda-se avaliar a necessidade de dividir esse fator em dois para observar a alocação e a disponibilidade de desenvolvedores separadamente.

Não pudemos observar nenhuma proposição em Endres & Rombach (2003) apontando os possíveis efeitos relacionados à disponibilidade e alocação dos desenvolvedores.

4.3.4.9. A comunicação entre os desenvolvedores afeta positivamente e é necessária para a produtividade do desenvolvimento de software.

Apesar de atrasos e interrupções entre os membros da equipe de desenvolvimento frequentemente causarem atrasos no cronograma, seus efeitos não superam o efeito positivo dos desenvolvedores se comunicando ou procurando informações uns dos outros sobre a produtividade do desenvolvimento de software, como mostrado pelo resultado das sínteses.

Isto pode indicar que a comunicação deve ser vista como uma característica intrínseca do desenvolvimento de software. A questão é identificar quando a comunicação não é produtiva, ou seja, não está relacionada a reuniões de design, a

resolução de problemas dentro da equipe, abordando mal-entendidos em especificações e outros problemas.

Com três estudos, a síntese deste fator apresentou nível de conflito 0,0%, indicando a ausência completa de contradições entre evidências e permitindo que um ganho de certeza de 36% em relação a evidência com maior certeza (Avritzer & Lima 2009, com certeza imediata de 34,3%). Isto demonstra como a *Teoria de Dempster-Shafer* é capaz de aumentar a confiança que podemos ter em um resultado a partir de uma combinação de evidências que não se contradizem ou se contradizem pouco.

Em relação a Endres & Rombach (2003), vislumbra-se que as proposições mais diretamente relacionadas, levando em conta os aspectos de comunicação no desenvolvimento de software, são a *Lei de Conway* (*"um sistema reflete a estrutura organizacional que o construiu"*) e a Primeira *Hipótese de Booch* (*"Modelos de objetos reduz problemas de comunicação entre analistas e usuários"*). No entanto, o entendimento se há ou quais são as implicações dessas declarações sobre o desenvolvimento de software e sua produtividade não é direta.

4.3.4.10. Níveis mais altos de maturidade dos processos de software podem contribuir mais efetivamente para a produtividade do desenvolvimento de software do que os mais baixos.

Normalmente, modelos de melhoria de processos de software, como CMMI-DEV ou MPS-SW (Santos *et al.*, 2010; Weber *et al.*, 2005), defendem ganhos específicos de produtividade na produção e qualidade de produto quando as organizações decidem adotá-los. De acordo com nossos resultados, é verdade.

De alguma forma, manter um certo nível de atividades estruturadas e adotar um conjunto de práticas e controles em toda a produção de artefatos é benéfico para o desenvolvimento de software. Os resultados das sínteses mostram que quanto mais as práticas e o controle abrangem diferentes áreas de processo, mais projetos podem se beneficiar da melhoria de sua produtividade durante o desenvolvimento do software. É a explicação que pode ser oferecida a partir dos resultados produzidos.

Do ponto de vista das sínteses, nota-se que os resultados apresentam 5,2% de nível de conflito, o que indica que mesmo tendo ganho em valores de certeza ao final (11%), existe algum nível de discordância entre as evidências relativas a esse fator. Todas as evidências da síntese deste fator apresentaram algum efeito positivo em maior ou menor grau, com exceção do estudo de Ramasubbu, Bharadwaj & Tayi (2015) que relata um efeito indiferente do nível de maturidade na produtividade do

desenvolvimento de software. Assim, esse estudo é quem mais contribuiu para o nível de conflito alcançado.

De qualquer forma, o efeito imediato esperado de curto prazo mais provável é um efeito positivo na produtividade do desenvolvimento de software. Estes achados reforçam a *Lei de Humphrey* (*"processos maduros e disciplina pessoal aprimoram o planejamento, aumentam a produtividade e reduzem os erros"*).

4.3.4.11. Quanto mais dinâmica e sofisticada é a evolução da atividade econômica (área de negócios), menor é a produtividade do desenvolvimento de software.

Alguns setores industriais ou atividades econômicas são mais propensos a pressões externas do que outros. Áreas de negócios específicas devem lidar com requisitos complexos de qualidade ou com a pressão por evolução constante ou rápida de seus produtos. Por exemplo, desenvolver softwares para a área de Telecomunicações exige regras menos flexíveis e maiores pressões de mercado do que para Finanças. Em geral, desenvolver softwares para o governo abrange leis e questões administrativas, que não impõem alta pressão para a evolução contínua dos produtos. De alguma forma, a maneira de como cada setor deve lidar com suas regras e pressões compromete a capacidade de ser mais produtivo quando as organizações estão desenvolvendo software.

De acordo com a síntese deste fator, a maior parte das evidências mostraram um sutil efeito das áreas de negócio na produtividade do desenvolvimento de software (Myrtveit & Stensrud, 1999, He et *al*., 2008, Wang, Wang & Zhang, 2008; Tsunoda et *al*., 2009). Somente em Tsunoda & Ono (2014) foi demonstrado um efeito negativamente mais contundente na produtividade do desenvolvimento de software.

As sínteses estruturadas não podem explicar quais características, como requisitos de segurança e/ou confiabilidade, facilidade de entendimento do domínio, alta volatilidade das regras de negócios e outras questões, impactam negativamente a produtividade do desenvolvimento de software. Embora não tenha sido abordado, esse fator parece estar relacionado à taxa na qual os requisitos mudam, ou seja, a volatilidade dos requisitos.

Assim, os estudos em andamento devem responder melhor a novas questões sobre quais características das áreas de negócio estão afetando a produtividade do desenvolvimento de software ou outra variável dependente sob observação. Isso exige que a área de negócios não seja medida exclusivamente por uma escala ordinal, como

tem acontecido nos estudos. Para tal, medidas diretas ou indiretas precisam ser desenvolvidas.

4.3.4.12. A manipulação de artefatos complexos ao longo do desenvolvimento de software diminui levemente ou não afeta sua produtividade.

Nos estudos selecionados, esse fator foi medido pela percepção de profissionais (Rothenberger et al., 2010) ou pela média da complexidade ciclomática das funções de software (Colazo, 2008; Rothenberger et *al*., 2010; Mohapatra, 2011). Mesmo assim, as sínteses estruturadas puderam alcançar um ganho significativo (28%) no valor da certeza ao observar o impacto da complexidade do artefato na produtividade do desenvolvimento de software, embora haja algum nível de discordância entre os resultados individuais dos estudos em termos de efeitos observáveis (conflito de 6%).

Portanto, este resultado mostra que, se os gerentes de projeto consideram que a complexidade média dos artefatos produzidos durante o desenvolvimento de software é uma característica inerente de um produto de software, eles devem considerar medir ou controlá-lo visando manter ou melhorar níveis específicos de produtividade do desenvolvimento de software. No entanto, não se descarta a necessidade de mais estudos para definir quais produtos podem mais se beneficiar ao avaliar o impacto da complexidade de artefatos e discutir as medidas apropriadas para capturar o fenômeno a ser observado.

Aqui, a única declaração relacionada à complexidade de artefatos em Endres & Rombach (2003) é a *Hipótese de McCabe* (*"métricas de complexidade são bons preditores de confiabilidade e manutenibilidade pós-lançamento"*). No entanto, as implicações desta hipótese no desenvolvimento de software e sua produtividade, via efeitos de confiabilidade, manutenibilidade ou não, é um desafio a ser conjecturado dentro deste contexto.

4.3.4.13. O aumento no nível de reuso de requisitos, análise e *design* promove ganhos na produtividade do desenvolvimento de software.

A ênfase na análise e *design* flexíveis visa a definição de estruturas (componentes, arquiteturas, catálogos de requisitos, padrões, dentre outros) que podem permitir atingir níveis desejáveis de modularidade, coesão, e outras características, e facilitar a reutilização de tais estruturas em diferentes soluções de software.

Os resultados das sínteses apontaram um baixo ganho (3%) no valor da certeza, indicando que a confiança obtida a partir das sínteses estruturadas é praticamente igual ao melhor valor de certeza dentro do conjunto de estudos. Apesar de todos os estudos considerados na síntese demonstrarem efeito positivo do reuso na produtividade do desenvolvimento de software, em maior ou menor grau, existe também uma discordância significativa entre os estudos individuais (6,9%). Isso pode ser explicado, em parte, porque algumas medidas relacionadas ao produto (como complexidade, modularidade, entre outras) usadas nesses estudos podem ser vistas como *proxies* ou variáveis indiretas para o fenômeno. Assim, mesmo que este conhecimento já possa ser colocado em prática, a academia ainda deveria reforçar esse resultado.

No entanto, o efeito mais provável e esperado da flexibilidade de reuso na produtividade do desenvolvimento de software é relevante e positivo. Nesse sentido, os resultados das sínteses corroboram parte da *Lei de McIlroy* (*"a reutilização de software reduz o tempo de implantação e aumenta a produtividade e a qualidade"*). Além disso, nossos resultados destacam que o uso de ferramentas modera positivamente a influência do reuso na produtividade do desenvolvimento de software, ou seja, o nível de sofisticação dos mecanismos fornecidos pelas ferramentas limita o ganho que pode ser obtido do nível pretendido de reuso.

4.3.4.14. Combinar o uso de métodos e técnicas de verificação, validação, teste ou qualquer outro relacionado à garantia de qualidade tem efeitos positivos na produtividade do desenvolvimento de software.

A declaração acima afirma que níveis mais altos de produtividade podem ser alcançados através da implementação e combinação de práticas relacionadas à qualidade no processo de desenvolvimento de software. As sínteses estruturadas mostram que combinar práticas de qualidade, como inspeções, verificação, validação, teste, gerenciamento de configuração e assim por diante, é benéfico para melhorar a produtividade do desenvolvimento de software. Isso mostra que os resultados das sínteses poderiam ajudar a fortalecer a *Lei de Hetzel-Myers* (*"a combinação de diferentes métodos de V&V supera qualquer método único"*), de acordo com Endres & Rombach (2003), se a combinação de métodos V&V for observada em termos do desempenho do processo de desenvolvimento e da sua produtividade.

De qualquer forma, embora não tenha sido capturado por estudos específicos nas revisões executadas, também não pode ser ignorado que pode haver possíveis implicações ao se aplicar um método único no processo de desenvolvimento de

software. *A Lei de Fagan* (*"inspeções aumentam significativamente a produtividade, a qualidade e a estabilidade do projeto"*) e a *Lei de Basili* (*"inspeções baseadas em perspectiva são (altamente) eficazes e eficientes"*) endereçam a questão quanto ao emprego de inspeções, mas o mesmo não pode ainda ser atribuído a testes ou outras práticas de garantia de qualidade. O que merece atenção e estudos futuros.

No entanto, esse resultado merece atenção. Na Tabela 7, observe que não há conflito entre os resultados dos estudos selecionados, e há um ganho de confiança (20%). Isso significa que as intensidades descritas dos resultados ao observar este fenômeno se interceptam, mesmo havendo alguma diferença, e os valores de certezas de cada estudo foram usados para re-ponderar o novo valor de certeza da evidência combinada. Isto enfatiza a necessidade potencial de discussão sobre maneiras apropriadas de medir o efeito desse fator na produtividade do desenvolvimento de software e realizar mais estudos para reforçar ou refutar esse resultado. No entanto, ele poderia ser colocado em prática de acordo com os critérios propostos neste trabalho.

4.3.4.15. Aumentar o número de desenvolvedores ao longo do tempo tem um efeito ligeiramente adverso na produtividade do desenvolvimento de software.

Todos os estudos contidos na síntese deste fator mostram que um aumento no número de desenvolvedores e sua variação ao longo do tempo afetam negativamente a produtividade do desenvolvimento de software, em maior ou menor grau. Além disso, ao contrário do que a *Lei de Brooks* (*"acrescentar mão-de-obra a um projeto atrasado o torna mais atrasado"*) defende, aparentemente acrescentar alguns desenvolvedores no desenvolvimento de software afeta projetos atrasados ou não, a qualquer momento.

De acordo com os resultados das sínteses, os estudos selecionados para esse fator não podem ser usados para fortalecer unicamente a *Lei de Brooks*, mas eles não podem refutar tal lei. Isso significa que, aparentemente, a *Lei de Brooks* pode ser generalizada para ser declarada como "acrescentar mão de obra a um projeto torna-o mais atrasado". Tal generalização não pretende responder a situações específicas, mas visa representar um ponto de vista unificado e de curto prazo do efeito esperado em relação ao fenômeno. Assim, se a curva de aprendizado dos desenvolvedores, o número de canais de comunicação, fatores sociais ou psicológicos estão por trás do efeito observado, sua duração ou desvios transcendem a discussão proposta para este trabalho, mas são questões reconhecidamente abertas, merecendo novas investigações.

Além disso, não podemos afirmar que “remover mão de obra de um projeto melhora/piora sua produtividade”, ponto que também merece atenção em estudos futuros.

4.3.5. Ameaças à Validade

A principal ameaça à validade desta etapa de metodologia de pesquisa é o viés dos pesquisadores, que foi minimizado por meio de avaliações por pares, onde os pesquisadores confrontam o planejamento aos resultados das sínteses.

Obviamente, há o risco de interpretar erroneamente a certeza e os efeitos relatados pelos estudos primários. O SSM minimiza-o ao impedir que os pesquisadores atribuam os valores iniciais de certeza dos estudos primários, usando questionários para cada tipo de estudo experimental (experimentos, estudos longitudinais, dentre outros) de acordo com a hierarquia de evidências GRADE. Quanto às intensidades reportadas nas representações de cada estudo primário, essas foram capturadas com base nos relatos, dados e análises encontradas nas próprias evidências, com intuito de minimizar o risco associado ao víes dos pesquisadores durante as tarefas de representação nas sínteses com SSM. No entanto, nenhuma avaliação com terceira parte foi realizada para verificar o nível de concordância na atribuição das intensidades ou das caracterizações de qualidade usadas na atribuição da certeza inicial de cada estudo primário.

Por fim, a abordagem para a síntese de evidências, SSM, não foi ampla e sistematicamente avaliada, tornando difícil estimar sua precisão, embora tanto o método de síntese quanto a ferramenta que o operacionaliza têm sido utilizados com sucesso e aceito em diferentes fóruns de publicações (dos Santos & Travassos, 2017; dos Santos *et al.*, 2018; Martinez-Fernandez *et al.*, 2015).

4.4. Considerações em Relação a Outras Iniciativas

Há iniciativas, do ponto de vista metodológico, com diferentes propostas de construção de proposições teóricas (Johnson & Ekstedt, 2007; Johnson *et al.*, 2012; Mannaert, Verelst, & Ven, 2008; Shull & Feldmann, 2008; Sjøberg *et al.*, 2008).

Quanto às propostas de construção/captura de proposições teóricas em ES, podemos identificar algumas iniciativas recentes que discutem relações estritamente sob a ótica de alguns poucos fatores, de maneira isolada, considerando aspectos, técnicos, sociais, econômicos ou interpessoais de ES (Bjarnason, Smolander, Engström, & Runeson, 2016; Dittrich, 2016; Erbas & Erbas, 2015; Munir, Runeson, & Wnuk, 2018; Ralph, 2016; Scholtes *et al.*, 2016). Apesar de algumas destas levarem

em conta proposições de pesquisas anteriores, tais como a *Lei de Brooks*, ou estarem pautadas em observações em estudos primários em contextos restritos, nenhuma ligação entre as proposições propostas neste estudos e produtividade de software foi explicitamente observado.

Mesmo observando isoladamente as proposições apresentadas em Endres & Rombach (2003) e os estudos primários e secundários encontrados na literatura técnica identificada nesta tese, não foi possível identificar nenhum conjunto de proposições lidando com tantos fatores diferentes. Trabalhar com este número de fatores é, a princípio, evidência que reforça a *Lei de Nelson-Jones* (*"uma multiplicidade de fatores afeta a produtividade"*).

Nenhuma das iniciativas aqui apresentadas também contempla a construção de proposições baseados na agregação de evidências experimentalmente obtidas de diferentes iniciativas, através de estudos primários executados por diferentes pesquisadores em diferentes contextos dentro de um tema específico. Algo que Petersen (2011) em seu protocolo já havia tentado observar (identificar um modelo geral nas evidências de sua revisão), mas relata não ter tido sucesso.

Já Shull & Feldmann (2008) consideram, dada a natureza das restrições dos estudos primários em ES (representatividade amostral, heterogeneidade de operacionalização de variáveis, falta de repetição, dentre outras), que é imprescindível a formulação de teorias a partir de dados e evidências experimentais. Esta tese define um conjunto de passos que endereçam esta questão sob uma perspectiva da ES baseada em evidência.

O método de pesquisa utilizado representou estas evidências utilizando estruturas teóricas, para quantificá-las e agregá-las para obter um conhecimento generalizável sobre os relacionamentos entre estes fatores e a produtividade de software, algo também nunca tentado antes e que pode ajudar a uniformizar o conhecimento entre pesquisadores e formar um consenso inicial necessário para estabelecer uma agenda de pesquisa. Além disso, a captura deste conhecimento está, desde do início, mapeada em um protocolo de revisão sistemática e seus resultados devidamente explicitados.

As expectativas de ganho para a ES como ciência, são que, uma vez avaliado tal modelo, este poderá abrir uma série de novas possibilidades de estudos que poderão ser utilizados para confirmar ou evoluir o conhecimento como o temos atualmente. A partir dos resultados alcançados ao final desta tese, novas hipóteses podem ser exploradas, leis confirmadas, e teorias formuladas. Assim, o modelo poderá

ser reavaliado, ter medidas e fatores incluídos ou revistos, e ter o entendimento sobre fenômenos melhores explicados. Neste sentido, o próximo capítulo apresenta a avaliação do modelo realizada com intuito de compará-lo às opiniões de gestores e líderes de projetos em organizações brasileiras.

O modelo proposto também poderia ser utilizado em iniciativas de interação entre academia e indústria para uniformizar o vocabulário e homogeneizar o conhecimento entre as partes através de conceitos gerais, quando estiverem tratando de questões de transferência e avaliação de tecnologias em contextos organizacionais no tema.

4.5. Implicações da Disponibilização do Conhecimento

4.5.1. Na Operacionalização dos Fatores

Dado o conjunto de proposições obtidos, um grande desafio, talvez o maior, seja definir o meio de operacionalizar os fatores, ou seja, definir medidas apropriadas para medir os fatores. Partimos do pressuposto que temos um corpo de conhecimento capaz de nos fornecer indícios a este respeito. A única questão ainda a considerar é se pesquisadores de ES têm a compreensão necessária sobre o significado das medidas utilizadas para operacionalizar tais construtos. Da mesma forma, há ainda a necessidade de endereçar a própria definição de medida de produtividade de software. Por exemplo, a questão de utilizar LOC ou variações como medida de saída não está clara. Há paradoxos a se considerar (Aquino & Meira, 2009; de Barros Sampaio *et al.*, 2010; Adrián Hernández-López *et al.*, 2015).

Uma outra opção talvez seja fazer a operacionalização através de uma tradução dos efeitos observados nas agregações em algum tipo de função que possa ser usada em um modelo de simulação, verificar o comportamento médio das interações entre variáveis e assim fornecer algo que estaria mais próximo de um modelo de percepção. De qualquer forma, esta é uma questão também em aberto e não será endereçada nesta pesquisa.

4.5.2. Na Indústria

No curto prazo, a indústria poderá utilizar o modelo obtido para priorizar aqueles fatores que são possíveis de serem medidos em suas iniciativas de melhoria ou de medição de processos. Assim, gestores poderão comparar os resultados de medições em seus processos com os achados do modelo proposto. Se houver uma discrepância entre as visões fornecidas entre eles será possível identificar e caracterizar causas bem como avaliar possibilidades de melhorias, inclusive uma

revisão no próprio processo de medição. Por exemplo, na revisão sistemática executada foi visto que Controle Estatístico de Processos tem sido pouco empregado para avaliar a produtividade em processos de software. Uma hipótese é que talvez esta observação esteja relacionada a dificuldade de correlacionar desvios no comportamento do processo de desenvolvimento de software com fatos observáveis e que deveriam, em tese, levar a uma avaliação da necessidade de uma ação corretiva ou preventiva. Mensurar alguns dos fatores encontrados pode ajudar na observação e auxiliar na correlação de fatos, não trivial ou óbvia devido à natureza intelectual, criativa e intensivamente humana dos processos de software. Ao medir produtividade e um conjunto de fatores selecionados, seria possível tentar identificar e analisar causas e, em um momento posterior, predizer comportamentos. Isto forneceria um caminho factível de alcançar de modo incremental maiores níveis de maturidade dentro de uma organização de software.

Em outras abordagens de medição, tais como GQM (Basili, Caldiera, & Rombach, 2002), medição baseada em valor (Boehm, 2003), ou GQM+Strategies (Basili *et al.*, 2009), todo o conhecimento fornecido (sob a forma de proposições, fatores, medidas e análises subsequentes) poderá ser utilizado para orientar na definição de objetivos, questões e medidas relacionadas à produtividade de software e fatores de interesse de diferentes stakeholders de acordo com os objetivos de projetos ou organizações.

Para a indústria, o modelo proposto pode fornecer meios de priorizar a observação de certos fatores em processos de software pelo modo como materializa a incerteza associada a fenômenos. Por exemplo, se um fator com mais de 65% de confiança de um efeito indiferente na produtividade do desenvolvimento software, por que investir tempo e recursos medindo ou gerenciando-o? Isto já um indicativo mais direto e prático para gestores do que o conhecimento fornecido atualmente, estando completamente baseado em evidências.

Já na questão da operacionalização de fatores, gerentes e líderes de projetos podem medi-los e realizar *benchmarks* entre projetos em suas organizações. Com isso, poderão ver como diferentes práticas estão levando às variações nos fatores e na produtividade em diferentes projetos.

Assim, ao final desta tese, espera-se que o conjunto final de proposições forneçam à indústria os seguintes benefícios: (1) um conjunto de fatores a serem levados em conta e, possivelmente, priorizados em iniciativas de medição de produtividade em projetos e processos de software; (2) um conjunto de fatores onde, a

princípio, a incerteza sobre o efeito esperado é muito grande e cujo esforços de medição devam ser cautelosamente avaliados; (3) um conjunto de medidas encontradas para mensuração dos fatores identificados nesta pesquisa; (4) uma discussão inicial sobre o modo como tais fatores e medidas poderão ser incorporados ou adaptados aos diferentes métodos de medição empregados nas organizações, e; (5) por fim, um conjuntos de assertivas genéricas capazes de sintetizar o conhecimento baseado em evidências e expressos em termos de conceitos gerais, que permitam a gestores e líderes de projetos refletir sobre que decisões tomar diante de situações adversas ou que melhorias buscar em seus processos de software.

Por fim, temos de destacar que, quando observados na prática, dificilmente os resultados das sínteses dos fatores poderão ser observados de forma isolada, ou seja, sem a possível interferência de outros fatores. Por exemplo, não faria sentido observar o efeito do aumento do tamanho da equipe sem observar as experiências no processo de desenvolvimento, as capacidades pessoais em resolver problemas ou o conhecimento prévio do domínio dos desenvolvedores introduzidos no projeto. Como também não faria sentido deixar de observar se a influência da autonomia da equipe não estaria sendo anulada ou diminuída pela influência da dispersão da equipe na produtividade do desenvolvimento de software. Então, os resultados da influência dos fatores devem ser pensados e usados da forma mais ampla e holística possível, a fim de se obter uma compreensão mais exata de suas capacidades de predizer seus respectivos fênomenos.

4.5.3. Na Academia

Para auxiliar no *gap* entre academia e indústria, a longo prazo o conhecimento a ser ofertado poderá ser utilizado para que pesquisadores possam refletir sobre a necessidade de melhorias em métodos e ferramentas já difundidas, ou para o desenvolvimento de novas.

A disponibilização dos fatores, suas relações e chances de ocorrências (de certezas) poderão fornecer à comunidade científica *background* com informações importantes sobre o que se sabe ou não com alguma confiança sobre produtividade do desenvolvimento de software.

4.6. Conclusão

Este capítulo apresentou o método de síntese e o *framework* teórico obtido a partir de fatores previamente identificados através de estudo secundário. Os resultados obtidos representam de forma direta e concisa, um *snapshot* da pesquisa experimental dos últimos 30 anos de acordo com um protocolo bem definido de seleção de estudos.

Tais resultados, proposições expressas na forma de um modelo, são compostos de 25 fatores e 34 relacionamentos entre fatores e a produtividade do desenvolvimento. A importância destes números é a amplitude da cobertura que representa, já que nenhum estudo primário ou secundário identificado contempla tantos fatores e ainda expressa os fenômenos em termos de efeitos esperados. Como esses resultados se interceptam com outras proposições com relativa aceitação dentro da Comunidade de ES também auxilia a reforçar alguns *insights* já conhecidos e dão uma visão mais holística de possíveis interações. Além disso, a aplicação do conhecimento capturado é discutida tanto na academia quanto na indústria.

Tal modelo poderá ajudar na uniformização do conhecimento e discussões realizadas por pesquisadores, diminuindo o *gap* conceitual entre acadêmicos e profissionais da prática sobre o tema produtividade do desenvolvimento de software. Além disso, com base nos resultados, será possível a realização de estudos ou a criação/aperfeiçoamento de ferramentas ou métodos que meçam determinados fatores ao longo da execução das atividades de processos de software e, então, utilizar uma abordagem “muito mais engenharia e menos ciência” de desenvolver tecnologia.

Porém, um resultado que se propõe a ser ponto de partida no campo teórico só alcançará tal objetivo se conseguir demonstrar que, mesmo em circunstâncias específicas, é capaz de refletir as situações vividas pelas pessoas e organizações desenvolvendo software. Assim, no próximo capítulo será abordado um estudo que visa avaliar o nível de convergência ou concordância entre a pesquisa (o modelo obtido) e a prática (percepção dos profissionais).

Capítulo 5: Avaliação Experimental dos Resultados

Neste capítulo está descrita a etapa de avaliação experimental dos resultados, que compara o modelo encontrado com a opinião dos profissionais que desenvolvem software em organizações brasileiras.

5.1. Considerações Iniciais

Apesar do fato de um aumento no número de estudos experimentais na literatura técnica nos últimos 30 anos, os pesquisadores de ES têm enfrentado problemas para colocar em prática os conhecimentos adquiridos (Hannay *et al.*, 2007; Sjøberg et al., 2008 Hernández-López *et al.*, 2015; Bjarnason *et al.*, 2016). Por um lado, porque os pesquisadores de ES têm falhado em perceber as lacunas existentes entre a pesquisa experimental e as práticas, e suas consequências na compreensão da produtividade do desenvolvimento de software (Ivanov *et al.*, 2017). Por outro, porque não está claro como a produtividade pode ser observada quando o produto é software (Fritz *et al.*, 2017).

De acordo com Endres & Rombach (2003), a principal forma de avaliar resultados no campo teórico é através de consenso e resultados amplamente aceitos por um grupo de indivíduos. Para alcançar tal objetivo, independentemente da estratégia adotada, uma avaliação de resultados como o obtido nesta tese tem o intuito de observar a convergência e, consequentemente, possíveis *gaps* entre a academia e indústria. O que pode apontar novas questões e oportunidades, além daquelas previamente ilustradas no capítulo anterior. Desta forma, acreditamos que com os estudos realizados podemos sensibilizar os interessados sobre a relevância dos resultados encontrados. Algumas das opções de avaliação que podem ser utilizadas para avaliação do modelo são:

- Comparar os resultados com dados de projetos: esta forma de avaliação poderia ser enquadrada como uma forma de prova, se fosse trivial obter uma amostra representativa de dados de projetos de software medindo tantos fatores com medidas operacionalizadas. Sem dúvida, esta seria a estratégia mais ambiciosa de ser adotada.
- *Survey* com especialistas da indústria: O grande problema nesta abordagem é selecionar uma amostra realmente representativa em tamanho (número de participantes) ou expressividade (participantes

que, além de interessados, tenham experiência e uma perspectiva relevante em relação ao problema).

- Revisão por pares acadêmicos com notório saber no tema: esta avaliação seria um meio de estabelecer uma uniformização de conhecimento e consenso inicial em um grupo específico de interessados no tema. Também tornaria possível ajustes no modelo proposto de modo mais minucioso e preciso, embora sob uma perspectiva acadêmica.

Como ponto de partida para abordar tais questões, este capítulo propõe como abordagem capturar as percepções dos profissionais brasileiros sobre os efeitos de fatores na produtividade do desenvolvimento de software, de acordo com os resultados do *framework* teórico obtido, para, então, compará-los entre si.

A comparação entre estes achados visa verificar se é possível reforçar a evidência sobre a influência de tais fatores na produtividade do desenvolvimento de software, através da indicação de qual o grau de concordância entre os pontos de vista da academia e da indústria. E mais, pode indicar que a pesquisa pelo menos tem tido relativo sucesso e capacidade em identificar e estudar a medição de fatores e a produtividade no contexto do desenvolvimento de software ao longo dos anos, mesmo com todas as limitações e a intangibilidade inerente ao software.

Deste modo, abordaremos a definição, planejamento, execução e análise de resultados de um estudo primário, um *survey* na web, para capturar as percepções de gerentes e líderes de projeto de software da indústria brasileira sobre fatores que afetam a produtividade em projetos de software.

Surveys têm sido frequentemente usados em ES, recebendo destaque deste tipo de estudo nos últimos anos. São estudos que permitem aos pesquisadores coletar dados de uma população-alvo com o objetivo de ter resultados gerais, representativos dessa população (Punter, Ciolkowski, Freimut, & John, 2003).

Molleri *et al*. (2016) identificam etapas e diretrizes gerais de pesquisa para este tipo de estudo, constituído dos seguintes passos: (1) Definição de Objetivos; (2) Identificação de público alvo e frame amostral; (3) Plano de Amostragem; (4) Instrumentação do *Survey*; (5) *Trials* de Avaliação; (6) Coleta e Análise de Dados; (7) Conclusões e Relatos. Nas subseções dentro deste capítulo, as informações sobre o *survey* executado serão discutidas de acordo com estes passos.

5.2. Método de Avaliação: *Survey* na Indústria

5.2.1. Objetivos e Questões de Pesquisa

De acordo com (Wohlin *et al*., 2012), o *survey* deste estudo é categorizado como uma pesquisa exploratória, ou seja, se propõe a investigar relações, no caso, causais. E por este motivo, representa uma escolha natural para capturar as percepções sobre a influência dos fatores de desenvolvimento de software na produtividade sob o ponto de vista dos profissionais.

Assim, o objetivo deste estudo pode ser declarado, de acordo com o GQM (Basili, 1992), da seguinte forma:

> *"**Analisar** fatores de desenvolvimento de software*
> ***com a finalidade de** caracterizar*
> ***com respeito à** concordância de suas influências na produtividade do desenvolvimento de software em relação a evidências experimentais adquiridas na literatura técnica*
> ***do ponto de vista de** engenheiros de software*
> ***no contexto de** gerentes e líderes de projetos de software em organizações que desenvolvem software no Brasil".*

5.2.2. Identificação da população-alvo e frame amostral

O público-alvo de um *survey* descreve a totalidade ou um subconjunto que representa demograficamente uma população. Já uma amostra é selecionada de acordo com um método de seleção ou prospecção, capaz de representar esta população.

Nesta avaliação, como os fatores estão relacionados às características contemplando diferentes aspectos do desenvolvimento de software, o público-alvo do *survey* são aqueles envolvidos com o desenvolvimento de software e capazes de perceber os impactos dos diferentes fatores nos projetos de software, ou seja, engenheiros de software que têm uma visão mais holística do processo de desenvolvimento, os gerentes e líderes de projeto.

A prospecção dos respondentes (amostragem) neste estudo tem o objetivo de obter amostra(s) de participantes somente no Brasil. Assim, os instrumentos foram escritos em português.

5.2.3. Plano de Amostragem

Os participantes foram selecionados de acordo com o acesso e vínculo a redes de negócios ou associações de organizações de software no Brasil, ou seja, uma

amostragem por conveniência. Os participantes receberam o convite por *e-mail* e por meio de postagens no *LinkedIn*. Portanto, não foi possível, nem o intuito, controlar ou identificar o tamanho da população ou os respondentes.

5.2.4. Instrumentação do Survey

O tipo de análise que pode ser feita com os resultados de um *survey* depende diretamente de como foi projetado. As perguntas devem refletir o ponto de vista dos respondentes, sejam abertas e/ou fechadas, de acordo com os tipos de respostas disponíveis (por exemplo, baseadas em escalas *Likert*) (Kasunic, 2005).

Questão abertas são aquelas que, em geral, exigem mais reflexão dos respondentes e dificilmente são expressas através de uma simples resposta, frase ou um conjunto de palavras pré-definidas.

Questões fechadas são aquelas que podem ser respondidas através de um conjunto restrito de valores, pré-definidos, que podem ser números, um intervalo, frases simples ou simples palavras, como "sim" ou "não".

A atenção a certos detalhes no design de um *survey* é importante, pois quando um *survey* não é claramente entendido, os respondentes chegam a conclusões erradas sobre as perguntas e, como resultado, respondem incorretamente (Torchiano & Ricca, 2013). Isso pode levar ao abandono ou o preenchimento incompleto das respostas, obscurecer fatos, tendências, impossibilitar a análise ou até mesmo invalidar resultados do estudo.

Como a intenção do estudo é ter uma indicação da concordância entre pesquisa e prática, além de minimizar o risco de desistência do preenchimento ou evasão de perguntas ao longo do preenchimento, outro cuidado tomado foi limitar o número de perguntas para, consequentemente, diminuir o tempo total de preenchimento do questionário. Além disso, nenhuma pergunta precisava ser obrigatoriamente respondida, para não tornar a tarefa de preenchimento maçante ou intimidante.

Somente fenômenos de influência dos fatores identificados na literatura técnica com maiores níveis de confiança (a segunda metade do *framework* teórico) foram levados em consideração nas formulações das perguntas. Tal decisão leva em conta que a medição dos fenômenos relacionados aos outros fatores inerentes a produtividade do desenvolvimento de software ainda é incipiente. Desta forma, incluí-los na comparação poderia aumentar a imprecisão ou erro decorrente do pouco conhecimento que se tem sobre medi-los, bem como a ausência de indícios de se os

próprios participantes conseguem percebê-los ou expressar os efeitos que se deseja comparar. Em outras palavras, a ideia é entender se a pesquisa está apta a se aproximar do que é vivenciado na prática quando se propõe a estudar fatores, logo fatores com baixos níveis de confiança não podem contribuir para se encontrar uma resposta.

A lista de fatores com maiores níveis de confiança e o público-alvo influenciaram o projeto e a opção de executar um *survey on-line*, disponibilizado na web, que foi estruturalmente organizado para conter as questões intencionalmente ordenadas da seguinte forma: (1) fatores de desenvolvimento de software (17 questões); (2) características dos participantes (por exemplo, nível educacional, experiência de gestão/liderança, com três questões) e; (3) características da organização (por exemplo, área de negócios, número de empregados, tipo de produtos, com três questões).

A intenção com a definição de uma ordem de apresentação das questões é minimizar a chance de não ter todas as questões relacionadas aos fatores respondidas, em caso de abandono do preenchimento por parte do participante.

Neste *survey*, as perguntas foram projetadas como fechadas. Sendo que aquelas relativas aos fatores foram expressas seguindo o *rationale* da Tabela 8, de acordo com os *templates* de descrição dos códigos abordados no Capítulo 3 (APÊNDICE V). Foram utilizadas Escalas Analógicas Visuais (*Visual Analogue Scale* - VAS) para a captura das respostas. Uma escala VAS permite que os participantes expressem valores subjetivos, indicando uma posição ao longo de uma linha, que pode ser traduzida em alguma escala intervalar ou razão para analisar as medidas. Os dois pontos finais de uma linha devem representar semanticamente os valores extremos da escala e um ponto na linha pode representar um valor neutro. Além disso, estudos mostram que uma VAS pode superar escalas discretas em relação às características métricas e permite a aplicação de mais métodos estatísticos (Bourdel *et al.*, 2015; Funke & Reips, 2012; Klimek *et al.*, 2017; Price, Staud, & Robinson, 2012; Reips & Funke, 2008).

Escalas VAS podem ser utilizadas no lugar de escalas *Likert* para mensurar valores estritamente qualitativos, por exemplo, “Concordo plenamente” a “Discordo plenamente”; “Insignificante” a “Totalmente relevante”; dentre outros. Note que, o significado da resposta está sempre relacionado a um ponto referencial na escala, possivelmente o valor neutro ou unitário. O uso deste tipo de escala tem sido avaliado em estudos para quantificar a dor (Bourdel *et al.*, 2015), reabilitação (Price *et al.*, 2012)

e sintomas de doenças crônicas (Klimek *et al.*, 2017). Exemplos da aplicação de VAS em ES pode ser vista em Gonçalves (2017) ou Guidini Gonçalves *et al.* (2018).

Tabela 8. *Rationale* da elaboração das questões referentes aos fatores

Fatores de Desenvolvimento de Software	***Rationale*: Qual é o efeito esperado << descrição >> sobre produtividade do desenvolvimento de software?**
Comunicação entre Desenvolvedores	da comunicação entre desenvolvedores
Área de Negócios	da dinâmica e pressões externas de uma área de negócios
Volatilidade de Requisitos	da volatilidade de requisitos
Duração do Projeto	do incremento na duração de um projeto
Tamanho de Software baseado em LOC	de ter aumentos significantes na quantidade de linhas de código ao longo do tempo
Nível de Maturidade (Capacidade)	de evoluir de níveis inferiores para superiores de maturidade em modelos como CMMI-DEV e MPS-SW
Metodologia do Desenvolvimento de Software	do uso de um desenvolvimento orientado a guias e bem formatados (cascata, modelo V, etc.) ao invés de processos de desenvolvimento flexíveis e menos estruturados (ágeis, cíclicos, espiral, etc.).
Experiência dos Desenvolvedores no Desenvolvimento de Software	da experiência de desenvolvedores em diferentes tecnologias, métodos, técnicas e ferramentas
Capacidade do Pessoal	da habilidade de desenvolvedores em cooperar, informar e produzir resultados para alcançar os objetivos de projeto
Complexidade de Artefatos	do aumento na complexidade de artefatos de software
Disponibilidade e Alocação de Recursos Humanos	da disponibilidade e correta alocação de recursos humanos
Conhecimento de Domínio dos Desenvolvedores	do conhecimento de domínio dos desenvolvedores sobre o problema
Linguagem de Programação	de usar linguagens de programação mais novas quando comparadas com as antigas
Implementação de Qualidade de Software	da combinação de diferentes práticas de qualidade de software (e.g., testes, inspeções, gerência de configuração)

Autonomia da Equipe	de equipes autonomamente planejarem, definirem e alocarem seu próprio trabalho
Flexibilidade de Reuso	do reuso de requisitos, análise e projeto
Uso de Ferramentas	de usar ferramentas CASE

Como pode ser observado na Tabela 8, cada fator do desenvolvimento de software tem uma questão correspondente apresentada como uma frase perguntando sobre sua influência na produtividade do desenvolvimento de software. Para responder a uma pergunta, o participante marca a sua respectiva VAS, variando de “diminuiu fortemente” a “aumenta fortemente”, a fim de preencher as sentenças que lhes são apresentadas. As respostas foram representadas em valores numéricos no intervalo de -1 a 1, com 1% de precisão.

Além disso, o participante do *survey* pode explicitamente indicar que não responderá à questão, marcando um botão de opção (*radio button*). A Figura 18 apresenta o *design* das questões relacionadas aos fatores, no caso para o fator *Comunicação entre Desenvolvedores*. Como pode ser visto, o participante é orientado a posicionar o indicador (o círculo) em uma posição da barra de modo a preencher uma sentença de acordo com sua interpretação.

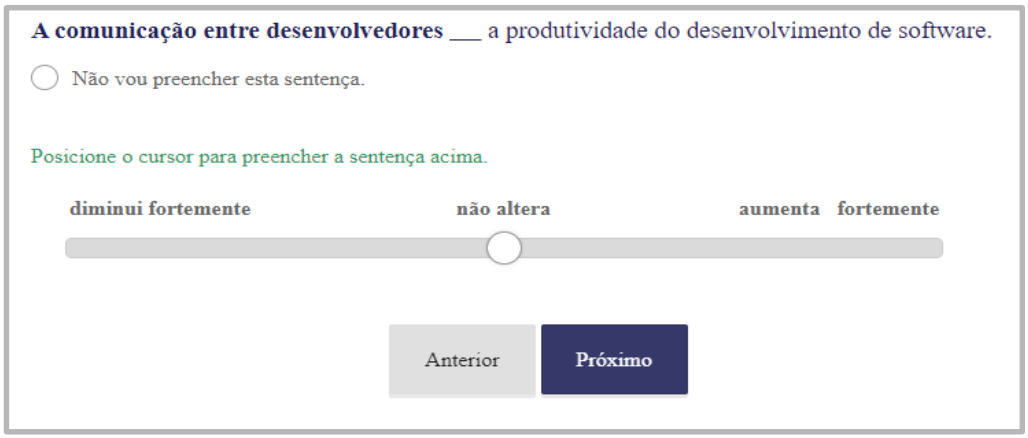

Figura 18. Exemplo de *design* das questões relacionadas aos fatores.

O *survey* foi implementado em software oferecido como um serviço na web, o *SurveyMonkey*, disponível em https://pt.surveymonkey.com/. A escolha deste software ocorreu por ter sido uma das poucas ferramentas que já possui um componente de rolagem (*slider*) pronto para a implementação de escalas VAS, de acordo com as necessidades de design das perguntas relacionadas aos fatores, não requerendo

nenhuma adequação. O serviço também conta com customizações de *layout*, *design* responsivo (que se adequa a diferentes tipos de dispositivos, por exemplo, celulares, *tablets*, *desktops*, dentre outros), a execução de rodadas de testes, divulgação em redes sociais, ferramentas de análise de acordo com o design das perguntas, entre outras. Mas uma característica importante que pode ser futuramente explorada é o recurso de internacionalização (i18n), que permite a definição das perguntas em diferentes linguagens numa mesma estrutura e instância de *survey*, o que centraliza a gestão da execução, coleta e análise dos dados.

5.2.5. Trials de Avaliação

Neste ponto do estudo, foram executadas duas rodadas para avaliar o *survey*. No primeiro *trial*, dois pós-doutorandos do Grupo de Engenharia de Software Experimental da COPPE/UFRJ foram convenientemente convidados, por sua experiência na execução de estudos experimentais, para responder a uma primeira versão do *survey*. Logo após a aplicação do questionário, os dois participantes foram solicitados via uma única pergunta a se manifestar sobre os pontos que tiveram dificuldades e sugestões que pudessem melhorar a legibilidade, compreensão ou usabilidade do questionário. Suas considerações foram discutidas, analisadas, e aquelas consideradas pertinentes consideradas em alterações em uma segunda versão.

Em um segundo *trial*, mais três participantes foram escolhidos por conveniência, um aluno de doutorado do Programa de Pós-Graduação de Engenharia de Sistemas COPPE/UFRJ e dois contatos externos com experiência na gestão de projetos de software na iniciativa pública. Todos foram contatados e questionados através da mesma pergunta realizada no primeiro *trial*. Suas considerações foram analisadas, as últimas mudanças realizadas, e o questionário foi considerado pronto para distribuição.

5.2.6. Coleta e Análise de Dados

O primeiro dia de disponibilização do *survey on-line* e envio dos convites ocorreu em 06 de Fevereiro de 2018. O *survey* ficou aberto para coleta de respostas até o dia 07 de Julho de 2018 (um período de 5 meses). As primeiras respostas foram capturadas no dia 07 de Fevereiro de 2018, e a última no dia 18 de Junho de 2018. A Figura 19 demonstra a distribuição do número de participações ao longo do período em que o survey estava disponível na web. Como pode ser constatado, o mês com mais respondentes foi em Abril com 32 respondentes, após as férias de verão no Brasil.

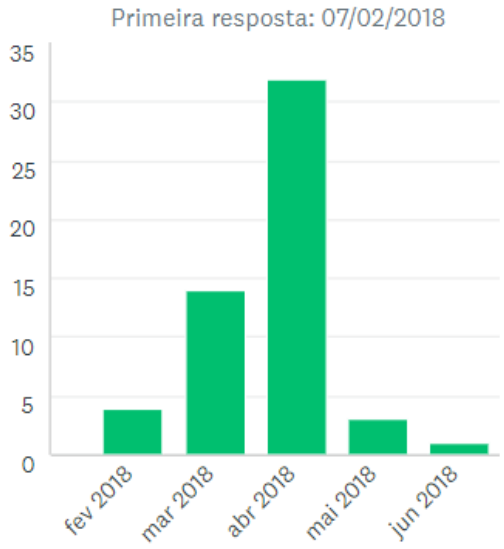

Figura 19. **Período de disponibilização do *survey*.**

Ao final, 53 gerentes de software e líderes de projetos brasileiros participaram da pesquisa, 46 deles responderam a todas as perguntas. O tempo médio de preenchimento pelos respondentes foi aproximadamente oito minutos.

Quanto à caracterização dos participantes, 38% (de 50 respondentes) relataram possuir mais de nove anos de experiência em atividades de gestão e liderança em projetos de software, os demais grupos de respondentes não passaram de 22% (ver Figura 20).

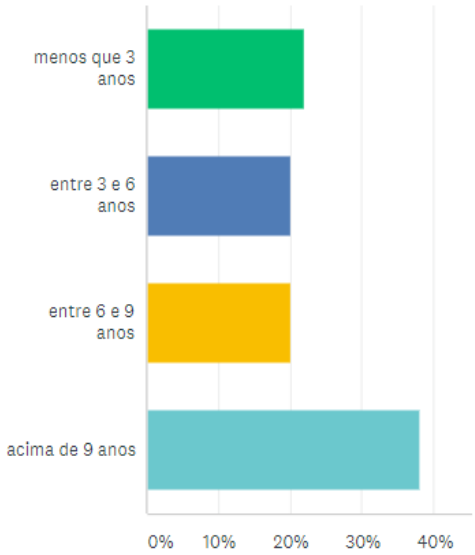

Figura 20. Experiência dos participantes em gestão e liderança.

Em relação ao nível de formação acadêmica dos participantes (Figura 17), 26% (de 50 respondentes) tinham nível superior, 34% mestrado. Nenhum participante com Nível Médio/Técnico. Já os outros grupos, 20% cada (Figura 20).

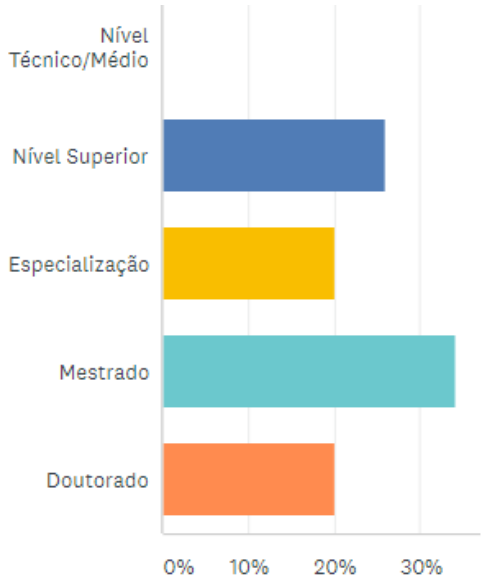

Figura 21. Nível de formação acadêmica dos participantes.

Ainda quanto à formação, os participantes foram solicitados a responder qual categoria de quatro apresentadas melhor representava a sua área de formação principal. Caso o respondente não achasse que nenhuma opção era adequada, era fornecida a opção “Outras”. Conforme pode ser visto na Figura 21, a maioria tinha formação na área de Computação ou Informática (92%) e somente um participante não foi capaz de se enquadrar em uma categoria.

OPÇÕES DE RESPOSTA	RESPOSTAS	
Computação ou Informática.	92,00%	46
Engenharia Eletrônica, Elétrica, Telecomunicações ou afins.	4,00%	2
Administração, Economia, Estatística ou afins.	2,00%	1
Outras.	2,00%	1
TOTAL		50

Figura 22. Formação principal dos participantes.

Quanto à caracterização das organizações dos respondentes, primeiro, os respondentes informaram o tamanho das equipes de TI de suas organizações em números de funcionários. As respostas nos forneceram a informação que 57% (de 49 organizações) tinham uma área de TI com mais de 100 funcionários. Os demais grupos de empresas não alcançaram mais que 21% (Figura 23). Lembrando que é possível ter mais de um respondente de uma mesma organização.

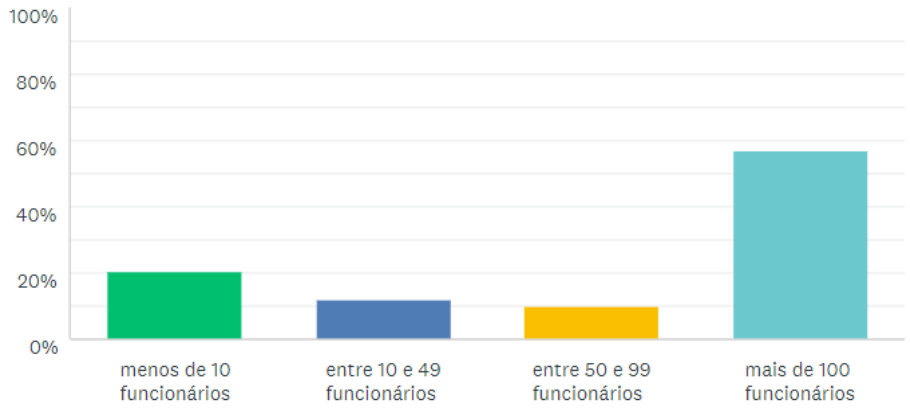

Figura 23. Tamanho das áreas de TI das organizações dos respondentes

A segunda questão de caracterização das organizações dos participantes foi sobre qual o setor melhor define a área de atuação da organização, onde as respostas foram apresentadas através de categorias. Dentre as categorias havia também a opção “Outras”, caso o participante não fosse capaz de enquadrar sua organização em nenhuma das outras áreas de atuação. As áreas de atuação mais representativas são Governo e Outsourcing de TI, conforme pode ser visto na Figura 24.

Em análises, nenhuma tendência que correlacione as informações individuais dos respondentes e das organizações às respostas dos fatores foi identificada.

OPÇÕES DE RESPOSTA		RESPOSTAS	
Saúde (Área Médica/Biológica/Farmacêutica,etc.)		4,00%	2
Finanças (Seguros, Bancos, etc.)		4,00%	2
Governo		32,00%	16
Educação		10,00%	5
Outsourcing de TI/Fábrica de Software/Customização de Software		26,00%	13
Logística		4,00%	2
Comércio Eletrônico		2,00%	1
Industria de Base (Energia, Mineração,etc.)		8,00%	4
Outro (especifique)	Respostas	10,00%	5
TOTAL			50

Figura 24. Áreas de atuações das organizações dos respondentes.

Por último, as organizações foram caracterizadas pelos tipos de software que os projeto(s) costumam produzir. Aqui, as respostas eram multivaloradas, ou seja, era fornecida um conjunto de opções fechadas e o participante poderia preencher quantos desejasse. Assim, temos que notar que o número total de respostas é superior ao número total de respondentes, como pode ser observado na Figura 25.

OPÇÕES DE RESPOSTA		RESPOSTAS	
Software embarcado		12,24%	6
Aplicativos móveis		55,10%	27
Aplicações Web		95,92%	47
Data Science/Business Inteligence		40,82%	20
ERP		22,45%	11
Soluções/Ferramentas de desenvolvimento de software		32,65%	16
Outro (especifique)	Respostas	2,04%	1
Total de respondentes: 49			

Figura 25. Tipos de software produzidos pelas organizações dos respondentes

Na Tabela 11, são exibidas as médias e o número total de respostas dos participantes de cada fator do desenvolvimento de software, sendo que estas médias computam as respectivas percepções médias dos respondentes.

A concordância entre os respondentes também foi avaliada, usando o *alpha* de *Krippendorff*. De acordo com Krippendorff (2011), o *alpha* de *Krippendorff* é amplamente aplicável sempre que dois ou mais métodos de geração de dados (respondentes) são aplicados a pelo menos dois itens de análise (perguntas da pesquisa), podendo lidar com dados incompletos e utilizando funções de diferenças para lidar com diferentes tipos de escala (nominal, ordinal, intervalar e razão).

Utilizando o cálculo com base na função de diferença para escalas intervalares, o valor encontrado para as respostas obtidas no *survey* é de 52% (uma concordância moderada) (Zapf, Castell, Morawietz, & Karch, 2016).

Como o *alpha* de *Krippendorff* não faz suposições sobre distribuição para o cálculo da concordância, em geral usa-se o método de *bootstrapping* para se analisar a distribuição teórica da amostra original. Ao realizar uma re-amostragem por bootstrapping com 5000 iterações com as respostas deste *survey*, a concordância média alcançada para a população teórica que a amostra representa é 52% com um desvio padrão de 5%. O intervalo de confiança é [41%, 61%] a um nível de significância de 95%, o que significa que a chance de a média real se encontrar dentro deste intervalo é de 95%, ou seja, a média real também está dentro da faixa de concordância moderada (entre 41% e 60%). Apesar disso, nenhuma tentativa de conclusão pode ser feita com uma média dentro desta faixa (Krippendorff, 2011). Os cálculos foram realizados com o *R Statistical Software*. A Figura 26 mostra o histograma de re-amostragens sobre a distribuição gaussiana equivalente, com a mesma média e desvio padrão (curva tracejada). Embora nenhum teste de

normalidade tenha sido realizado, a distribuição obtida a partir da re-amostragem é levemente deslocada para a direita, quando comparada à gaussiana equivalente.

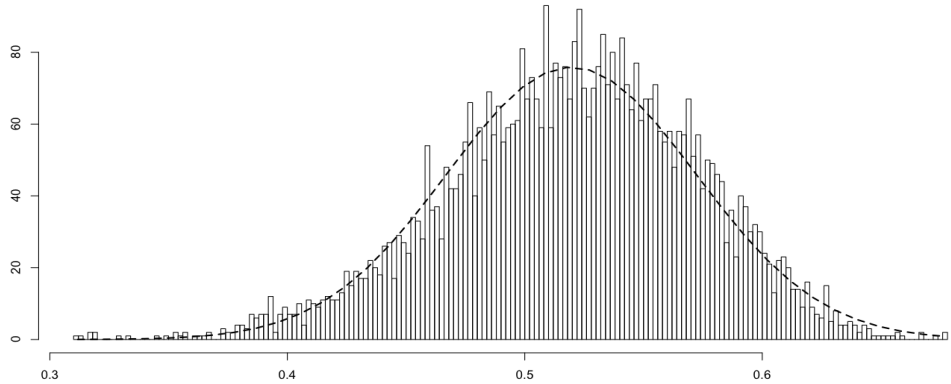

Figura 26. Histograma a partir do *bootstrapping* das concordâncias dos respondentes

Para a análise de concordância entre pesquisa e prática, ao invés de usar as intensidades dos valores de certezas imediatos encontrados (via SSM, Capítulo 4), para comparação com as percepções dos profissionais, propõe-se o uso dos valores esperados das intensidades dos fatores de desenvolvimento de software. Pois a comparação entre as intensidades dos valores de certezas imediatos e as médias das respostas dos respondentes não faria sentido neste momento. Já que o valor de certeza imediata representa a conjunção das certezas adjacentes que me dão a melhor chance de ocorrência de observar um subconjunto de efeitos do *frame* de discernimento. O que significa que os efeitos resultantes de sínteses em SSM representam intervalos na escala *Likert* do *frame* de discernimento.

Por exemplo, se considerarmos a função de certeza obtida a partir das sínteses do fator referente à Capacidade do Pessoal, a intensidade resultante pela síntese com SSM é fracamente positiva ou positiva ({↑,↑↑}), decorrente do valor de certeza imediata igual a 87%, que é composta por três componentes da função de certeza: m({↑})=0,11, m({↑↑})=0,56 e m({↑,↑↑})=0,17. Note que é através delas que o método chega a que o valor de efeito mais provável a ser observado é a o subconjunto {↑,↑↑}. Desta forma, em escalas diferentes, a única análise que seria viável é observar se as médias das percepções dos profissionais estão dentro dos intervalos resultantes das sínteses.

Como, na Teoria da Probabilidade, o valor esperado de uma variável aleatória é o valor médio de longo prazo das tentativas de um experimento (Hamming, 2018), utilizar os valores esperados com base nas funções de certezas são mais apropriados, porque: (1) eles dão a oportunidade para cada elemento único e, até mesmo, divergente de evidência, se manifestar através de sua contribuição ponderada no

cálculo; como uma previsão de longo prazo que leva em consideração todos os possíveis efeitos presentes em um conjunto de evidências, de acordo com a função de certeza resultante de cada síntese. Em outras palavras, encontramos um efeito médio para cada fator do desenvolvimento de software baseado em sua função de certeza; (2) as respostas de nosso *survey* coletam informações através de frases que tentam capturar as percepções gerais sobre os fenômenos, que são mais próximas das impressões de longo prazo sobre os efeitos na produtividade do desenvolvimento de software do que as de um intervalo no curto prazo.

Entretanto, ao contrário de calcular um valor esperado com base em uma função de densidade de probabilidade, calcular um valor esperado com base em uma função de certeza requer a decisão sobre como lidar com sua incerteza ($m(\Theta)$).

Tabela 9. Valores de conversão entre escalas

Intensidade (a)	Valores (v(a))
↓ ↓ ↓	-1.000
↓ ↓ ↓ ou ↓ ↓	-0.833
↓ ↓	-0.667
↓ ↓ ou ↓	-0.500
↓	-0.333
↓ ou ↔	-0.167
↔	0.000
↔ ou ↑	0.167
↑	0.333
↑ ou ↑ ↑	0.500
↑ ↑	0.667
↑ ↑ ou ↑ ↑ ↑	0.833
↑ ↑ ↑	1.000

Seguindo uma das opções de (Strat, 1994), a incerteza foi igualmente distribuída sobre todos os valores possíveis no frame do discernimento. Além disso, a intensidade

dos efeitos causais teve de ser traduzida em valores numéricos para o cálculo dos valores esperados para cada fator de desenvolvimento de software. Portanto, mapeamos todos os valores possíveis de intensidades no intervalo [-1 ... 1], variando em 16,7% de acordo com os valores pré-estabelecidos, de acordo com a Tabela 9.

Em um outro exemplo, a Tabela 10 mostra o cálculo do valor esperado ($E(a)$) do exemplo da síntese na Tabela 7. Note que a incerteza obtida ao final da síntese ($m_3(\Theta) = m_{1,2}(\Theta)$) foi igualmente distribuído entre todos os 13 valores de Θ (51,8% ÷ 13 = 4%) e os possíveis valores do frame de discernimento são convertidos adequadamente usando $v(a)$ (Tabela 9). Assim, o valor esperado da influência do fator de desenvolvimento de software (diversidade da equipe) obtido é -0,12.

Tabela 10. Cálculo de valor esperado com funções de certeza.

A	$m_{1,2}(a) + m_{1,2}(\Theta)/13$	$a \cdot m_{1,2}(a)$
$\{\uparrow\}$	24% + 4%	0,09
$\{\downarrow\downarrow, \downarrow\}$	24,1%+4%	-0,23
$\forall a' \in \Theta - \{\{\uparrow\}, \{\downarrow\downarrow, \downarrow\}\}$, $m_{1,2}(a') = 0.0\%$	0,0%+4%	$[4\% . \sum v(a')]$=0,02
	$E(a)$=	-0,12

Do mesmo modo, o caminho contrário, de valores esperados para valores do frame de discernimento, também pode ser percorrido, através de uma função de mapeamento que distribua os valores possíveis da escala *Likert* em faixas equidistantes dentro do intervalo [-1 … 1], conforme pode ser visto na Figura 27.

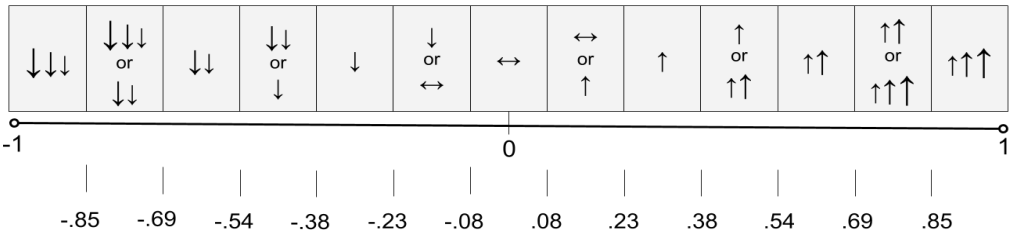

Figura 27. Mapeamento de valores esperados em *Likert*

Assim, a Tabela 11 apresenta os resultados obtidos para a pesquisa (os valores esperados a partir das funções de certeza das sínteses) e a prática (as médias das percepções a partir do *survey*). Podemos verificar que é possível confrontar tais

resultados. Esta tabela também mostra a influência dos fatores de desenvolvimento de software na produtividade de acordo com a escala *Likert* de SSM em ambos os casos.

Tabela 11. Comparação entre pesquisa e prática.

	Prática (*Survey*)			Pesquisa (SLR+Sínteses)		
	# de respondentes	Intensidade Média	Média da Percepção	Valor de certeza imediata	Intensidade Média	Valor esperado
Linguagem de Programação	49	↑ ou ↑↑	0.401	72%	↑ ou ↑↑	0.399
Área de Negócios	53	↓↓ ou ↓	-0.351	68%	↓	-0.257
Capacidade do Pessoal	54	↑↑	0.675	84%	↑↑	0.547
Nível de Maturidade	43	↑	0.292	70%	↑ ou ↑↑	0.423
Uso de Métodos de Qualidade	47	↑	0.250	62%	↑ ou ↑↑	0.383
Uso de Ferramentas	46	↑	0.343	49%	↔ ou ↑	0.186
Duração do Projeto	50	↓ ou ↔	-0.105	68%	↓	-0.292
Volatilidade de Requisitos	53	↓↓	-0.597	85%	↓	-0.363
Flexibilidade de Reuso	50	↑	0.350	86%	↑↑	0.588
Autonomia da Equipe	48	↔ ou ↑	0.150	72%	↑ ou ↑↑	0.417
Conhecimento de Domínio	53	↑↑ ou ↑↑↑	0.701	74%	↑ ou ↑↑	0.429
Experiência dos Desenvolveres no Desenvolvimento de Software	53	↑↑ ou ↑↑↑	0.708	76%	↑	0.351
Complexidade de Artefatos	53	↓	**-0.298**	67%	↔	**0.068**

Comunicação entre Desenvolvedores	52	↑↑ ou ↑↑↑	0.802	70%	↑ ou ↑↑	0.407
Disponibilidade e Alocação de Pessoal	52	↑↑ ou ↑↑↑	0.705	71%	↑	0.295
Metodologia de Desenvolvimento de Software	51	↓↓ ou ↓	**-0.372**	68%	↔ ou ↑	**0.147**
Tamanho de software baseado em LOC	52	↓ ou ↔	**-0.147**	96%	↑↑	**0.581**

Tomando as percepções médias dos respondentes como componentes de um vetor representando a prática vivenciada nas organizações de software e os valores esperados a partir das funções de certezas das sínteses como componentes de um vetor representando os resultados acadêmicos nos permite medir a distância euclidiana entre pesquisa e prática relativa à influência de tais fatores na produtividade do desenvolvimento de software. Tal distância pode sugerir um nível de convergência entre a academia e a indústria. Distância zero representa uma situação ideal (convergência total). A situação inesperada é divergência total (uma distância máxima igual a 8,246, assumindo valores extremos e opostos).

Neste trabalho, a distância calculada é de 1,327, representando 16,09% do valor máximo. Além disso, ao levar em consideração as intensidades das percepções médias dos praticantes e os valores esperados das sínteses convertidos na escala *Likert* de SSM, a concordância entre pesquisa e prática é de 72% com *p-value* = 0,113 ($\chi 2$ (16) = 23,09), usando o coeficiente *w* de *Kendall* não corrigido por laços dentre os avaliadores (Legendre, 2005).

O coeficiente *w* de *Kendall* é útil para avaliar a concordância entre avaliadores quando os itens sob análise são classificados/julgados através de escalas ordinais com mais de três valores distintos, pois seu cálculo leva em consideração que quão maior a diferença entre as avaliações de um item de análise, menor a chance de concordância entre avaliadores, ou seja, a diferença da classificação/julgamento de um novo item em relação à sua respectiva média importa. Por outro lado, o *p-value* nos indica que a chance de considerarmos um resultado inválido como sendo válido está em torno 12%. Então, para um estudo que busca demonstrar uma indicação inicial, podemos assumir que os respondentes estão aplicando essencialmente os mesmos padrões ao avaliar os itens de análise (Siegal, 1956).

Além disso, todos os itens de análise (fatores) estão mapeados em intervalos de efeitos do frame de discernimento, podemos adicionalmente constatar que as médias de todos os fatores, exceto os fatores Metodologia de Desenvolvimento e Tamanho de Software baseado em LOC, se interseccionam ou se tangenciam. Por exemplo, no fator Área de Negócios, podemos constatar que "↓" da pesquisa está contido no intervalo "↓↓ ou ↓" da prática. Há casos ao contrário, em que a prática está contida na pesquisa (Nível de Maturidade), e aqueles em que os intervalos coincidem integralmente (Linguagem de Programação), corroborando visualmente para o resultado da concordância obtido anteriormente. No fator Complexidade de Artefatos, os efeitos se tangenciam no frame de discernimento.

À título meramente ilustrativo, trazendo os resultados dos efeitos de curto prazo de SSM e colocando-os lado a lado com as médias da Tabela 11 (ver Tabela 12), verificamos que em sua grande maioria os efeitos se interseccionam ou se tangenciam. Mostrando que mesmo considerando efeitos de curto prazo os resultados ainda essencialmente convergem.

Tabela 12. Comparação entre pesquisa (valores esperados e SSM) e prática (*survey*).

	Prática	Pesquisa	
	Percepção Média dos profissionais (*survey*)	Intensidade Média (valores esperados das funções de certeza)	Intensidade de curto prazo (SSM)
Linguagem de Programação	↑ ou ↑↑	↑ ou ↑↑	↑ ou ↑↑
Área de Negócios	↓↓ ou ↓	↓	↓ ou ↔
Capacidade do Pessoal	↑↑	↑↑	↑ ou ↑↑
Nível de Maturidade	↑	↑ ou ↑↑	↑ ou ↑↑
Uso de Métodos de Qualidade	↑	↑ ou ↑↑	↑ ou ↑↑
Uso de Ferramentas	↑	↔ ou ↑	↑ ou ↑↑
Duração do Projeto	↓ ou ↔	↓	↓↓ ou ↓
Volatilidade de Requisitos	↓↓	↓	↓↓ ou ↓

Flexibilidade de Reuso	↑	↑ ↑	↑
Autonomia da Equipe	↔ ou ↑	↑ ou ↑ ↑	↑ ou ↑ ↑
Conhecimento de Domínio	↑ ↑ ou ↑ ↑ ↑	↑ ou ↑ ↑	↑ ou ↑ ↑
Experiência na Tecnologia	↑ ↑ ou ↑ ↑ ↑	↑	↑
Complexidade de Artefatos	**↓**	**↔**	**↓ ou ↔**
Comunicação entre Desenvolvedores	↑ ↑ ou ↑ ↑ ↑	↑ ou ↑ ↑	↑ ↑
Disponibilidade e Alocação de Pessoal	↑ ↑ ou ↑ ↑ ↑	↑	↑ ou ↑ ↑
Metodologia de Desenvolvimento de Software	**↓ ↓ ou ↓**	**↔ ou ↑**	**↔ ou ↑**
Tamanho de software baseado em LOC	**↓ ou ↔**	**↑ ↑**	**↑ ou ↑ ↑**

5.2.7. Conclusões e Relatos

É possível que a maioria dos pesquisadores e profissionais de ES considere uma distância de 16,09% abaixo do esperado, dadas as atividades intensivas em conhecimento e as dificuldades na realização de estudos empíricos sobre produtividade de software em ES. O fato de termos limitado o conjunto de fatores de desenvolvimento de software que os selecionam com base nos níveis mais baixos de incerteza poderia ter contribuído para alcançar esse resultado.

No entanto, os seis últimos fatores de desenvolvimento de software na Tabela 11 (abaixo da linha tracejada) são os fatores que suas diferenças contribuem mais positivamente para aumentar a distância entre os vetores acadêmico e de representação prática. Portanto, a pesquisa e a prática divergem mais notoriamente sobre esses fatores. Ao retirar esses fatores do cálculo, a distância euclidiana vai para 0,558 (8,42% do respectivo valor máximo), e o valor do *w* de *Kendall* passa para 79,8%, com *p-value* = 0,101 ($\chi 2$ (10) = 15,95), não corrigido para laços entre os avaliadores também.

Além disso, os fatores “Complexidade de Artefatos”, “Metodologia do Desenvolvimento de Software” e “Tamanho do Software baseado em LOC” divergem na direção da influência revelada, ou seja, apresentam sinais opostos. Assim, eles

destacam as contradições relevantes entre evidências da academia e as percepções da prática. A partir dos dados coletados, não foi possível entender esse comportamento inesperado, mas tal ponto merece atenção em estudos futuros.

5.3. Ameaças à Validade

Sobre o *survey*, existem três ameaças significativas à sua validade. A primeira é uma ameaça de validade de construção, e surge do viés dos pesquisadores durante o planejamento da execução e o projeto do *survey*, que foram abordados através da definição de sentenças com base nos conceitos encontrados nas evidências obtidas de um protocolo de SLR e definidos através de codificação teórica.

A segunda é uma ameaça à validade interna, e refere-se à incompreensão dos conceitos quando apresentados aos participantes. Ele foi abordado definindo a perspectiva a ser adotada pelos participantes ao apresentar cada conceito através de questões com um *rationale* ao invés de usar um simples rótulo. Além disso, cinco participantes em dois *trials* também avaliaram o *survey* antes de seu uso pelos respondentes finais.

A terceira é a população local e amostragem por conveniência. Com 32% de participantes atuando no desenvolvimento de software para o Governo, 26% atuando em empresas de *Outsourcing de TI* e 8% atuando na Indústria de Base, não há como assumir qualquer representatividade da amostra em relação a sua população. No entanto, todos os participantes apresentaram perfis adequados, representaram diferentes organizações de software e domínios de aplicação, além de apresentarem experiência suficiente em projetos de software. Adicionalmente, segundo a pesquisa realizada pela IDC Brasil em 2017 (disponível em http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/af_abes_publicacao-mercado_2018_small.pdf), esses três estratos representam fatias significativas do mercado brasileiro atualmente (Governo com 3,7%, Outsourcing de TI com 41,3% e Indústria com 19,1%).

Por último, embora a replicação da pesquisa em outras populações seja possível, está além do escopo desta tese.

5.4. Conclusão

Como mostrado ao longo deste capítulo, temos indícios razoavelmente significantes que, pelo menos quando se propõe a identificar fatores e suas influências na produtividade do desenvolvimento de software, a pesquisa experimental em ES tem alcançado certo sucesso. Além disso, demonstramos que usar a certeza imediata como indicativo da confiança nos resultados das sínteses é viável e razoável.

De 17 fatores considerados na comparação proposta, a pesquisa e a prática somente divergem sobre três deles. Curiosamente, a maior divergência refere-se ao preditor mais significativo encontrado nas sínteses, Tamanho de Software baseado em LOC. Como este fator é quantitativamente mensurado e as evidências apontam para uma certeza quase absoluta, ou a percepção média dos profissionais não condiz com a realidade, ou este fator representa algo que não é tamanho de software de acordo com o senso geral admitido por boa parte dos praticantes.

Levando em conta todas essas considerações, os pesquisadores de ES ainda precisam discutir a necessidade de realizar novas investigações, mesmo no contexto do fortalecimento das evidências exploradas nesta tese, em termos de resultados das sínteses, do *survey* com profissionais da prática, bem como dos resultados da comparação entre ambas. O caso do Tamanho de software baseado em LOC é um bom exemplo.

Adicionalmente, a Comunidade de ES deveria periodicamente avaliar se as diferenças encontradas entre a pesquisa e prática fazem sentido e podem ser de alguma forma aceitas. Se sim, devemos defender a necessidade de se chegar a um consenso (Johnson *et al.*, 2012; Ralph, 2015; Johnson *et al.*, 2018). Caso contrário, será necessário estabelecer novas direções que a pesquisa deve tomar para resolver tais diferenças. No contexto desta tese, para todos os fatores avaliados no *survey* (exceto *Tamanho de Software baseado em LOC*) acreditamos que é possível discutir e aceitar os resultados encontrados, e a partir dessa aceitação estabelecer leis estabelecendo os comportamentos observados para pelo menos parte desses achados.

Capítulo 6: Conclusões

Este capítulo conclui este manuscrito descrevendo as conclusões, bem como expectativas e oportunidades futuras.

6.1. Considerações Finais

Esta tese apresentou um modelo de estruturas teóricas que organiza e sintetiza o conhecimento produzido sobre a influência de fatores na produtividade do desenvolvimento de software, com base em uma análise qualitativa a partir de evidências obtidas através de um protocolo de revisão sistemática. Os fatores identificados foram devidamente relatados e disponibilizados de acordo com suas definições e perspectiva de síntese das evidências.

O modelo de estruturas teóricas alcançado tem seus resultados discutidos em termos da confiança que podemos ter nas observações, e é comparado em termos de suas implicações em proposições já estabelecidas e de seu uso pela Comunidade de ES. Ao final, uma avaliação experimental é realizada na tentativa de verificar a convergência e o grau de concordância entre a pesquisa, que este modelo representa, e a prática, vivenciada em projetos de software.

As ameaças à validade são colocadas e discutidas em cada capítulo de acordo com a respectiva etapa da metodologia de pesquisa. Sem dúvida nenhuma, o viés do pesquisador na seleção e interpretação das evidências é a maior delas, devidamente endereçada pelo uso de SSM e discussões com o Grupo de Engenharia de Software Experimental da COPPE/UFRJ. Entretanto, nenhuma análise ou estudo com terceira parte para verificar a confiabilidade interna ou a concordância da codificação teórica e das sínteses foi realizada. De qualquer forma, atente que a intenção desta Tese sempre foi ter um resultado que sirva como um ponto de partida, ou seja, sem nenhum caráter confirmatório.

Além disso, determinar como os resultados do *framework* teórico obtido e sua comparação com a prática podem ser generalizadas é um grande desafio. É provavelmente baixo, já que houveram limites na seleção das evidências e a amostragem dos participantes no *survey* foi não-probabilística. Os respondentes do *survey* demonstraram experiência na gestão/liderança de projetos, em organizações e setores da indústria de software bem variados. Embora

Já a utilização das funções de certeza como um indício da confiança que podemos ter nos resultados das sínteses se demonstrou viável e consistente, mas o

nível de confiança estatística não pode ser determinado/fundamentado a partir dos dados.

Baseado nas informações sobre a convergência e concordância entre pesquisa e prática obtidas, a última parte dos resultados serve para nos dar uma indicação inicial de que parte das estruturas teóricas merecem ser discutidas. Entretanto, cabe à Comunidade de ES determinar se as diferenças entre a pesquisa e prática podem ser de alguma forma aceitas para, então, estabelecer um consenso inicial para pelo menos parte dos achados. Além disso, determinar o quanto e em que pontos este corpo de conhecimento deveria evoluir passa pela questão de sermos capazes de periodicamente entendermos o status dessas diferenças.

Por fim, voltando às questões de pesquisa apresentadas no Capítulo 1, resumidamente, concluímos que:

> ***RQ1: É possível encontrar proposições que sintetizem hipóteses, leis ou teorias utilizadas em estudos já realizados em produtividade no contexto do processo de desenvolvimento de software?***
>
> Sim, é possível encontrar proposições em estudos primários realizados em produtividade do desenvolvimento de software, bem como descrevê-las em termos de efeitos de curto prazo, com base nas certezas imediatas, ou de longo prazo, com base em valores esperados, de acordo com a metodologia de pesquisa empregada.
>
> > ***RQ1.1: Se sim, essas proposições podem ser usadas para oferecer um referencial teórico inicial no tema produtividade do desenvolvimento de software a ser utilizado em diferentes situações pela Comunidade de ES?***
> >
> > Mesmo para os casos cuja confiança nos resultados é alta (> 65%), esta tese naturalmente poderia caracterizar qualquer um dos achados do *framework* teórico como hipótese, mas certamente ainda não seria prudente rotulá-los como leis. O resultado da comparação entre as proposições oriundas das estruturas teóricas e as percepções dos profissionais da prática fornecem indícios de que o *framework* teórico obtido poderia ser aceito como um conjunto de teorias do Tipo-III (Hannay *et al.*, 2007), ou seja, possuem o intuito de predizer sem obrigação de oferecer uma explicação, completa ou não, dos fenômenos. Para que isto ocorra, todos os resultados precisam ser

colocados e discutidos de maneira ampla e suas implicações avaliadas de maneira profunda.

6.2. Contribuições

As contribuições desta pesquisa incluem aspectos relativos à engenharia de software experimental e baseada em evidência.

Primeiramente, observando as revisões encontradas na literatura e as leis e hipóteses abordadas em Endres & Rombach (2003), pode-se verificar que nenhuma iniciativa anterior identifica ou expressa os efeitos esperados na produtividade do desenvolvimento de software e a confiança que se pode atribuir a tais efeitos. Então, uma primeira contribuição desta tese é alcançar resultados explorando e explicitando tais aspectos, com uma abordagem baseada em evidência. Por si só, esta tentativa já se diferencia do que foi encontrado na literatura técnica e explicita de forma mais incisiva o que se deve esperar ou observar em campo ou laboratório, o que pode ser considerado uma das principais contribuições desta Tese. O corpo de conhecimento proposto é concretamente composto por: (1) um conjunto de estruturas teóricas já interpretadas e avaliadas, que podem ser imediatamente utilizadas em futuros estudos sobre produtividade do desenvolvimento de software e iniciativas de melhorias; (2) um outro conjunto de proposições cujo o grau de certeza ainda precisará ser melhorado para que os fenômenos sejam melhor compreendidos e aplicados à prática, subsidiando novas pesquisas e incentivando a busca por repetibilidade de resultados, e; (3) um conjunto de informações sobre fatores e medidas para operacionalização de construtos envolvidos nas observações tanto em estudos primários quanto na prática envolvendo a produtividade do desenvolvimento de software.

Uma segunda contribuição é que, notadamente, este trabalho endereça a questão metodológica sobre dois desafios: (1) como identificar e depois agregar novas evidências para evoluir o conhecimento adquirido, e; (2) como melhor observar a lacuna entre pesquisa e prática. No primeiro, este trabalho se diferencia pelo número de fatores de produtividade (elementos causais) colocados sob uma mesma representação, indo além do que o SSM inicialmente se propôs e das sínteses até o momento realizadas com esse método. No segundo, esta Tese quantifica o nível de concordância entre respondentes e, em seguida, entre pesquisa e prática, demonstrando que pesquisadores de ES têm conseguido identificar e investigar fenômenos relacionados à produtividade do desenvolvimento de software, mesmo quando não está claro como medi-los.

Para gerentes e líderes de projetos de software, nossos resultados podem ser usados para avaliar os riscos dos projetos de software, servindo como um conjunto inicial de fatores baseados em evidências que podem afetar a produtividade do desenvolvimento de software. Assim, a diferença absoluta entre pesquisa e prática revelada para cada fator de desenvolvimento de software pode ser uma indicação de como sua ocorrência e impacto não podem ser facilmente previsíveis em longo prazo. Portanto, quanto maior a diferença, mais cuidadosamente os riscos de projetos de software relacionados a esse fator de desenvolvimento de software devem ser monitorados e avaliados.

A última contribuição desta Tese é a metodologia empregada para investigar e representar o conhecimento sobre a influência de fatores em uma varíavel dependente específica. Tudo indica que esta metodologia pode ser utilizada para investigar a influência de outros fatores em outros temas, em diversos períodos.

6.3. Perspectivas Futuras

Em pesquisas futuras os pesquisadores terão oportunidades de tratar os problemas em aberto ou discutir e aplicar o método de pesquisa executado em outros temas da ES. Como perspectivas futuras diretamente relacionadas a esta Tese listamos as seguintes possibilidades:

- As proposições e evidências deveriam ser analisadas e discutidas em busca de explicações para os fenômenos observados. Assim, algumas leis e, talvez, teorias possam ser propostas, bem como alguns *insights* iniciais sob a quantificação dos fenômenos descritos. A expectativa é que ter este corpo de conhecimento explicitado e disponibilizado para toda a Comunidade de ES possa de alguma forma motivar alguma reflexão sobre o uso dos resultados alcançados em alguns dos métodos de medição empregados na prática.
- Para aqueles fatores que já possuem um grau de certeza significativo (> 65%) fica a possibilidade de questionar se o que já foi feito basta ou deveriam ser explorados novos cenários capazes de refutar ou confirmar o conhecimento que se tem sobre o fenômeno. O que pode ser uma oportunidade para repetição de estudos primários com foco nos resultados mais relevantes até um determinado momento. Já para os outros fatores fica a possibilidade da realização de novos estudos experimentais cobrindo as deficiências e buscando resultados mais expressivos.

- Através da evolução, formalização e disponibilização dos resultados do protocolo de revisão sistemática executado, viabilizar atualizações futuras do *framework* a um esforço e tempo menores.

Além disso, os pesquisadores devem se perguntar quais melhorias metodológicas devem ser implementadas em seus estudos primários e secundários, ou mesmo nos métodos empregados na prática. Deste modo, é necessário endereçar os seguintes desafios no futuro:

1. Definir medidas apropriadas de entrada e saída e sua relevância para as partes interessadas através do processo de desenvolvimento de software. Neste primeiro ponto, há trabalhos com discussões pertinentes. Por exemplo, Hernández-López *et al.* (2015) e Duarte (2017) questionam o uso de linhas de código como medida de saída. Em outro exemplo no contexto de projetos *open source*, (Scholtes *et al.*, 2016) propôs o uso da distância de *Levenshtein* (em resumo, a quantidade de bytes alterados) no código-fonte como medida de saída e o número de desenvolvedores que realizaram um *commit* no repositório de controle de versões em um determinado período como medida de entrada (representando o tamanho da equipe durante a codificação). Então, uma sugestão seria explorar a utilização de ambas as medidas em outros artefatos sob gerência de configuração ao longo do processo de desenvolvimento de software. Como exemplo, através do *log* de mudanças na especificação de requisitos ou da matriz de rastreabilidade, calcular a distância de *Levenshtein* destes artefatos em um período como uma medida da volatilidade de requisitos.

Além disso, uma grande parte das evidências utilizadas na obtenção do corpo de conhecimento proposto neste trabalho possuem mais de dez anos de publicação. E ao longo do tempo houveram muitas mudanças técnicas ou não (diferenças na cultura e nas gerações de equipes e organizações, hábitos de trabalho, infraestrutura, metodologias de desenvolvimento e muitas outras).

Então, utilizar o *framework* e a lista de fatores obtidos para facilitar o estudo de questões contemporâneas, tais como: a pouca utilização de alguns métodos; a identificação de gargalos entre atividades e processos de software; a medição de fenômenos em novos meios ou formas de desenvolvimento de software, e; a mensuração de efeitos derivados de introduções e/ou mudanças tecnológicas ou organizacionais, etc. Todas essas são questões atualmente pertinentes e necessárias. Por exemplo, novos conceitos e metodologias têm sido propostas nos últimos anos,

tais como *DevOps* (Cois, Yankel, & Connell, 2014; Wahaballa, Wahballa, Abdellatief, Xiong, & Qin, 2015), Integração Contínua (Vasilescu, Yu, Wang, Devanbu, & Filkov, 2015), Implantação Contínua (Leppanen *et al.*, 2015; Rodríguez *et al.*, 2017), Entrega Contínua (Chen, 2016; Itkonen, Udd, Lassenius, & Lehtonen, 2016), etc. O estudo sobre a "engenharia" diante dessa nova realidade muda a ênfase de uma gestão tradicional de projetos (com foco em atividades, marcos e artefatos) para a gestão de filas, fluxos e recursos utilizados para a produção de software, o que se tem chamado de Engenharia de Software Contínua (ESC) (Fitzgerald & Stol, 2014, 2017). Se ajustarmos a perspectiva do processo de desenvolvimento a essa realidade poderíamos defini-lo como um *pipeline* de tarefas ou atividades. Então, seria importante entender quais as adaptações necessárias nos métodos de medição de produtividade do desenvolvimento de software dentro desta realidade. No contexto dessa tese, seria razoável mensurar e observar os fenômenos descritos no *framework* de estruturas teóricas em cada elemento do *pipeline* e observar quais fatores têm maior contribuição ou provocam gargalos entre *releases*, ou seja, uma instanciação de um modelo de medição com base nos fatores e relações descritos no *framework* obtido para cada elemento do *pipeline*.

Certamente é necessário estabelecer os novos rumos que a pesquisa precisa tomar para responder aos desafios desta nova realidade. Entretanto, a Comunidade de ES pode se beneficiar e acelerar este processo se evoluir seu corpo de conhecimento cobrindo novos métodos e paradigmas através da compreensão do que foi colocado em prática no passado.

Referências Bibliográficas

ABNT. (2015a). “NBR ISO 9000:2015 - – Sistemas de gestão da qualidade e garantia da qualidade – Fundamentos e Vocabulário”. Disponível em: <http://www.abntcatalogo.com.br/norma.aspx?ID=345040>. Acesso em: 16 de novembro de 2018.

ABNT. (2015b). “NBR ISO 9001:2015 - – Sistemas de gestão da qualidade - Requisitos”. Disponível em: <https://www.abntcatalogo.com.br/norma.aspx?ID=345041>. Acesso em: 16 de novembro de 2018.

ABNT. (2010). “NBR ISO 9004:2010 - – Gestão para o sucesso sustentado de uma organização — Uma abordagem da gestão da qualidade”. Disponível em: <https://www.abntcatalogo.com.br/norma.aspx?ID=62192>. Acesso em: 16 de novembro de 2018.

Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. Z. (2010). “Agile software development: Impact on productivity and quality”. In: *2010 IEEE International Conference on Management of Innovation & Technology*. https://doi.org/10.1109/icmit.2010.5492703

Anderson, T. R., & Ghavami, P. K. (1999). “Using data envelopment analysis for evaluating alternative software development process configurations”. In: *Management of Engineering and Technology, 1999. Technology and Innovation Management. PICMET ’99*. *Portland International Conference on*, v. 1, pp. 298-.

Aquino, G., & Meira, S. (2009). “Software productivity measurement: Past analysis and future trends”. In: *SETP’09: Proceedings of Third International Conference on Software Engineering Theory and Practice*.

Ardagna, C. A. a., Damiani, E. a., Frati, F. a., Oltolina, S. b., Regoli, M. a., & Ruffatti, G. b. (2010). “Spago4Q and the QEST nD Model: An open source solution for software performance measurement”. In: *IFIP Advances in Information and Communication Technology*, 319 AICT, pp. 1–14.

Atkins, D., Best, D., Briss, P. A., Eccles, M., Falck-Ytter, Y., Flottorp, S., GRADE Working Group. (2004). “Grading quality of evidence and strength of recommendations”. *BMJ (Clinical research Ed.)* , v. 328.

Avritzer, A., & Lima, A. (2009). “An Empirical Approach for the Assessment of Scheduling Risk in a Large Globally Distributed Industrial Software Project”. In:

2009 Fourth IEEE International Conference on Global Software Engineering. https://doi.org/10.1109/icgse.2009.53

Banker, R. D., & Slaughter, S. A. (1997). “A Field Study of Scale Economies in Software Maintenance”. *Management science*, v. 43, n.12, pp. 1709–1725.

Barke, H., & Prechelt, L. (2018). “Some reasons why actual cross-fertilization in cross-functional agile teams is difficult”. In: *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*. pp. 97–103.

Basili, V., Caldiera, G., & Rombach, H. D. (2002). “Goal question metric (gqm) approach”. *Encyclopedia of Software Engineering*. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/0471028959.sof142/full>. Acesso em: 16 de novembro de 2018.

Basili, V., Heidrich, J., Lindvall, M., Münch, J., Seaman, C., Regardie, M., & Trendowicz, A. (2009). “Determining the Impact of Business Strategies Using Principles from Goal-oriented Measurement”. In: *Proceedings of 9th BI Conference*. Disponível em:

<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.700.229>.

Basili, V. R. (1992). “Software modeling and measurement: the Goal/Question/Metric paradigm”. Disponível em:

<https://drum.lib.umd.edu/bitstream/handle/1903/7538/?sequence=1>. Acesso em: 17 de novembro de 2018.

Beecham, S., OLeary, P., Richardson, I., Baker, S., & Noll, J. (2013). “Who are we doing global software engineering research for?” In: *2013 IEEE 8th International Conference on Global Software Engineering (ICGSE),* pp. 41–50.

Bibi, S., Stamelos, I., & Angelis, L. (2008). “Combining probabilistic models for explanatory productivity estimation”. *Information and Software Technology*, v. 50 n. 7, pp. 656–669.

Biolchini, J., Mian, P. G., Natali, A. C., & Travassos, G. H. (2005). “Systematic Review in Software Engineering: Relevance and Utility”, Relatório Técnico Nº. ES 679/05. COPPE/UFRJ. Disponível em:
<https://www.cos.ufrj.br/uploadfile/es67905.pdf>. Acesso em 17 de Novembro de 2018.

Bjarnason, E., Smolander, K., Engström, E., & Runeson, P. (2016). “A theory of distances in software engineering”. *Information and Software Technology*, v. 70, pp. 204–219.

Boehm, B.W. (1981). “Software engineering economics”. Englewood Cliffs (NJ): Prentice-hall.

Boehm, B. (2003). “Value-based Software Engineering: Reinventing”. *SIGSOFT Softw. Eng. Notes*, v. 28, n. 2.

Boehm, B. W., Madachy, R., & Steece, B. (2000). “Software cost estimation with Cocomo II”. Disponível em: <https://dl.acm.org/citation.cfm?id=557000>.

Bourdel, N., Alves, J., Pickering, G., Ramilo, I., Roman, H., & Canis, M. (2015). “Systematic review of endometriosis pain assessment: how to choose a scale?”, *Human Reproduction Update*, v. 21, n.1, pp. 136–152.

Briand, L. C., Morasca, S., & Basili, V. R. (2002). “An operational process for goal-driven definition of measures”. *IEEE Transactions on Software Engineering*, v. 28, n.12, pp. 1106–1125.

Card, D. N. (1987). “A software technology evaluation program”. *Information and Software Technology*. https://doi.org/10.1016/0950-5849(87)90028-0 .

Carrillo de Gea, J. M., de Gea, J. M. C., Nicolás, J., Fernández Alemán, J. L., Toval, A., Ouhbi, S., & Idri, A. (2016). “Co-located and distributed natural-language requirements specification: traditional versus reuse-based techniques”. *Journal of Software: Evolution and Process*, v. 28, n. 3, pp. 205–227.

Castelvecchi, D., & Witze, A. (2016). “Einstein’s gravitational waves found at last”. *Nature News*. https://doi.org/10.1038/nature.2016.19361 .

Cataldo, M., & Herbsleb, J. D. (2013). “Coordination Breakdowns and Their Impact on Development Productivity and Software Failures”. *IEEE Transactions on Software Engineering*, v. 39, n. 3, pp. 343–360.

C. Duarte, C. H. (2017). “Productivity paradoxes revisited”. *Empirical Software Engineering*, v. 22 n.2, pp. 818–847.

Celik, N., Lee, S., Mazhari, E., Son, Y.-J., Lemaire, R., & Provan, K. G. (2011). “Simulation-based workforce assignment in a multi-organizational social network for alliance-based software development”. *Simulation Modelling Practice and Theory*, v. 19 n.10, pp. 2169–2188.

Chang, C. K., & Jiang, H.-Y. (2006). “Modeling Software Project Scheduling Based on Team Productivity and Its Simulations”. In: *Proceedings of 2006 International Symposium on Distributed Computing and Applications to Business, Engineering and Science*.

Chapetta, W. A. (2018). “SLR’s Supplementary Material". Disponível em < https://goo.gl/1eGNme >. Acesso em: 16 de novembro de 2018.

Charmaz, K. (2008). “Grounded theory as an emergent method”. In: *Handbook of emergent methods*, pp. 155-172.

Cheikhi, L., Al-Qutaish, R. E., & Idri, A. (2012). “Software productivity: Harmonization in ISO/IEEE software engineering standards”. *JOURNAL OF SOFTWARE*, v. 7, n. 2, pp. 462–470.

Chen, L. (2016). “Continuous Delivery: Overcoming Adoption Obstacles”. In: *Proceedings of the International Workshop on Continuous Software Evolution and Delivery,* pp. 84–84.

Chen, P.-C., Chern, C.-C., & Chen, C.-Y. (2012). “Software Project Team Characteristics and Team Performance: Team Motivation as a Moderator”. In: *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, v. 1, pp. 565–570.

Chidamber, S. R., & Kemerer, C. F. (1991). “Towards a metrics suite for object oriented design”. In: *Conference proceedings on Object-oriented programming systems, languages, and applications - OOPSLA ’91*, pp. 197–211.

Cois, C. A., Yankel, J., & Connell, A. (2014). “Modern DevOps: Optimizing software development through effective system interactions”. In: *2014 IEEE International Professional Communication Conference (IPCC)* , pp. 1–7.

Colazo, J. A. (2008). “Following the sun: Exploring productivity in temporally dispersed teams”. In: *AMCIS 2008 Proceedings*, v. 3, pp. 1833–1839.

Colomo-Palacios, R., Casado-Lumbreras, C., Soto-Acosta, P., García-Peñalvo, F. J., & Tovar, E. (2014). “Project managers in global software development teams: a study of the effects on productivity and performance”. *Software Quality Journal*, v. 22, n. 1, pp. 3–19.

Cruzes, D. S., & Dybå, T. (2010). “Synthesizing evidence in software engineering research”. In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1.

de Aquino Júnior, G. S., & d. L. Meira, S. R. (2009). “An Approach to Measure Value-Based Productivity in Software Projects”. In: *2009 Ninth International Conference on Quality Software,* pp. 383–389.

Dale, C. J., & van der Zee, H. (1992). “Software productivity metrics: who needs them?”, *Information and Software Technology*, v. 34, n. 11, pp. 731–738.

de Barros Sampaio, S. C., Barros, E. A., d. Aquino Junior, G. S., e. Silva, M. J. C., & d. L. Meira, S. R. (2010). “A Review of Productivity Factors and Strategies on Software Development”. In: *2010 Fifth International Conference on Software Engineering Advances*, pp. 196–204.

de Oliveira, S. B., Valle, R., & Mahler, C. F. (2009). “A comparative analysis of CMMI software project management by Brazilian, Indian and Chinese companies”. *Software Quality Journal*, v. 18, n. 2, pp. 177–194.

de O. Melo, C., S. Cruzes, D., Kon, F., & Conradi, R. (2013). “Interpretative case studies on agile team productivity and management”. *Information and Software Technology*, v. 55, n.2, pp. 412–427.

de Souza Carvalho, W. C., Rosa, P. F., dos Santos Soares, M., da Cunha Junior, M. A. T., & Buiatte, L. C. (2011). “A Comparative Analysis of the Agile and Traditional Software Development Processes Productivity”. In: *2011 30th International Conference of the Chilean Computer Science Society*. https://doi.org/10.1109/sccc.2011.11 .

Dittrich, Y. (2016). “What does it mean to use a method? Towards a practice theory for software engineering”. *Information and Software Technology*, v. 70, pp. 220–231.

Di Tullio, D., & Bahli, B. (2013). “The impact of Software Process Maturity on Software Project Performance: The Contingent Role of Software Development Risk”. *Systèmes d’information & management*, v.18, n.3, pp. 85-.

dos Santos, P. S. M., Beltrão, A. C., de Souza, B. P., & Travassos, G. H. (2018). “On the benefits and challenges of using kanban in software engineering: a structured synthesis study”. *Journal of Software Engineering Research and Development*, v. 6, n. 1, pp. 13-.

dos Santos, P. S. M., & Travassos, G. H. (2017). “Structured Synthesis Method: The Evidence Factory Tool”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 480–481.

Dybå, T., Sjøberg, D. I. K., & Cruzes, D. S. (2012). “What works for whom, where, when, and why?: on the role of context in empirical software engineering”. In: *Proceedings of the ACM-IEEE International Symposium on Empirical software Engineering and Measurement,* pp. 19–28.

Endres, A., & Rombach, H.D. (2003). “A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories”. Pearson/Addison Wesley.

Erbas, C., & Erbas, B. C. (2015). “Modules and transactions: Building blocks for a theory of software engineering”. *Science of Computer Programming*, v. 101, pp. 6–20.

Farshchi, M., Jusoh, Y., & Murad, A. (2012). “Impact of personnel factors on the recovery of delayed software projects: A system dynamics approach”. *Computer Science and Information Systems*, v. 9, n. 2, pp. 627–652.

Felizardo, K. R., Mendes, E., Kalinowski, M., Souza, É. F., & Vijaykumar, N. L. (2016). “Using Forward Snowballing to update Systematic Reviews in Software Engineering”. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* pp. 53-.

Fenton, N. E., & Pfleeger, S. L. (1997). “Software Metrics: A Rigorous and Practical Approach”. PWS Publishing Company.

Fitzgerald, B., & Stol, K.-J. (2014). “Continuous software engineering and beyond: trends and challenges”. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering,* pp. 1–9.

Fitzgerald, B., & Stol, K.-J. (2017). “Continuous software engineering: A roadmap and agenda”. *The Journal of Systems and Software*, v. 123, 176–189.

Florac, W. A., & Carleton, A. D. (1999). “Measuring the Software Process: Statistical Process Control for Software Process Improvement”. Addison-Wesley Professional.

Foulds, L. R., Quaddus, M., & West, M. (2007). “Structural Equation Modelling of Large-scale Information System Application Development Productivity: the Hong Kong Experience”. In: *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*. https://doi.org/10.1109/icis.2007.174

França, B. B. N. de. (2015). Guidelines for Experimentation with Dynamic Simulation Models in the Context of Software Engineering, Tese de Doutorado (D.Sc.). UFRJ/COPPE, Rio de Janeiro, Brasil. Disponível em: <https://www.cos.ufrj.br/uploadfile/1433862223.pdf>.

Fritz, T., Mark, G., Murphy, G. C., & Zimmermann, T. (2017). “Rethinking Productivity in Software Engineering (Dagstuhl Seminar 17102)”. In: Dagstuhl Reports *(Vol. 7). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*. Disponível em: http://drops.dagstuhl.de/opus/volltexte/2017/7359/. Acesso em 17 de novembro de 2018.

Funke, F., & Reips, U.-D. (2012). “Why Semantic Differentials in Web-Based Research Should Be Made from Visual Analogue Scales and Not from 5-Point Scales”. *Field methods*, v. 24, n. 3, pp. 310–327.

Giske, T., & Artinian, B. (2007). “A Personal Experience of Working with Classical Grounded Theory: From Beginner to Experienced Grounded Theorist”. *International Journal of Qualitative Methods*, v. 6, n. 4, pp. 67–80.

Glaser, B. G. (1978). “Theoretical sensitivity: advances in the methodology of grounded theory”. Sociology Press.

Gonçalves, T. G. (2017). “Integration of human-computer interaction engineering issues into software process capability maturity models”. Ph.D. dissertation, Université de Valenciennes et du Hainaut-Cambresis. Disponível em: <https://tel.archives-ouvertes.fr/tel-01729393/document >. Acesso em 17 de novembro de 2018.

Gralha, C., Damian, D., Wasserman, A. I. (tony), Goulão, M., & Araújo, J. (2018). “The evolution of requirements practices in software startups”. In: *Proceedings of the 40th International Conference on Software Engineering*, p. 823–833.

Guidini Gonçalves, T., Marçal de Oliveira, K., & Kolski, C. (2018). “HCI in practice: An empirical study with software process capability maturity model consultants in Brazil”. *Journal of Software: Evolution and Process*, v. 46.

Günsel, A., & Açikgöz, A. (2011). “The Effects of Team Flexibility and Emotional Intelligence on Software Development Performance”. *Group Decision and Negotiation*, v. 22, n.2, pp. 359–377.

Hallberg, L. R.-M. (2006). “The “core category” of grounded theory: Making constant comparisons”. *International journal of qualitative studies on health and well-being*, v. 1, n.3. https://doi.org/10.3402/qhw.v1i3.4927

Hamming, R. W. (2018). “The Art of Probability”. CRC Press.

Hanakawa, N., Morisaki, S., & Matsumoto, K. (1998). “A learning curve based simulation model for software development”. In: *Proceedings of the 20th International Conference on Software Engineering*. https://doi.org/10.1109/icse.1998.671388

Hannay, J. E., & Benestad, H. C. (2010). “Perceived Productivity Threats in Large Agile Development Projects”. In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 15:1–15:10. New York, NY, USA: ACM.

Hannay, J. E., Sjoberg, D. I. K., & Dyba, T. (2007). “A Systematic Review of Theory Use in Software Engineering Experiments”. *IEEE Transactions on Software Engineering*, v. 33, n. 2, pp. 87–107.

He, M. a. b., Yang, Y. a., Wang, Q. a., & Li, M. a. (2008). “An investigation on performance of software enhancement projects in china”. In: *2008 15th Asia-Pacific Software Engineering Conference,* pp. 67–74.

Hernandez, C. A. (2009). “Theoretical Coding in Grounded Theory Methodology”. *Grounded Theory Review*, v.8, n. 3.

Hernández-López, A., Colomo-Palacios, R., & García-Crespo, Á. (2012). “Productivity in software engineering: A study of its meanings for practitioners: Understanding the concept under their standpoint”. In: *7th Iberian Conference on Information Systems and Technologies (CISTI 2012),* pp. 1–6.

Hernández-López, A., Colomo-Palacios, R., & García-Crespo, Á. (2013). “SOFTWARE ENGINEERING JOB PRODUCTIVITY — A SYSTEMATIC REVIEW”. *International Journal of Software Engineering and Knowledge Engineering*, v. 23, n. 03, pp. 387–406.

Hernández-López, A., Colomo-Palacios, R., García-Crespo, Á., & Cabezas-Isla, F. (2013). “Software Engineering Productivity: Concepts, Issues and Challenges”. In: *Perspectives and Techniques for Improving Information Technology Project Management*. IGI Global, pp. 69–79.

Hernández-López, A., Colomo-Palacios, R., Soto-Acosta, P., & Lumberas, C. C. (2015). “Productivity Measurement in Software Engineering: A Study of the Inputs and the Outputs”. *International Journal of Information Technologies and Systems Approach*, v. 8, n.1, pp. 46–68.

Holmes, R., Allen, M., & Craig, M. (2018). “Dimensions of experientialism for software engineering education”. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 31–39.

Hsu, J. S.-C., & Hung, Y. W. (2013). “Exploring the interaction effects of social capital”. *INFORMATION & MANAGEMENT*, 50(7), pp.415–430.

Huang, J., Sun, H., & Li, Y.-F. (2015). “An empirical study of the impact of project factors on software economics”. In: *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. https://doi.org/10.1109/ieem.2015.7385605

Hulse, R. A., & Taylor, J. H. (1975). “Discovery of a pulsar in a binary system”. *The Astrophysical journal*. Disponível em: <http://adsabs.harvard.edu/full/1975ApJ...195L..51H>.

Humphrey, W. S. (1988). “Characterizing the software process: a maturity framework”. *IEEE Software*, v. 5, n. 2, pp. 73–79.

Itkonen, J., Udd, R., Lassenius, C., & Lehtonen, T. (2016). “Perceived Benefits of Adopting Continuous Delivery Practices”. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 42-.

Ivanov, V., Rogers, A., Succi, G., Yi, J., & Zorin, V. (2017). “What do software engineers care about? Gaps between research and practice”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,* pp. 890–895.

Jeet, K., Bhatia, N., & Minhas, R. S. (2011). “A model for estimating the impact of low productivity on the schedule of a software development project”. *ACM SIGSOFT Software Engineering Notes*, v. 36, n. 4.

Jeffery, D. R. (1987). “A software development productivity model for MIS environments”. *The Journal of systems and software*, v. 7, n. 2, pp. 115–125.

Jeffery, R., Staples, M., Andronick, J., Klein, G., & Murray, T. (2015). “An empirical research agenda for understanding formal methods productivity”. *Information and Software Technology*, v. 60, pp. 102–112.

Johnson, P., & Ekstedt, M. (2007). “Search of a Unified Theory of Software Engineering”. In: *International Conference on Software Engineering Advances (ICSEA 2007)*.

Johnson, P., Ekstedt, M., & Jacobson, I. (2012). “Where’s the Theory for Software Engineering?” *IEEE Software*, v. 29, n. 5, pp. 96–96.

Johnson, P., Ralph, P., Ekstedt, M., Exman, I., & Goedicke, M. (2018). “Consensus in Software Engineering: A Cognitive Mapping Study”. *Cornell University Library arXiv [cs.SE]*. Disponível em < http://arxiv.org/abs/1802.06319>.

Kamma, D., & G, S. K. (2014). “Effect of Model Based Software Development on Productivity of Enhancement Tasks -- An Industrial Study”. In: *2014 21st Asia-Pacific Software Engineering Conference*. https://doi.org/10.1109/apsec.2014.20

Kang, D., Jung, J., & Bae, D. H. (2011). “Constraint-based human resource allocation in software projects”. *Software: practice & experience*. Disponível em: < https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.1030 >.

Kasunic, M. (2005). Designing an effective survey. Disponível em <https://resources.sei.cmu.edu/asset_files/Handbook/2005_002_001_14435.pdf >. Acesso em: 17 de Novembro de 2018.

Kemerer, C. F. (1987). “An empirical validation of software cost estimation models”. *Communications of the ACM*, v. 30, n. 5, pp. 416–429.

Kemerer, C. F. (1993). “Reliability of function points measurement: a field experiment”. *Communications of the ACM*, v. 36, n. 2, pp. 85–97.

Kenny, M., & Fourie, R. (2015). “Contrasting Classic, Straussian, and Constructivist Grounded Theory: Methodological and Philosophical Conflicts”. *The Qualitative Report*, v. 20, n.8, pp. 1270–1289.

Kitchenham, B. (2007). “Empirical paradigm--the role of experiments”. In: *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer. pp. 25–32.

Kitchenham, B. A., Dyba, T., & Jorgensen, M. (2004). “Evidence-Based Software Engineering”. In: *Proceedings of the 26th International Conference on Software Engineering*, pp. 273–281. Washington, DC, USA: IEEE Computer Society.

Klimek, L., Bergmann, K.-C., Biedermann, T., Bousquet, J., Hellings, P., Jung, K., Pfaar, O. (2017). “Visual analogue scales (VAS): Measuring instruments for the documentation of symptoms and therapy monitoring in cases of allergic rhinitis in everyday health care”. *Allergo Journa*: v. 26, n. 1, pp.16–24.

Krippendorff, K. (2011). “Computing Krippendorff’s Alpha-Reliability”. Disponível em: <https://repository.upenn.edu/asc_papers/43/>.

Krishnan, M. S., Kriebel, C. H., Kekre, S., & Mukhopadhyay, T. (2000). “An Empirical Analysis of Productivity and Quality in Software Products”. *Management Science*, v. 46, n. 6, pp. 745–759.

Kuchar, T., & Vondrak, I. (2013). “Simulating human resource capability and productivity in software process simulations”. In: *Proceedings of 25th European Modeling and Simulation Symposium, EMSS 2013,* pp. 255–263.

Lavazza, L., Morasca, S., & Tosi, D. (2016). “An Empirical Study on the Effect of Programming Languages on Productivity”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing,* pp. 1434–1439. New York, NY, USA: ACM.

Lee, Lee, & Xia. (2010). “Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility”. *The Mississippi Quarterly*, v. 34, n.1, pp. 87-.

Legendre, P. (2005). “Species associations: the Kendall coefficient of concordance revisited”. *Journal of Agricultural, Biological, and Environmental Statistics*, v. 10, n. 2, pp. 226-.

Leppanen, M., Makinen, S., Pagels, M., Eloranta, V.-P., Itkonen, J., Mantyla, M. V., & Mannisto, T. (2015). “The Highways and Country Roads to Continuous Deployment”. *IEEE Software*, n. 2, pp. 64–72.

Liang, T.-P., Wu, J. C.-H., Jiang, J. J., & Klein, G. (2012). “The impact of value diversity on information system development projects”. *International Journal of Project Management*, v. 30, n. 6, pp. 731–739.

Lindman, J., & Hammouda, I. (2018). “Support mechanisms provided by FLOSS foundations and other entities”. *Journal of Internet Services and Applications*, v. 9, n. 1, pp 8-.

Liu, J. Y.-C., Chen, H.-G., Chen, C. C., & Sheu, T. S. (2011). “Relationships among interpersonal conflict, requirements uncertainty, and software project performance”. *International Journal of Project Management*, v. 29, n. 5, pp. 547–556.

Liu, L., Kong, X., & Chen, J. (2015). “How project duration, upfront costs and uncertainty interact and impact on software development productivity? A

simulation approach". *International Journal of Agile Systems and Management*, 8(1), 39.

Lopez-Martin, C., Chavoya, A., & Meda-Campana, M. E. (2014). "A machine learning technique for predicting the productivity of practitioners from individually developed software projects". In: *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. https://doi.org/10.1109/snpd.2014.6888690

Low, G. C., & Jeffery, D. R. (1990). "Function points in the estimation and evaluation of the software process". *IEEE Transactions on Software Engineering*, v. 16, n.1, pp. 64–71.

MacCormack, A., Kemerer, C. F., Cusumano, M., & Crandall, B. (2003). "Trade-offs between productivity and quality in selecting software development practices". *IEEE Software*, v. 20, n. 5, pp. 78–85.

Madill, A., Jordan, A., & Shirley, C. (2000). "Objectivity and reliability in qualitative analysis: realist, contextualist and radical constructionist epistemologies". *British Journal of Psychology*, v. 91, n. 1, pp. 1–20.

Mannaert, H., Verelst, J., & Ven, K. (2008). "Exploring the Concept of Systems Theoretic Stability as a Starting Point for a Unified Theory on Software Engineering". In: *2008 The Third International Conference on Software Engineering Advances* , ppp. 360–366.

Martinez-Fernandez, S., Santos, P. S. M. D., Ayala, C. P., Franch, X., & Travassos, G. H. (2015). "Aggregating Empirical Evidence about the Benefits and Drawbacks of Software Reference Architectures". In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–10.

Maxwell, K. D., & Forselius, P. (2000). "Benchmarking software development productivity". *IEEE Software*, v. 17, n. 1, pp. 80–88.

Maxwell, K. D., Wassenhove, L. V., & Dutta, S. (1996). "Software development productivity of European space, military, and industrial applications". *IEEE Transactions on Software Engineering*, v. 22, n. 10, pp. 706–718.

Mendes, E., Di Martino, S., Ferrucci, F., & Gravino, C. (2008). "Cross-company vs. single-company web effort models using the Tukutuku database: An extended study". *The Journal of systems and software*, v. 81, n. 5, pp. 673–690.

Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., & Fritz, T. (2017). “The Work Life of Developers: Activities, Switches and Perceived Productivity”. *IEEE Transactions on Software Engineering*, v. 43, n. 12, pp. 1178–1193.

Meyer, A. N., Fritz, T., Murphy, G. C., & Zimmermann, T. (2014). “Software Developers’ Perceptions of Productivity”. In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* pp. 19–29. New York, NY, USA: ACM.

Meyer, A. N., Zimmermann, T., & Fritz, T. (2017). “Characterizing Software Developers by Perceptions of Productivity”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM),* pp. 105–110.

Middleton, P., & Joyce, D. (2012). “Lean Software Management: BBC Worldwide Case Study”. *IEEE Transactions on Engineering Management*, v. 59, n.1, pp. 20–32.

Miller, J. (2000). “Applying meta-analytical procedures to software engineering experiments”. *The Journal of systems and software*, v. 54, n. 1, pp. 29–39.

Miller, J. (2005). “Replicating software engineering experiments: a poisoned chalice or the Holy Grail”. *Information and Software Technology*, v. 47, n. 4, pp. 233–244.

Mishra, B., & Shukla, K. K. (2013). “Data Mining Techniques for Software Quality Prediction”. *IGI Global*.

Mizuno, O. a., Kikuno, T. a., Inagaki, K. b., Takagi, Y. b., & Sakamoto, K. b. (2000). “Statistical analysis of deviation of actual cost from estimated cost using actual project data”. *Information and Software Technology*, v. 42, n. 7, pp. 465–473.

Moazeni, R., Link, D., & Boehm, B. (2014). “COCOMO II parameters and IDPD: bilateral relevances”. In: *Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014*. https://doi.org/10.1145/2600821.2600847

Mohapatra, S. (2011). “Maximising productivity by controlling influencing factors in commercial software development”. *International journal of information and communication technology education: an official publication of the Information Resources Management Association*, v. 3, n. 2, pp. 160-.

Molleri, J. S., Petersen, K., & Mendes, E. (2016). “Survey Guidelines in Software Engineering: An Annotated Review”. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM ’16* , pp. 1–6. New York, New York, USA: ACM Press.

Monteiro, L. F. S., & de Oliveira, K. M. (2011). “Defining a catalog of indicators to support process performance analysis”. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 23, n. 6, pp. 395–422.

Morasca, S., & Russo, G. (2001). “An empirical study of software productivity”. In: *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*. https://doi.org/10.1109/cmpsac.2001.960633

Morisio, M., Romano, D., & Stamelos, I. (2002). “Quality, productivity, and learning in framework-based development: an exploratory case study”. *IEEE Transactions on Software Engineering*, v. 28, n. 9, pp. 876–888.

Munch, J., & Armbrust, O. (2003). “Using empirical knowledge from replicated experiments for software process simulation: a practical example”. In *Proceedings of International Symposium on Empirical Software Engineering, 2003. ISESE 2003*. https://doi.org/10.1109/isese.2003.1237961

Munir, H., Runeson, P., & Wnuk, K. (2018). “A theory of openness for software engineering tools in software organizations”. *Information and Software Technology*, v. 97, pp. 26–45.

Myrtveit, I., & Stensrud, E. (1999). “Benchmarking COTS projects using data envelopment analysis”. In: *Proceedings Sixth International Software Metrics Symposium*, pp. 269–278.

Nguyen-Duc, A., Cruzes, D. S., & Conradi, R. (2015). “The impact of global dispersion on coordination, team performance and software quality--A systematic literature review”. *Information and Software Technology*, v. 57, pp. 277–294.

Nguyen, V., Huang, L., & Boehm, B. (2011). “An analysis of trends in productivity and cost drivers over years”. In: *Proceedings of the 7th International Conference on Predictive Models in Software Engineering - Promise ’11*. https://doi.org/10.1145/2020390.2020393

Otero, L. D., Centeno, G., Otero, C. E., & Reeves, K. (2012). “A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers”. *IEEE Transactions on Engineering Management*, v. 59, n.3, pp. 391–400.

Pai, D. R., Subramanian, G. H., & Pendharkar, P. C. (2015). “Benchmarking software development productivity of CMMI level 5 projects”. *Information Technology and Management*, v. 16, n. 3, pp. 235–251.

Pai, M., McCulloch, M., Gorman, J. D., Pai, N., Enanoria, W., Kennedy, G., Colford, J. M., Jr. (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.

Paiva, E., Barbosa, D., Lima, R., & Albuquerque, A. (2010). “Factors that Influence the Productivity of Software Developers in a Developer View”. In: *T. Sobh & K. Elleithy (Orgs.)*, *Innovations in Computing Sciences and Software Engineering*, pp. 99–104. Dordrecht: Springer Netherlands.

Parthasarathy, S., & Sharma, S. (2016). “Efficiency analysis of ERP packages - A customization perspective”. *Computers in Industry*, v. 82, pp. 19–27.

Petersen, K. (2011). “Measuring and predicting software productivity: A systematic map and review”. *Information and Software Technology*, v. 53, n.4, pp. 317–343.

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). “Guidelines for conducting systematic mapping studies in software engineering: An update”. *Information and Software Technology*, v. 64, pp. 1–18.

Poser, H. (1998). “On Structural Differences between Science and Engineering”. *Techné: Research in Philosophy and Technology*, v. 4, n. 2, pp. 128–135.

Prechelt, L. (2000). “An empirical comparison of seven programming languages”. *Computer*, n. 10, pp. 23–29.

Premraj, R., Shepperd, M., Kitchenham, B., & Forselius, P. (2005). “An Empirical Analysis of Software Productivity over Time”. In: *11th IEEE International Software Metrics Symposium (METRICS’05)* . https://doi.org/10.1109/metrics.2005.8

Price, D. D., Staud, R., & Robinson, M. E. (2012). “How should we use the visual analogue scale (VAS) in rehabilitation outcomes? II: Visual analogue scales as ratio scales: an alternative to the view of Kersten et al”. *Journal of Rehabilitation Medicine: Official Journal of the UEMS European Board of Physical and Rehabilitation Medicine*, v. 44, n. 9, pp. 800–.

Punter, T., Ciolkowski, M., Freimut, B., & John, I. (2003). Conducting on-line surveys in software engineering. In: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. p. 80–88.

Putnam, L. H., & Myers, W. (1992). “Measures for Excellence: Reliable Software on Time, Within Budget”. *Yourdon Press*.

Rahman, A., Stallings, J., & Williams, L. (2018). “Defect prediction metrics for infrastructure as code scripts in DevOps”. In: *Proceedings of the 40th International Conference on Software Engineering* , pp. 414–415. ACM.

Ralph, P. (2015). “Developing and Evaluating Software Engineering Process Theories”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* , v. 1, pp. 20–31.

Ralph, P. (2016). “Software engineering process theory: A multi-method comparison of Sensemaking–Coevolution–Implementation Theory and Function–Behavior–Structure Theory”. *Information and Software Technology*, v. 70, pp. 232–250.

Ramasubbu, N., & Balan, R. K. (2007). “Globally distributed software development project performance: An empirical analysis”. In: *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering,* pp. 125–134).

Ramasubbu, N., Bharadwaj, A., & Tayi, G. K. (2015). “Software Process Diversity: Conceptualization, Measurement, and Analysis of Impact on Project Performance”. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2627173

Ramasubbu, N., Cataldo, M., Balan, R. K., & Herbsleb, J. D. (2011). “Configuring global software teams”. In: *Proceeding of the 33rd international conference on Software engineering - ICSE ’11*. https://doi.org/10.1145/1985793.1985830

Ramil, J. F., & Lehman, M. M. (2001). “Defining and applying metrics in the context of continuing software evolution”. In: *Proceedings of Seventh International of Software Metrics Symposium*, *2001. METRICS 2001*. pp. 199–209.

Ramírez-Mora, S. L., & Oktaba, H. (2017). “Productivity in Agile Software Development: A Systematic Mapping Study”. In: *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT),* pp. 44–53.

Reips, U.-D., & Funke, F. (2008). “Interval-level measurement with visual analogue scales in Internet-based research: VAS Generator”. *Behavior Research Methods*, v. 40, n. 3, pp. 699–704.

Reynolds, P. D. (2015). “Primer in Theory Construction: An A&B Classics Edition”. Routledge.

Rodríguez, D., Sicilia, M. A., García, E., & Harrison, R. (2012). “Empirical findings on team size and productivity in software development”. *The Journal of systems and software*, v. 85, n. 3, pp. 562–570.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., *et al.* (2017). “Continuous deployment of software intensive products and services: A systematic mapping study”. *The Journal of systems and software*, v. 123, pp. 263–291.

Rothenberger, M. A., Kao, Y.-C., & Van Wassenhove, L. N. (2010). “Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects”. *Information & Management*, v. 47, n.7-8, pp. 372–379.

Santos, J. L. G. dos, Erdmann, A. L., Sousa, F. G. M. de, Lanzoni, G. M. de M., Melo, A. L. S. F. de, & Leite, J. L. (2016). “Perspectivas metodológicas para o uso da teoria fundamentada nos dados na pesquisa em enfermagem e saúde”. *Escola Anna Nery*, 20(3). https://doi.org/10.5935/1414-8145.20160056

Santos, P. S. M. dos, & G. H. Travassos. (2017). “Evidence of Usage-Based Reading Effects by Using the Structured Synthesis Method”. Relatório Técnico Nº. RT-ES 753/17. Universidade Federal do Rio de Janeiro (UFRJ/COPPE). Disponível em: <https://pdfs.semanticscholar.org/8509/ca8170c007672ef88752ecd2c0e4e5547053.pdf>.

Santos, P. S. M. dos. (2015). “EVIDENCE REPRESENTATION AND AGGREGATION IN SOFTWARE ENGINEERING USING THEORETICAL STRUCTURES AND BELIEF FUNCTIONS”. Tese de Doutorado. UFRJ/COPPE. Disponível em: <https://pdfs.semanticscholar.org/bd61/3f9daba33372fe5ed6faee3c2833a9e7ae4f.pdf>.

Santos, G., Kalinowski, M., Rocha, A. R., Travassos, G. H., Weber, K. C., & Antonioni, J. A. (2010). “MPS.BR: A Tale of Software Process Improvement and Performance Results in the Brazilian Software Industry”. In: *2010 Seventh International Conference on the Quality of Information and Communications Technology*. https://doi.org/10.1109/quatic.2010.75

Santos, J. L. G. D., Cunha, K., Adamy, E. K., Backes, M. T. S., Leite, J. L., & Sousa, F. G. M. de. (2018). “Data analysis: comparison between the different methodological perspectives of the Grounded Theory”. *Revista Da Escola de Enfermagem Da U S P*, 52, e03303.

Scacchi, W. (1991). “UNDERSTANDING SOFTWARE PRODUCTIVITY: TOWARDS A KNOWLEDGE-BASED APPROACH”. *International Journal of Software Engineering and Knowledge Engineering*, v. 01, n. 03, pp. 293–321.

Schluter, T., & Birkholzer, T. (2012). “Modeling and analysis of software development management as closed loop control”. In: *2012 International Conference on Software and System Process (ICSSP)*. https://doi.org/10.1109/icssp.2012.6225966

Scholtes, I., Mavrodiev, P., & Schweitzer, F. (2016). “From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects”. *Empirical Software Engineering*, v. 21, n.2, pp. 642–683.

SEI. (2018). “CMMI® for Development, Version 2.0”. Disponível em <https://cmmiinstitute.com/products/cmmi/cmmi-v2-products >.

Shafer, G. (1976). “A Mathematical Theory of Evidence”. Princeton University Press.

Shafer, G. (1990). “Perspectives on the theory and practice of belief functions”. *International journal of approximate reasoning: official publication of the North American Fuzzy Information Processing Society*. Disponível em: <https://core.ac.uk/download/pdf/82712963.pdf>.

Sharpe, J. L., & Cangussu, J. W. (2005). “A productivity metric based on statistical pattern recognition”. In: *Computer Software and Applications Conference*, *2005. COMPSAC 2005. 29th Annual International,* v. 1, pp. 59–64.

Shull, F., & Feldmann, R. L. (2008). “Building Theories from Multiple Evidence Sources”. In: *F. Shull, J. Singer, & D. I. K. Sjøberg (Orgs.), Guide to Advanced Empirical Software Engineering,* pp. 337–364. London: Springer London.

Siegal, S. (1956). *Nonparametric statistics for the behavioral sciences*. McGraw-hill.

Sjøberg, D. I. K., Dybå, T., Anda, B. C. D., & Hannay, J. E. (2008). “Building Theories in Software Engineering”. In: *F. Shull, J. Singer, & D. I. K. Sjøberg (Orgs.)*, *Guide to Advanced Empirical Software Engineering*, pp. 312–336. London: Springer London.

Soares De Mello, J. C. C. B., Angulo-Meza, L., Gomes, E. G., & Biondi Neto, L. (2005). “Mini Curso de Análise Envoltória de Dados”. *Apresentado em XXXVII Simpósio Brasileiro de Pesquisa Operacional*. Disponível em: <https://www.researchgate.net/profile/Joao_Mello/publication/237473886_CUR

SO_DE_ANALISE_DE_ENVOLTORIA_DE_DADOS/links/0deec5226afdc4f679000000/CURSO-DE-ANALISE-DE-ENVOLTORIA-DE-DADOS.pdf>.

SOFTEX. (2016). “MPS.BR - Melhoria de Processo do Software Brasileiro: Guia Geral MPS de Software. MPS.BR - Melhoria de Processo do Software Brasileiro: Guia Geral MPS de Software”. Disponível em: <http://www.softex.br/wp-content/uploads/2016/04/MPS.BR_Guia_Geral_Software_2016-com-ISBN.pdf>. Acesso em 16 de novembro de 2018.

Stol, K.-J., & Fitzgerald, B. (2015). “Theory-oriented software engineering”. *Science of Computer Programming*, v. 101, pp. 79–98.

Strat, T. M. (1994). “Advances in the Dempster-Shafer Theory of Evidence”. In: *R. R. Yager, J. Kacprzyk, & M. Fedrizzi (Orgs.)* (p. 275–309). New York, NY, USA: John Wiley & Sons, Inc.

Strauss, A., & Corbin, J. M. (2015). “Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory”. *SAGE Publications*.

Suppe, F. (1977). “The Structure of Scientific Theories”*. University of Illinois Press*.

Tan, T., Li, Q., Boehm, B., Yang, Y., He, M., & Moazeni, R. (2009). “Productivity trends in incremental and iterative software development”. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1109/esem.2009.5316044

Teasley, S. D. b., Covi, L. A. a. c., Krishnan, M. S. a. d., & Olson, J. S. e. (2002). “Rapid software development through team collocation”. *IEEE Transactions on Software Engineering*, v. 28, n. 7, pp. 671–683.

Thornberg, R., & Charmaz, K. (2014). “Grounded Theory and Theoretical Coding”. In: *U. Flick*, *The SAGE Handbook of Qualitative Data Analysis,* pp. 153–169. 1 Oliver’s Yard, 55 City Road, London EC1Y 1SP United Kingdom: SAGE Publications Ltd.

Torchiano, M., & Ricca, F. (2013). “Six reasons for rejecting an industrial survey paper”. In: *2013 1st International Workshop on Conducting Empirical Studies in Industry* (CESI), pp. 21–26.

Travassos, G. H., d. Santos, P. S. M., Mian, P. G., Neto, A. C. D., & Biolchini, J. (2008). “An Environment to Support Large Scale Experimentation in Software Engineering”. In: *13th IEEE International Conference on Engineering of Complex Computer Systems* (iceccs 2008), pp. 193–202.

Trendowicz, A., & Münch, J. (2009). “Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences”. In: *Advances in Computers* , v. 77, pp. 185–241. Elsevier.

Tsunoda, M., & Amasaki, S. (2017). “On Software Productivity Analysis with Propensity Score Matching”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* , pp. 436–441.

Tsunoda, M., Monden, A., Yadohisa, H., Kikuchi, N., & Matsumoto, K. (2009). “Software development productivity of Japanese enterprise applications”. *INFORMATION TECHNOLOGY & MANAGEMENT*, 10(4), pp. 193–205.

Tsunoda, M., & Ono, K. (2014). “Pitfalls of analyzing a cross-company dataset of software maintenance and support”. In: *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. https://doi.org/10.1109/snpd.2014.6888729

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., & Filkov, V. (2015). “Quality and Productivity Outcomes Relating to Continuous Integration in GitHub”. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 805–816. New York, NY, USA.

Wagner, S., & Ruhe, M. (2018). A Systematic Review of Productivity Factors in Software Development. arXiv [cs.SE]. Disponível em: <http://arxiv.org/abs/1801.06475>.

Wahaballa, A., Wahballa, O., Abdellatief, M., Xiong, H., & Qin, Z. (2015). “Toward unified DevOps model”. In: *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)* , pp. 211–214.

Wang, H., Wang, H., & Zhang, H. (2008). “Software Productivity Analysis with CSBSG Data Set”. In: *2008 International Conference on Computer Science and Software Engineering*. https://doi.org/10.1109/csse.2008.1178

Weber, K. C., Araújo, E. E. R., da Rocha, A. R. C., Machado, C. A. F., Scalet, D., & Salviano, C. F. (2005). “Brazilian Software Process Reference Model and Assessment Method”. In: *P. Yolum, T. Güngör, F. Gürgen, & C. Özturan*, *Computer and Information Sciences - ISCIS 2005* , v. 3733, pp. 402–411.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). “Experimentation in Software Engineering”. *Springer Science & Business Media*.

Wohlin, C., Šmite, D., & Moe, N. B. (2015). “A general theory of software engineering: Balancing human, social and organizational capitals”. *The Journal of systems and software*. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121215001740>.

Xu, P. a., & Yao, Y. b. (2013). “Knowledge sharing in offshore software development a vendor perspective”. *Journal of Global Information Technology Management*, v. 16, n. 1, pp. 58–84.

Yilmaz, M., O’Connor, R. V., & Clarke, P. (2016). “Effective Social Productivity Measurements during Software Development — An Empirical Study”. *International Journal of Software Engineering and Knowledge Engineering*, v. 26, n. 03, pp. 457–490.

Zapf, A., Castell, S., Morawietz, L., & Karch, A. (2016). “Measuring inter-rater reliability for nominal data - which coefficients and confidence intervals are appropriate?” *BMC Medical Research Methodology*, v. 16, pp. 93-.

APÊNDICE I: Artigos Selecionados no Protocolo de PETERSEN (2011)

Arnold, M., Pedross, P., Software size measurement and productivity rating in a large-scale software development department, In: Proceedings of the 20th International Conference on Software Engineering (ICSE 1998), IEEE Computer Society, 1998, pp. 490–493.

Asmild, M., Paradi, J.C., Kulkarni, A., Using data envelopment analysis in software development productivity measurement, Software Process Improvement and Practice 11 (6) (2006) 561–572.

Baldassarre, M.T., Boffoli, N., Caivano, D., Visaggio, G., Improving dynamic calibration through statistical process control, in: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005), 2005, pp. 273–282.

Banker, R.D., Datar, S.M., Kemerer, C.F., Model to evaluate variables impacting the productivity of software maintenance projects, Management Science 37 (1) (1991) 1–18

Bok, H.S., Raman, K., Software engineering productivity measurement using function points: a case study, Journal of Information Technology 15 (2000) 79– 90.

Chatman, V., Change-points: a proposal for software productivity measurement, Journal of Systems and Software 31 (1) (1995) 71–91.

Donzelli, P., Iazeolla, G., A hybrid software process simulation model, Software Process: Improvement and Practice 6 (2) (2001) 97–109.

Donzelli, P., Iazeolla, G., A software process simulator for software product and process improvement, in: Proceedings of the 1st International Conference on Product Focused Software Process Improvement (PROFES 1999), 1999, pp. 525–538

Finnie, G.R., Wittig, G.E., Effect of system and team size on 4gl software development productivity, South African Computer Journal (11) (1994) 18–25.

Foulds, L.R., Quaddus, M., West, M., Structural equation modeling of large-scale information system application development productivity: the hong kong

experience, in: Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ACIS-ICIS 2007), 2007, pp. 724–731.

Hanakawa, N., Morisaki, S., Matsumoto, K., A learning curve based simulation model for software development, in: Proceedings of the 20th International Conference on Software Engineering (ICSE 1998), 1998, pp. 350–359.

Humphrey, W.S., Singpurwalla, N.D., Predicting (individual) software productivity, IEEE Transactions on Software Engineering 17 (2) (1991) 196– 207.

Kadary, V., On application of earned value index to software productivity metrics in embedded computer systems, in: Proceedings of the Conference on Computer Systems and Software Engineering (CompEuro 92), 1992, pp. 666– 670.

Khosrovian, K., Pfahl, D., Garousi, V., Gensim 2.0: a customizable process simulation model for software process evaluation, in: Proceedings of the International Conference on Software Process (ICSP 2008), 2008, pp. 294–306

Kitchenham, B.A., Jeffery, D.R., Connaughton, C., Misleading metrics and unsound analyses, IEEE Software 24 (2) (2007) 73–78.

Kitchenham, B., Mendes, E., Software productivity measurement using multiple size measures, IEEE Transactions on Software Engineering 30 (12) (2004) 1023–1035.

Li, R., Yongji, W., Qing, W., Fengdi, S., Haitao, Z., Shen, Z., Arimammse: an integrated arima-based software productivity prediction method, in: Proceedings of the 30th International Conference on Computer Software and Applications Conference (COMPSAC 2006), vol. 2, 2006, pp. 135–138.

Lin, C.Y., Abdel-Hamid, T., Sherif, J.S., Software-engineering process simulation model (seps), Journal of Systems and Software 38 (3) (1997) 263–277.

Liping, D., Qiusong, Y., Liang, S., Jie, T., Yongji, W., Evaluation of the capability of personal software process based on data envelopment analysis, in: Proceedings of the International Software Process Workshop (SPW 2005), Springer-Verlag, Berlin, Germany, 2005.

List, B., Bruckner, R.M., Kapaun, J., Holistic software process performance measurement from the stakeholders' perspective, in: Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 2005, pp. 941–947.

Yang, Z.J., Paradi, J.C., Dea evaluation of a y2k software retrofit program, IEEE Transactions on Engineering Management 51 (3) (2004) 279–287.

Jeffery, D.R., Software development productivity model for mis environments, Journal of Systems and Software 7 (2) (1987) 115–125.

Mahmood,M.A., Pettingell, K.J., Shaskevich, A.L., Measuring productivity of software projects: a data envelopment approach, Decision Sciences 27 (1) (1996) 57–81.

Maxwell, K., Wassenhove, L.V., Dutta,S., Software development productivity of european space, military, and industrial applications, IEEE Transactions on Software Engineering 22 (10) (1996) 706–718.

Morasca, S., Russo, G., An empirical study of software productivity, In: Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001), 2001, pp. 317–322.

Münch, J., Armbrust, O., Using empirical knowledge from replicated experiments for software process simulation: A practical example, in: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2003), 2003, pp. 18–27.

Numrich, L. R.W., Hochstein, V.R., Basili, V.R., A metric space for productivity measurement in software development, in: Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications (SE-HPCS 2005), ACM, New York, NY, USA, 2005, pp. 13–16.

Pfleeger, S.L., Model of software effort and productivity, Information and Software Technology 33 (3) (1991) 224–231.

Raffo, D., Evaluating the impact of process improvements quantitatively using process modeling, in: Proceedings of the Conference of the Center for Advanced Studies on Collaborative research (CASCON 1993), IBM Press, 1993, pp. 290–313.

Raffo, D., Harrison, W., Vandeville, J., Coordinating models and metrics to manage software projects, Software Process: Improvement and Practice 5 (2– 3) (2000) 159–168.

Raffo, D.M., Harrison, W., Vandeville, J., Software process decision support: making process tradeoffs using a hybrid metrics, modeling and utility framework, in:

Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM, New York, NY, USA, 2002, pp. 803– 809.

Raffo, D., Vandeville, J., Martin, R.H., Software process simulation to achieve higher cmm levels, Journal of Systems and Software 46 (2–3) (1999) 163– 172.

Romeu, J.L., A simulation approach for the analysis and forecast of software productivity, Computers and Industrial Engineering 9 (2) (1985) 165–174.

Ruan, L., Wang, Y., Wang, Q., Li, M., Yang, Y., Xie, L., Liu, D., Zeng, H., Zhang, S., Xiao, J., Zhang, L., Nisar, M.W., Dai, J., Empirical study on benchmarking software development tasks, in: Proceedings of the International Conference on Software Process (ICSP 2007), 2007, pp. 221–232.

Stensrud, E., Myrtveit, I., Identifying high performance erp projects, IEEE Transactions on Software Engineering 29 (5) (2003) 398–416.

Yu, W.D., Smith, D.P., Huang, S.T., Software productivity measurements, In: Proceedings of the 15th International Conference on Computer Software and Applications Conference (COMPSAC 1991), 1991, pp. 558–564.

APÊNDICE II: Artigos Selecionados na Repetição de PETERSEN (2011)

Ahmed A, Ahmad S, Ehsan N, Mirza E and Sarwar S (2010), "Agile software development: Impact on productivity and quality", 5th IEEE International Conference on Management of Innovation and Technology, ICMIT2010. Singapore, Singapore, pp. 287 - 291.

Ahmedshareef Z, Petridis M and Hughes R (2013), "Empirical Examination of Internal Dynamics of Software Project Management: Mixed Methods Approach", In PROCEEDINGS OF THE 12TH EUROPEAN CONFERENCE ON RESEARCH METHODOLOGY FOR BUSINESS AND MANAGEMENT STUDIES. , pp. 365-374.

de Aquino Junior GS and de Lemos Meira SR (2009), "An Approach to Measure Value-Based Productivity in Software Projects", In 2009 NINTH INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE (QSIC 2009). , pp. 383-389.

Avritzer A and Lima A (2009), "An empirical approach for the assessment of scheduling risk in a large globally distributed industrial software project", Proceedings - 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009. Limerick, Ireland, pp. 341-346.

Bibi S, Stamelos I and Angelis L (2008), "Combining probabilistic models for explanatory productivity estimation", INFORMATION AND SOFTWARE TECHNOLOGY., 2008. Vol. 50(7-8), pp. 656-669.

Card DN (1987), "SOFTWARE TECHNOLOGY EVALUATION PROGRAM.", Information and Software Technology. https://doi.org/10.1016/0950-5849(87)90028-0.

Celik N, Lee S, Mazhari E, Son Y-J, Lemaire R and Provan KG (2011), "Simulation-based workforce assignment in a multi-organizational social network for alliance-based software development", Simulation Modelling Practice and Theory. P.O. Box 211, Amsterdam, 1000 AE, Netherlands Vol. 19(10), pp. 2169-2188.

Choi K and Bae D-H (2009), "Dynamic project performance estimation by combining static estimation models with system dynamics", INFORMATION AND SOFTWARE TECHNOLOGY., JAN 2009. Vol. 51(1), pp. 162-172.

Czekster RM, Fernandes P, Sales A and Webber T (2010), "Analytical modeling of software development teams in globally distributed projects", Proceedings - 5th International Conference on Global Software Engineering, ICGSE 2010. Princeton, NJ, United states, pp. 287-296.

Di Tullio D and Bahli B (2006), "The impact of software process maturity and software development risk on the performance of software development projects", ICIS 2006 Proceedings - Twenty Seventh International Conference on Information Systems. , pp. 1479-1490.

Ge Y, Chang CK and Jiang H-y (2006), "Modeling software project scheduling based on team productivity and its simulations", In DCABES 2006 Proceedings, Vols 1 and 2., pp. 1259-1262.

Gencel C (2008), "How to Use COSMIC Functional Size in Effort Estimation Models?", In SOFTWARE PROCESS AND PRODUCT MEASUREMENT. Vol. 5338, pp. 196- 207.

Gopal A and Gosain S (2010), "The Role of Organizational Controls and Boundary Spanning in Software Development Outsourcing: Implications for Project Performance", INFORMATION SYSTEMS RESEARCH, DEC, 2010. Vol. 21(4), pp. 960-982.

Graziotin D, Wang X and Abrahamsson P (2013), "Are happy developers more productive? The correlation of affective states of software developers and their self-assessed productivity", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Paphos, Cyprus Vol. 7983 LNCS, pp. 50 - 64.

Jeet K, Bhatia N and Minhas RS (2011), "A Model for Estimating the Impact of Low Productivity on the Schedule of a Software Development Project", SIGSOFT Softw. Eng. Notes. New York, NY, USA, August 2011. Vol. 36(4), pp. 1-6. ACM.

Jeffery D (1987), "A software development productivity model for MIS environments", The Journal of Systems and Software. Vol. 7(2), pp. 115-125.

Kang D, Jung J and Bae D-H (2011), "Constraint-based human resource allocation in software projects", Software - Practice and Experience. Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom Vol. 41(5), pp. 551 - 577.

Krishnan M, Kriebel C, Kekre S and Mukhopadhyay T (2000), "An empirical analysis of productivity and quality in software products", MANAGEMENT SCIENCE., JUN 2000. Vol. 46(6), pp. 745-759.

Kuchar t and Vondrak I (2013), "Simulating human resource capability and productivity in software process simulations", 25th European Modeling and Simulation Symposium, EMSS 2013. Athens, Greece, pp. 255-263.

Lee G and Xia W (2010), "TOWARD AGILE: AN INTEGRATED ANALYSIS OF QUANTITATIVE AND QUALITATIVE FIELD DATA ON SOFTWARE DEVELOPMENT AGILITY", MIS QUARTERLY., MAR 2010. Vol. 34(1), pp. 87-114.

Liang T-P, Wu JC-H, Jiang JJ and Klein G (2012), "The impact of value diversity on information system development projects", INTERNATIONAL JOURNAL OF PROJECT MANAGEMENT., AUG 2012. Vol. 30(6, SI), pp. 731-739.

Lopez-MARTIN C, Chavoya A and Meda-CAMPANA ME (2013), "Software development productivity prediction of individual projects applying a neural network", IMETI 2013 - 6th International Multi-Conference on Engineering and Technological Innovation, Proceedings. Orlando, FL, United States, pp. 47 – 52.

Low, G. C., Jeffery, D. R., Function points in the estimation and evaluation of the software process, IEEE Transactions on Software Engineering, vol. 16, no. 1, pp. 64-71, 1990.

Mohapatra S (2011), "Maximising productivity by controlling influencing factors in commercial software development", International Journal of Information and Communication Technology. P.O.Box 735, Olney, Bucks, MK46 5WB, United Kingdom Vol. 3(2), pp. 160 - 179.

Monteiro L and De Oliveira K (2011), "Defining a catalog of indicators to support process performance analysis", Journal of Software Maintenance and Evolution. Vol. 23(6), pp. 395-422.

Nguyen V, Huang L and Boehm B (2011), "An analysis of trends in productivity and cost drivers over years", ACM International Conference Proceeding Series.

Peck C and Callahan D (2002), "A proposal for measuring software productivity in a working environment", In PROCEEDINGS OF THE THIRTY-FOURTH SOUTHEASTERN SYMPOSIUM ON SYSTEM THEORY. , pp. 339-343.

Potok T and Vouk M (1997), "Effects of the business model on object-oriented software development productivity", IBM Systems Journal. Armonk, NY, United States Vol. 36(1), pp. 140 - 161.

Prernraj R, Shepperd M, Kitchenham B and Forsellus P (2005), "An empirical analysis of software productivity over time", In 2005 11th International Symposium on Software Metrics (METRICS). , pp. 338-347.

Rodriguez D, Sicilia MA, Garcia E and Harrison R (2012), "Empirical findings on team size and productivity in software development", JOURNAL OF SYSTEMS AND SOFTWARE., MAR 2012. Vol. 85(3), pp. 562-570.

Rothenberger MA, Kao Y-C and Van Wassenhove LN (2010), "Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects", Information and Management. P.O. Box 211, Amsterdam, 1000 AE, Netherlands Vol. 47(7-8), pp. 372 - 379.

Tan T, Li Q, Boehm B, Yang Y, He M and Moazeni R (2009), "Productivity trends in incremental and iterative software development", 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009. Lake Buena Vista, FL, United states, pp. 1 - 10.

Yang Z and Paradi J (2009), "A DEA evaluation of software project efficiency", IEEM 2009 - IEEE International Conference on Industrial Engineering and Engineering Management. Hong Kong, China, pp. 1723 – 1727.

APÊNDICE III: Artigos Selecionados na Evolução de PETERSEN (2011)

Anderson T and Ghavami P (1999), "Using data envelopment analysis for evaluating alternative software development process configurations", In Management of Engineering and Technology, 1999. Technology and Innovation Management. PICMET '99. Portland International Conference on. Vol. 1, pp. 298 vol.1-.

Ardagna C, Damiani E, Frati F, Oltolina S, Regoli M and Ruffatti G (2010), "Spago4Q and the QEST nD Model: An open source solution for software performance measurement", IFIP Advances in Information and Communication Technology. Vol. 319 AICT, pp. 1-14.

Banker R and Slaughter S (1997), "A field study of scale economies in software maintenance", Management Science. Vol. 43(12), pp. 1709-1725.

Bush M (1990), "Getting started on metrics - Jet Propulsion Laboratory productivity and quality", Proceedings - International Conference on Software Engineering. (12), pp. 133-142.

Chen P-C, Chern C-C and Chen C-Y (2012), "Software Project Team Characteristics and Team Performance: Team Motivation as a Moderator", In Software Engineering Conference (APSEC), 2012 19th Asia-Pacific., Dec 2012. Vol. 1, pp. 565-570.

Chinubhai A (2011), "Efficiency in software development projects", International Journal of Software Engineering and its Applications. Vol. 5(4), pp. 171-180.

Colazo J (2010), "Collaboration structure and performance in new software development: Findings from the study of open source projects", International Journal of Innovation Management. Vol. 14(5), pp. 735-758.

Colazo J (2008), "Following the sun: Exploring productivity in temporally dispersed teams", 14th Americas Conference on Information Systems, AMCIS 2008. Vol. 3, pp. 1833-1839.

DeToma D and Perry J (1994), "A software metrics program", In Electro/94 International. Conference Proceedings. Combined Volumes.., May, 1994. , pp. 230-238.

Farshchi M, Jusoh Y and Murad M (2012), "Impact of personnel factors on the recovery of delayed software projects: A system dynamics approach", Computer Science and Information Systems. Vol. 9(2), pp. 627-651.

Flitman A (2003), "Towards meaningful benchmarking of software development team productivity", Benchmarking. Vol. 10(4), pp. 382-399.

Florac W, Carleton A and Barnard J (2000), "Statistical process control: analyzing space shuttle onboard software process", Software, IEEE, Jul, 2000. Vol. 17(4), pp. 97-106.

Fujii T and Kimura M (2011), "Analysis Results on Productivity Variation in Force.com Applications", In Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)., Nov, 2011. , pp. 314-317.

Georgieva K, Neumann R, Fiegler A and Dumke R (2011), "Validation of the Model for Prediction of the Human Performance", In Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)., Nov, 2011. , pp. 245-250.

Ghapanchi A and Aurum A (2012), "The impact of project capabilities on project performance: Case of open source software projects", International Journal of Project Management. Vol. 30(4), pp. 407-417.

Ghapanchi AH and Aurum A (2012), "Competency rallying in electronic markets: implications for open source project success", ELECTRONIC MARKETS., JUN 2012. Vol. 22(2), pp. 117-127.

Gunsel A and Açikgöz A (2013), "The Effects of Team Flexibility and Emotional Intelligence on Software Development Performance", Group Decision and Negotiation. Vol. 22(2), pp. 359-377.

Hayes W (1998), "Using a Personal Software ProcessSM to improve performance", In Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International, Nov 1998. , pp. 61-71.

He Mb, Yang Y, Wang Q and Li M (2008), "An investigation on performance of software enhancement projects in china", Proceedings - Asia-Pacific Software Engineering Conference, APSEC. , pp. 67-74.

Hsu JS-C and Hung YW (2013), "Exploring the interaction effects of social capital", INFORMATION & MANAGEMENT., NOV 2013. Vol. 50(7), pp. 415-430.

Huang H, Yang Q, Xiao J and Zhai J (2011), "Automatic Mining of Change Set Size Information from Repository for Precise Productivity Estimation", In Proceedings of the 2011 International Conference on Software and Systems Process. New York, NY, USA, pp. 72-80. ACM.

Huawei Z and Gang D (2011), "The evaluation to regional development efficiency of software industry in China", In Emergency Management and Management Sciences (ICEMMS), 2011 2nd IEEE International Conference on, Aug, 2011. , pp. 644-648.

Hwang S-M and Kim H-M (2005), "A study on metrics for supporting the software process improvement based on SPICE", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 3647 LNCS, pp. 71-80.

Koch S (2006), "ERP implementation effort estimation using data envelopment analysis", Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI). , pp. 164-178.

Liu JY-C, Chen H-G, Chen CC and Sheu TS (2011), "Relationships among interpersonal conflict, requirements uncertainty, and software project performance", INTERNATIONAL JOURNAL OF PROJECT MANAGEMENT., JUL 2011. Vol. 29(5, SI), pp. 547-556.

MacCormack A, Kemerer C, Cusumano M and Crandall B (2003), "Trade-offs between productivity and quality in selecting software development practices", Software, IEEE., Sept, 2003. Vol. 20(5), pp. 78-85.

Marczak S and Gomes V (2013), "On the development of a theoretical model of the impact of trust in the performance of distributed software projects", In Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, May, 2013., pp. 97-100.

Maxwell K and Forselius P (2000), "Benchmarking software development productivity", Software, IEEE, Jan, 2000. Vol. 17(1), pp. 80-88.

Mizuno O, Kikuno T, Inagaki K, Takagi Y and Sakamoto K (2000), "Statistical analysis of deviation of actual cost from estimated cost using actual project data", Information and Software Technology. Vol. 42(7), pp. 465-473.

Mockus A, Fielding R and Herbsleb J (2000), "A case study of open source software development: the Apache server", In Software Engineering, 2000. Proceedings of the 2000 International Conference on. , pp. 263-272.

Morisio M, Romano D and Stamelos I (2002), "Quality, productivity, and learning in framework-based development: an exploratory case study", Software Engineering, IEEE Transactions on, Sep, 2002. Vol. 28(9), pp. 876-888.

Myrtveit I and Stensrud E (1999), "Benchmarking COTS projects using data envelopment analysis", International Software Metrics Symposium, Proceedings. , pp. 269-278.

de Oliveira SB, Valle R and Mahler CF (2010), "A comparative analysis of CMMI software project management by Brazilian, Indian and Chinese companies", SOFTWARE QUALITY JOURNAL., JUN, 2010. Vol. 18(2), pp. 177-194.

Ramasubbu N and Balan R (2007), "Globally distributed software development project performance: An empirical analysis", 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2007., pp. 125-134.

Ramasubbu N, Cataldo M, Balan R and Herbsleb J (2011), "Configuring global software teams: a multi-company analysis of project productivity, quality, and profits", In Software Engineering (ICSE), 2011 33rd International Conference on, 2011. , pp. 261-270.

Ramil J and Lehman M (2001), "Defining and applying metrics in the context of continuing software evolution", In Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International. , pp. 199-209.

Rodrigues Ac and Williams T (1998), "System dynamics in project management: Assessing the impacts of client behaviour on project performance", Journal of the Operational Research Society. Vol. 49(1), pp. 2-15.

Rodrigues A. c.guez D, Ruiz M, Riquelme J and Harrison R (2011), "Multiobjective simulation optimisation in software project management", Genetic and Evolutionary Computation Conference, GECCO'11. , pp. 1883-1890.

Sakamoto K, Niihara N, Tanaka T, Nakakoji K and Kishida K (1996), "Analysis of software process improvement experience using the project visibility index", In Software Engineering Conference, 1996. Proceedings, 1996 Asia-Pacific., Dec, 1996. , pp. 139-148.

Santos G, Kalinowski M, Rocha A, Travassos G, Weber K and Antonioni J (2010), "MPS.BR: A Tale of Software Process Improvement and Performance Results in the Brazilian Software Industry", In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the., Sept, 2010., pp. 412-417.

Schalken J, Brinkkemper S and Van Vliet H (2006), "Using linear regression models to analyse the effect of software process improvement", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 4034 LNCS, pp. 234-248.

Schluter T and Birkholzer T (2012), "Modeling and analysis of software development management as closed loop control", In Software and System Process (ICSSP), 2012 International Conference on., June, 2012. , pp. 210-214.

Sharpe J and Cangussu J (2005), "A productivity metric based on statistical pattern recognition", In Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, July, 2005. Vol. 1, pp. 59-64 Vol. 2.

Siok M and Tian J (2011), "Benchmarking Embedded Software Development Project Performance", In High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on, Nov 2011. , pp. 277-284.

Sneed H (1997), "Measuring the performance of a software maintenance department", In Software Maintenance and Reengineering, 1997. EUROMICRO 97, First Euromicro Conference on., Mar 1997., pp. 119-127.

de Souza Carvalho W, Rosa P, Dos Santos Soares M, Teixeira da Cunha Junior M and Buiatte L (2011), "A Comparative Analysis of the Agile and Traditional Software Development Processes Productivity", In Computer Science Society (SCCC), 2011 30th International Conference of the Chilean., Nov 2011. , pp. 74-82.

Teasley S, Covi Lc, Krishnan Md and Olson J (2002), "Rapid software development through team collocation", IEEE Transactions on Software Engineering. Vol. 28(7), pp. 671- 683.

Trammell T, Madnick S and Moulton A (2013), "Using system dynamics to analyze the effect of funding fluctuations on software development", International Annual Conference of the American Society for Engineering Management 2013, ASEM 2013. , pp. 79-89.

Tsunoda M, Monden A, Yadohisa H, Kikuchi N and Matsumoto K (2009), "Software development productivity of Japanese enterprise applications", INFORMATION TECHNOLOGY & MANAGEMENT., DEC, 2009. Vol. 10(4), pp. 193-205.

Von Mayrhauser A, Wohlin C and Ohlsson M (2000), "Assessing and understanding efficiency and success of software production", Empirical Software Engineering. Vol. 5(2), pp. 125-154.

Wang H, Wang H and Zhang H (2008), "Software Productivity Analysis with CSBSG Data Set", In Computer Science and Software Engineering, 2008 International Conference on., Dec, 2008. Vol. 2, pp. 587-593.

Wray B and Mathieu R (2008), "Evaluating the performance of open source software projects using data envelopment analysis", Information Management and Computer Security. Vol. 16(5), pp. 449-462.

Xu P and Yao Y (2013), "Knowledge sharing in offshore software development a vendor perspective", Journal of Global Information Technology Management. Vol. 16(1), pp. 58-84.

Yilmaz M and O'Connor R (2012), "Social capital as a determinant factor of software development productivity: An empirical study using structural equation modeling", International Journal of Human Capital and Information Technology Professionals. Vol. 3(2), pp. 40-62.

Yilmaz S and Tarhan A (2011), "Systematic analysis of the incremental process as a base for comparison with the Agile process", In Evaluation Assessment in Software Engineering (EASE 2011), 15th Annual Conference on., April, 2011., pp. 86-90.

Yun SJ and Simmons D (2004), "Continuous productivity assessment and effort prediction based on Bayesian analysis", In Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International., Sept 2004, pp. 44-49 vol.1.

Zhou M and Mockus A (2010), "Developer fluency: Achieving true mastery in software projects", Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering., pp. 137-146.

APÊNDICE IV: Artigos Selecionados no *Snowballing*

Narayanan S, Swaminathan JM, Talluri S. Knowledge Diversity, Turnover, and Organizational-Unit Productivity: An Empirical Analysis in a Knowledge-Intensive Context. Prod Oper Manag. 2014;23: 1332–1351.

Otero LD, Centeno G, Otero CE, Reeves K. A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers. IEEE Trans Eng Manage. 2012;59: 391–400.

Moazeni R, Link D, Boehm B. COCOMO II parameters and IDPD: bilateral relevances. Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014. 2014. doi:10.1145/2600821.2600847

Yilmaz M, O'Connor RV, Clarke P. Effective Social Productivity Measurements during Software Development — An Empirical Study. Int J Software Engineer Knowledge Engineer. World Scientific Publishing Co.; 2016;26: 457–490.

Lopez-Martin C, Chavoya A, Meda-Campaña ME. A machine learning technique for predicting the productivity of practitioners from individually developed software projects. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). 2014. pp. 1–6.

Parthasarathy S, Sharma S. Efficiency analysis of ERP packages—A customization perspective. Comput Ind. 2016;82: 19–27.

Kamma D, G SK. Effect of Model Based Software Development on Productivity of Enhancement Tasks -- An Industrial Study. 2014 21st Asia-Pacific Software Engineering Conference. 2014. pp. 71–77.

Czekster RM, Fernandes P, Lopes L, Sales A, Santos AR, Webber T. Stochastic Performance Analysis of Global Software Development Teams. ACM Trans Softw Eng Methodol. New York, NY, USA: ACM; 2016;25: 26:1–26:32.

Xu Y, Yeh C-H. A performance-based approach to project assignment and performance evaluation. Int J Project Manage. 2014;32: 218–228.

Scholtes I, Mavrodiev P, Schweitzer F. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. Empirical Software Engineering. 2016;21: 642–683.

Liu L, Kong X, Chen J. How project duration, upfront costs and uncertainty interact and impact on software development productivity? A simulation approach. IJASM. 2015;8: 39.

Tsunoda M, Ono K. Pitfalls of analyzing a cross-company dataset of software maintenance and support. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). 2014. pp. 1–6.

Eveleens L, Kampstra P, Verhoef C. Quantifying fair payment after outsourcing - a case study. J Softw Evol and Proc. 2015; 27: 147–165.

Sudhaman P, Thangavel C. Efficiency analysis of ERP projects—software quality perspective. Int J Project Manage. 2015; 33: 961–970.

Ramasubbu N, Bharadwaj A, Tayi GK. Software Process Diversity: Conceptualization, Measurement, and Analysis of Impact on Project Performance. MISQ. 2015;39: 787–807.

Shao BBM, Yin P-Y, Chen ANK. Organizing knowledge workforce for specified iterative software development tasks. Decis Support Syst. 2014;59: 15–27.

Duarte CHC. Productivity paradoxes revisited. Empirical Software Engineering. 2016;22: 818–847.

Bhowmik T, Niu N, Wang W, Cheng J-RC, Li L, Cao X. Optimal Group Size for Software Change Tasks: A Social Information Foraging Perspective. IEEE Trans Cybern. 2016;46: 1784–1795.

Azzeh M, Nassif AB, Banitaan S. An Application of Classification and Class Decomposition to Use Case Point Estimation Method. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). 2015. doi:10.1109/icmla.2015.105

Huang J, Sun H, Li Y-F. An empirical study of the impact of project factors on software economics. 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). 2015. doi:10.1109/ieem.2015.7385605

Simões C, Montoni M. Applying statistical process control in small sized evolutionary projects: results and lessons learned in the implementation of CMMI-DEV maturity level 5 in Synapsis Brazil. Journal of Software Engineering Research and Development. 2014.

Hsu JS-C, Hung YW. Exploring the interaction effects of social capital. INFORMATION & MANAGEMENT. 2013;50: 415–430.

Carrillo de Gea JM, de Gea JMC, Nicolás J, Fernández Alemán JL, Toval A, Ouhbi S, et al. Co-located and distributed natural-language requirements specification: traditional versus reuse-based techniques. Journal of Software: Evolution and Process. 2016;28: 205–227.

Colomo-Palacios R, Casado-Lumbreras C, Soto-Acosta P, García-Peñalvo FJ, Tovar E. Project managers in global software development teams: a study of the effects on productivity and performance. Software Quality Journal. 2013;22: 3–19.

Meyer AN, Fritz T, Murphy GC, Zimmermann T. Software Developers' Perceptions of Productivity. Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY, USA: ACM; 2014. pp. 19–29.

Fagerholm F, Guinea AS, Münch J, Borenstein J. The role of mentoring and project characteristics for onboarding in open source software projects. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14. 2014. doi:10.1145/2652524.2652540

APÊNDICE V: Memorandos de Códigos Encontrados

Code/Factor: Artifact Complexity

Definition:

It intends to observe empirically the complexity of software artifacts (document, models, resource, functionalities).

Theoretical Basis:

The understanding of artifact complexity can improve the understanding of its influence on software productivity. It may include observations about the complexity nature of software systems, concerning control operations, computational operations, device-dependent operations, data management operations, and the user interface management operations (NGUYEN et al., 2011).

Viewpoints

A complex artifact may be very difficult to understand and, therefore, manipulating any of its parts may be hard. Moreover, it may lead to additional issues or false positive defects than a very simple one (MUNCH & ARMBRUST, 2003). This way, the complexity of software artifacts can affect software productivity.

Measures found in studies:

Name	Specification
Application Complexity (Mohapatra, 2011)	An application is defined to be complex, if the project manager considers the required functionalities. Along with that, functional coupling and cohesion would also contribute to observe such complexity. The project manager perception can be captured, for instance, using a scale of 1 to 10, with 10 being highly complex and 1 being the least complex.
Project Complexity (Rothenberger et al., 2010)	The 3 items below are scored on a 0 to 10 point Likert-type scale with 0 being none, 1 being the minimal level, 5 being the average, and 10 being the highest level: ● Component complexity: it refers to the number of elements and requirements of the project. ● Coordinative complexity: it involves the indication on the level of interface with other projects (system

	interoperability). ● Dynamic complexity: It relates to the volatility of the project, i.e., the frequency of requirements changes, usually requested by the customers. Therefore, four items can capture the software dynamic complexity in terms of changes in coding and testing requirements, systems interfaces, and customer interfaces.
Task Complexity (McCabe, 1976) (Colazo, 2008)	Ratio scale, it captures the idea of the number of decision points in the software functions and it is measured by using the project's average McCabe's cyclomatic complexity factor.
Product Complexity (ROTHENBERGER et al., 2010)	Five point Likert-based scale, ranging from Very Low (e.g., simple input, output operations) to Extra High (e.g., natural language interface, distributed hard real-time control). It measures the complexity of product using five areas including control operations, computational operations, device-dependent operations, data management operations, and the user interface management operations.

Articles:

References
COLAZO, Jorge A. Following the sun: Exploring productivity in temporally dispersed teams. **AMCIS 2008 Proceedings**, p. 240, 2008.
MUNCH, Jurgen; ARMBRUST, Ove. Using empirical knowledge from replicated experiments for software process simulation: A practical example. In: **Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on**. IEEE, 2003. p. 18-27.
MOHAPATRA, Sanjay. Maximising productivity by controlling influencing factors in commercial software development. **International Journal of Information and Communication Technology**, v. 3, n. 2, p. 160-179, 2011.
NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.
ROTHENBERGER, Marcus A.; KAO, Yi-Ching; VAN WASSENHOVE, Luk N. Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects. **Information & Management**, v. 47, n. 7, p. 372-379, 2010.

Code/Factor: Business Area

Definition:

It indicates the application domain within the organization/industry that the project/system/software will be supporting.

Theoretical Basis:

There are empirical studies investigating the hypothesis that developing applications to some business areas are more complex than to other ones. Although there is no clear theoretical statement about how the business area can affect the software productivity, it seems that some types of business area may be very difficult to understand and, therefore, developing software applications for them may be hard when compared to others.

Viewpoints:

To develop software for some types of business area may apparently be more productive than other ones.

Measures found in studies:

Name	Specification
Business Area	Nominal Scale. It assumes values as “Telecom”, “Banking”, “Health”, “Aerospace”,etc.

Articles:

References
HE, Mei et al. An Investigation on Performance of Software Enhancement Projects in China. In: **Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific**. IEEE, 2008. p. 67-74.
MYRTVEIT, Ingunn; STENSRUD, Erik. Benchmarking COTS projects using data envelopment analysis. In: **Software Metrics Symposium, 1999. Proceedings. Sixth International**. IEEE,

1999. p. 269-278.

TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.

TSUNODA, Masateru; ONO, Kenichi. Pitfalls of analyzing a cross-company dataset of software maintenance and support. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.

WANG, Hao; WANG, Haiqing; ZHANG, Hefei. Software productivity analysis with CSBSG data set. In: **Computer Science and Software Engineering, 2008 International Conference on**. IEEE, 2008. p. 587-593.

Code/Factor: Communication among Developers

Definition:

It intends to support the observation of whether and how the communication channels, delays, and breakdowns among developers can affect software productivity.

Theoretical Basis:

Most of the developers' time is spent in design meetings, resolving problems within the team, addressing specification misunderstanding with the customers, doing other communications with the customer and product testing, and so on. Thus, communication channels, delays, and breakdowns among the development team members often cause schedule delays.

Viewpoints:

Communication mechanisms among team members are essential to the successful design and implementation of innovative projects (LIANG et al., 2012). Communication delays and breakdowns have been identified as one reason behind such schedule and cost overruns in software projects, affecting the software development productivity. Most of the developer’s time is spent in communicating with or looking for information from other team members and it can influence the software productivity.

Measures found in studies:

Name	Specification
Communication (Liang et al., 2012)	Likert scale (5-points) ranging from 1 (strongly disagree) to 5 (strongly agree), regarding the following perspectives: ● There is frequent communication within the team. ● The team members often communicate in spontaneous meetings, phone conversations, etc. ● The team members are happy with the timeliness in which they receive information from other team members. ● The team members are happy with the precision of the information they receive from other team members. ● The team members are happy with the usefulness of the information they receive from other team members.

Articles:

References
AVRITZER, Alberto & LIMA, Adailton. An Empirical Approach for the Assessment of Scheduling Risk in a Large Globally Distributed Industrial Software Project. In: **Fourth IEEE International Conference on Global Software Engineering, Limerick, Ireland**. 2009.
LIANG, Ting-Peng et al. The impact of value diversity on information system development projects. **International Journal of Project Management**, v. 30, n. 6, p. 731-739, 2012.
TEASLEY, Stephanie D. et al. Rapid software development through team collocation. **IEEE Transactions on Software Engineering**, v. 28, n. 7, p. 671-683, 2002.

Code/Factor:Developer's Domain Knowledge

Definition:

It intends to represent the level of shared representation, interpretation, and systems of developers' knowledge about the problem, including organizational operations, the functioning of user departments, administrative experience and skill, and expertise in the application area of the software under development.

Theoretical Basis:

Taking into account the domain knowledge of developers may become necessary when building core system functions. When there are only novices or not experts in the application domain, critical aspects may go unnoticed until late in the development process and affect software productivity in different developers' tasks.

Viewpoints:

When developers put themselves in the place of their users they can contribute more efficiently to the software development, being more precise, avoiding rework and therefore increase the software productivity.

Measures found in studies:

Name	Specification
Cognitive social capital (HSU & HUNG, 2013)	a five-point Likert scale ranging from ''strongly disagree'' to ''strongly agree'' for the following items: ● Team's level of expertise in the overall knowledge of the organizational operations ● Team's level of expertise in in-depth knowledge of the functioning of user departments. ● Team's level of expertise in overall administrative experience and skill. ● Team's level of expertise in the specific application area of the software system.

Articles:

References
AVRITZER, Alberto; LIMA, Adailton. An empirical approach for the assessment of scheduling risk in a large globally distributed industrial software project. In: **Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on**. IEEE, 2009. p. 341-346.
HSU, Jack Shih-Chieh; HUNG, Yu Wen. Exploring the interaction effects of social capital. **Information & Management**, v. 50, n. 7, p. 415-430, 2013.
MUNCH, Jurgen; ARMBRUST, Ove. Using empirical knowledge from replicated experiments for software process simulation: A practical example. In: **Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on**. IEEE, 2003. p. 18-27.

Code/Factor: Developers' Availability and Allocation

Definition:

It intends to represent the perception of absence/presence of human resources at the moment of performing tasks and activities throughout software development.

Theoretical Basis:

The allocation of resources in software projects is crucial for successful software development. Among various types of resources, human resources are the most important as software development is a human-intensive activity. Human resource allocation is very complex owing to the human characteristics of developers (KANG et al., 2011). The human characteristics affecting the allocation of resources can be grouped into individual-level characteristics and team-level characteristics. At the individual level, familiarity with the tasks needs to be taken into account as it affects the performance of developers. In addition, developers have different levels of productivity, depending on their capability and experience; the productivity of developers also varies according to their allocated tasks. Before defining resources allocation strategies, it is important to understand the availability of resources in the software project. Usually, it is expected that one developer cannot perform two activities at the same time and when he/she is executing some activity in the process he/she is unavailable to other concurrent activities. However, in particular situations, managers and team leaders can opt to concentrate more developers in problematic or collaborative-nature activities for the belief that will be more productive when cooperating.

Viewpoints:

A lack of developers' availability or inadequate allocation strategies may negatively impact the software productivity.

Measures found in studies:

Name	Specification
Staff Availability (MAXWELL & FORSELIUS,2000)	Five-point Likert-based scale ranging from 1 (very low) to 5 (very high), reported by the managers and team leaders through questionnaires.

Articles:

References
KANG, Dongwon; JUNG, Jinhwan; BAE, Doo- Hwan. Constraint- based human resource allocation in software projects. **Software: Practice and Experience**, v. 41, n. 5, p. 551-577, 2011.
Kuchar, t. & Vondrak, I. (2013), Simulating human resource capability and productivity in software process simulations, In: **25th European Modeling and Simulation Symposium, EMSS 2013**, pp. 255-263.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.
SCHLÜTER, Thea; BIRKHÖLZER, Thomas. Modeling and analysis of software development management as closed loop control. In: **Software and System Process (ICSSP), 2012 International Conference on**. IEEE, 2012. p. 210-214.

Code/Factor: Developers' Experience in Software Development

Definition:

It represents the subjective perception of the developer's experience on developing software. It can be observed through the acquisition of knowledge from training, coaching, doing differents tasks or using methods and techniques performed in different software projects over time.

Theoretical Basis:

When a developer has experience of different technologies, techniques, methods and activities in the software development, he/she can produce results faster than an inexperienced one. Thus, a project team having more experienced developers in the technologies applied in the current software development process can have higher productivity than other team having no such experience. This expectation is in conformity with the findings of Boehm (1981 apud MOHAPATRA, 2011), Chrysler (1978 apud MOHAPATRA, 2011) and Vosburg (1984 apud MOHAPATRA, 2011).

Following Boehm (1981)'s perspective, developers' experience and capability must be independently treated as distinct personal factors.

Under Boehm (1981)'s perspective, the level of experience shall be seen as result of learning obtained from an immersive environment related to teaching, coaching, performed tasks and laboral activities, using or not tools, methods and techniques.

Some researchers have considered various measures, such as foreign language experience, requirements, design or programming experience as proxies for the level of experience of a team member.

Viewpoints:

If developers have been previously exposed to necessary knowledge about how to use the necessary resources to produce the expected results, then they can access this knowledge to develop necessary competences to be more productive affecting the software productivity. For example, if a team has experience in some pertinent technology, it can develop one specific application using their technology faster than an inexperienced one.

Measures found in studies:

Name	Specification
Avareage Developer Experience (NGUYEN et al., 2011)	Usually, Likert-based scale based on COCOMO II (Boehm, 1981) developers' experience time (month or years) in software development, ranging according the following values: ● Very Low → No new features; standard components; conversions only; ● Low → at least 6 months; ● Nominal → at least 1 year; ● High → at least 3 years; ● Very High → at least 6 years.
Average project team experience (MAXWELL & FORSELIUS, 2000)	Likert-based scale, based on developers' experience time (month or years) in software development, ranging according the following values: ● Very Low → up to 6 months; ● Low → at least 1 months; ● Nominal → at least 3 year; ● High → at least 6 years; ● Very High → up to 6 years.
Personnel Experience (Otero et al., 2012) (LOPEZ-MARTIN et al., 2014)	Ratio scale, defined as the average number of months or years of experience.
Experience in technology (MOHAPATRA, 2011)	Ratio scale, measured in person-months of experience.

Articles:

References
CHANG, Carl K.; JIANG, Hsin-yi. Modeling Software Project Scheduling Based on Team Productivity And Its Simulations. In: **Proceedings of 2006 International Symposium on Distributed Computing and Applications to Business**, Engineering and Science. 2006.
HANAKAWA, Noriko; MORISAKI, Syuji; MATSUMOTO, Ken-ichi. A learning curve based simulation model for software development. In: **Software Engineering, 1998. Proceedings of the 1998 International Conference on**. IEEE, 1998. p. 350-359.
LOPEZ-MARTIN, Cuauhtemoc; CHAVOYA, Arturo; MEDA-CAMPANA, Maria Elena. A machine learning technique for predicting the productivity of practitioners from individually developed software projects. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.
MOHAPATRA, Sanjay. Maximising productivity by controlling influencing factors in commercial software development. **International Journal of Information and Communication Technology**, v. 3, n. 2, p. 160-179, 2011.
MUNCH, Jurgen; ARMBRUST, Ove. Using empirical knowledge from replicated experiments for software process simulation: A practical example. In: **Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on**. IEEE, 2003. p. 18-27.
NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.
OTERO, Luis Daniel et al. A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers. **IEEE Transactions on Engineering Management**, v. 59, n. 3, p. 391-400, 2012.

Code/Factor: Functional (FP-based) Size

Definition:

It represents the idea of observing the software functional size measured through counting function-points (FP).

Theoretical Basis:

Some researchers investigate whether there is some relationship between software functional size and software productivity. They have tried to observe this relationship by using FP-based measures, assuming that this kind of measure may theoretically represent the functional size of software.

Viewpoints:

Increasing the software functional size may influence software productivity over time.

Measures found in studies:

Name	Specification
FP-based size	Ratio scale, measured as software functional size measured in function-points.

Articles:

References
FOULDS, Les R.; QUADDUS, Mohammed; WEST, Martin. Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. In: **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. IEEE, 2007. p. 724-731.
MORASCA, Sandro; RUSSO, Giuliano. An empirical study of software productivity. In: **Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International**. IEEE, 2001. p. 317-322
TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.

TSUNODA, Masateru; ONO, Kenichi. Pitfalls of analyzing a cross-company dataset of software maintenance and support. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.

WANG, Hao; WANG, Haiqing; ZHANG, Hefei. Software productivity analysis with CSBSG data set. In: **Computer Science and Software Engineering, 2008 International Conference on**. IEEE, 2008. p. 587-593.

Code/Factor: LOC-based Software Size

Definition:

It represents the observation of software size measured through counting lines of source code.

Theoretical Basis:

Some researchers investigate whether there is some relationship between software size and software productivity. They have tried to find out this relationship by using LOC-based measures, assuming that this kind of measure may theoretically represent the idea of software size.

Viewpoints:

Increasing the software size may impact software productivity over time.

Measures found in studies:

Name	Specification
LOC-based size	Software size measured in lines of code.

Articles:

References
BANKER, Rajiv D.; SLAUGHTER, Sandra A. A field study of scale economies in software maintenance. **Management science**, v. 43, n. 12, p. 1709-1725, 1997.
FOULDS, Les R.; QUADDUS, Mohammed; WEST, Martin. Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. In: **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. IEEE, 2007. p. 724-731.
JEFFERY, D. Ross. A software development productivity model for MIS environments. **Journal of Systems and Software**, v. 7, n. 2, p. 115-125, 1987.

LOPEZ-MARTIN, Cuauhtemoc; CHAVOYA, Arturo; MEDA-CAMPANA, Maria Elena. A machine learning technique for predicting the productivity of practitioners from individually developed software projects. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.

MACCORMACK, Alan et al. Trade-offs between productivity and quality in selecting software development practices. **Ieee Software**, v. 20, n. 5, p. 78-85, 2003.

MAXWELL, Katrina D.; VAN WASSENHOVE, Luk; DUTTA, Soumitra. Software development productivity of European space, military, and industrial applications. **IEEE Transactions on Software Engineering**, v. 22, n. 10, p. 706-718, 1996.

MOAZENI, Ramin; LINK, Daniel; BOEHM, Barry. COCOMO II parameters and IDPD: bilateral relevances. In: **Proceedings of the 2014 International Conference on Software and System Process**. ACM, 2014. p. 20-24.

OTERO, Luis Daniel et al. A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers. **IEEE Transactions on Engineering Management**, v. 59, n. 3, p. 391-400, 2012.

ROTHENBERGER, Marcus A.; KAO, Yi-Ching; VAN WASSENHOVE, Luk N. Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects. **Information & Management**, v. 47, n. 7, p. 372-379, 2010.

Code/Factor: Maturity (Capability) Level

Definition:

It represents an indication on the capability of the organization regarding the software development process.

Theoretical Basis:

In recent decades new generations of standards, methods and models have emerged to support the software engineers achieve higher standards of quality. Essentially, they allow replacing improvisation with common or best practice and standards, which can assure acceptable levels of quality, covering factors such as feasibility, adaptability, reusability, and maintainability (De Oliveira et al., 2010).

Software process improvement (SPI) is largely concerned with improving a software project's capability to develop high-quality software based on the requirements of customers or end-users. The underlying principles of SPI are key elements of an effective software process and the description of an evolutionary, stepwise improvement path for software organizations from ad hoc, immature processes, to mature, disciplined ones (DI TULLIO & BAHLI, 2006). Thus, some models have been proposed to organize these improvement paths through the definition of maturity models.

One such model is the Capability Maturity Model Integration (CMMI), which is gaining ground internationally as a management tool for software development and maintenance processes and as a benchmark for contracting customized software.

Another one is the MPS model, in which, just like CMMI, their maturity levels are defined in two dimensions (based on the requirements of ISO/IEC 15504): process dimension and process capability dimension (process attributes) (Santos et al., 2010).

These maturity models are used on the one hand to strive at a standardization of processes and on the other hand to realize a decrease of product failures by eliminating their causes. However, it is questionable whether these improvement models can be applied in the same way for every organization. They all adopt maturity models in order to increase some parameters that can not measured and tracked at corporate level, such as productivity, lead times, and product quality.

The adoption of accepted processes best practices can help organizations achieve success in the software development market and is one-way software developers and their subcontractors have found to deal with the problems of quality and productivity resulting from the growing complexity and size of applications. Therefore, as higher is the level of maturity as higher should be the productivity. However, ‘‘software groups cannot be compared on these parameters, as the productivity level is highly dependent

on the type of software being developed, lead times cannot be compared without taking the business and development context into account, and a certain post-release defect density can be acceptable in one business while being totally unacceptable in another business''(De Oliveira et al., 2010).

Viewpoints:

As higher is the level of maturity as higher should be the productivity (and quality).

Measures found in studies:

Name	Specification
Organizational Maturity Level (DI TULLIO & BAHLI, 2006) (MOHAPATRA, 2011) (NGUYEN et al., 2011) (SANTOS et al,2010)	Ordinal scale based on the levels of CMMI (from 1 to 5) or MPS (from G to A).
Control Variable	
Company Size (De Oliveira et al., 2010)	small: with up to 99 employees; medium-sized: 100-499 employees; large: more than 500 employees (according to the classification made by SEBRAE (Brazilian Small Business Service).
Country (De Oliveira et al., 2010)	Nominal scale describing countries by name.

Articles:

References
DE OLIVEIRA, Saulo Barbará; VALLE, Rogério; MAHLER, Cláudio Fernando. A comparative analysis of CMMI software project management by Brazilian, Indian and Chinese companies. **Software Quality Journal**, v. 18, n. 2, p. 177-194, 2010.
DI TULLIO, Dany; BAHLI, Bouchaib. The impact of software process maturity and software development risk on the performance of software development projects. In: **ICIS 2006 Proceedings**, p. 90, 2006.
MOHAPATRA, Sanjay. Maximising productivity by controlling influencing factors in commercial software development. **International Journal of Information and Communication Technology**, v. 3, n. 2, p. 160-179, 2011.
NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.
PREMRAJ, Rahul et al. An empirical analysis of software productivity over time. In: **Software Metrics, 2005. 11th IEEE International Symposium**. IEEE, 2005. p. 10 pp.-37
RAMASUBBU, Narayan; BHARADWAJ, Anandhi; TAYI, Giri Kumar. Software process diversity: conceptualization, measurement, and analysis of impact on project performance. **MIS Quarterly**. 2015. v. 39 n. 4, pp. 787-807.
SANTOS, Gleison et al. MPS. BR: A tale of software process improvement and performance results in the Brazilian software industry. In: **Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the**. IEEE, 2010. p. 412-417.

Code/Factor: Level of Personnel Capability

Definition:

It represents the level of ability on which the team members efficiently or accurately perform their tasks in specific roles throughout the software development.

Theoretical Basis:

Software processes are also a special type of business processes that are highly dependent on human creativity, competencies and interaction (Dutoit et al. 2006). Such processes tend to be more uncertain than automated processes performed by machines, because human behaviour is very complex and depends on a lot of factors, including, emotions, social status, health, etc. Following Boehm (1981)'s perspective, developers' experience and capability must be independently treated as distinct personal factors. Under Boehm (1981)'s perspective, level of capability shall be seen as the developer's personal ability to use the knowledge acquired over time to cooperate, inform and produce the expected results efficiently and accurately.

Viewpoints:

If developers have the necessary capabilities (competencies, abilities, skills, behaviours, etc.) on how to use the necessary resources to produce the expected results, then they can be more productive and affect the software productivity.

Measures found in studies:

Name	Specification
Personnel Capability (KRISHNAN et al. 2000) (FARSHCHI et al., 2012) (TAN et al., 2009)	A five-point Likert scale available in COCOMO II, ranging from 1("Very low") to 5 ("Very high"), when the product manager and at least two randomly selected software engineers from the team rate the capability of the team on this scale. The final score for each product was obtained by averaging these responses.

Personnel Capability (Otero et al., 2012)	Ordinal scale, it is a subjective measure of the ability of resources based on their perceived potential, including motivation and communication skills. This parameter is subjectively determined by the managers and team leaders as a categorical variable with two levels: High = 1, Low = 0 .

Articles:

References
CARD, David N. A software technology evaluation program. **Information and Software Technology**, v. 29, n. 6, p. 291-300, 1987.
CHANG, Carl K.; JIANG, Hsin-yi. Modeling Software Project Scheduling Based on Team Productivity And Its Simulations. In: **Proceedings of 2006 International Symposium on Distributed Computing and Applications to Business, Engineering and Science**. 2006.
FARSHCHI, Mostafa; JUSOH, Yah Yusmadi; MURAD, Azmi Azrifah Masrah. Impact of personnel factors on the recovery of delayed software projects: A system dynamics approach. **Computer Science and Information Systems**, v. 9, n. 2, p. 627-652, 2012.
KRISHNAN, Mayuram S. et al. An empirical analysis of productivity and quality in software products. **Management Science**, v. 46, n. 6, p. 745-759, 2000.
Kuchar, t. & Vondrak, I. (2013), Simulating human resource capability and productivity in software process simulations, In: **Proceddings of 25th European Modeling and Simulation Symposium, EMSS 2013**, pp. 255-263.
MUNCH, Jurgen; ARMBRUST, Ove. Using empirical knowledge from replicated experiments for software process simulation: A practical example. In: **Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on**. IEEE, 2003. p. 18-27.
OTERO, Luis Daniel et al. A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers. **IEEE Transactions on Engineering Management**, v. 59, n. 3, p. 391-400, 2012.
TAN, Thomas et al. Productivity trends in incremental and iterative software development. In: Proceedings of the 2009 3rd **International Symposium on Empirical Software Engineering and Measurement**. IEEE Computer Society, 2009. p. 1-10.

Code/Factor: Product Quality

Definition:

It represents an indication on whether and how the absence/presence of defects may impact the software productivity.

Theoretical Basis:

There are empirical studies arguing that there are relationships between the product quality and software productivity and they may differently affect each other (KRISHNAN et al., 2000, RAMASUBBU & BALAN, 2007; RAMASUBBU et al., 2011).

Viewpoints:

The idea behind this variable is to observe whether high levels of productivity may be largely achieved through the high levels of product quality. Software Products having high level of quality are simpler to develop/maintain than any others.

Measures found in studies:

Name	Specification
Conformance Quality (KRISHNAN et al., 2000)	Ratio scale, measured as the number of unique problems reported by customers normalized by the product size mea- sured in thousand lines of code.
Quality (RAMASUBBU et al., 2011)	Ratio Scale, measured as the delivered code defect density as following, where Defects Delivered were the number of defects reported by the client after the project had been delivered
Conformance Quality (RAMASUBBU & BALAN, 2007)	Ratio scale, it captures the number of unique problems reported by customers during the acceptance tests and production. It is calculated as follows: Quality Conformance = 1/(#reported defects +1)

Articles:

References
KRISHNAN, Mayuram S. et al. An empirical analysis of productivity and quality in software products. **Management science**, v. 46, n. 6, p. 745-759, 2000.
RAMASUBBU, Narayan; BALAN, Rajesh Krishna. Globally distributed software development project performance: an empirical analysis. In: **Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. ACM, 2007. p. 125-134.
RAMASUBBU, Narayan et al. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: **Proceedings of the 33rd international conference on Software engineering**. ACM, 2011. p. 261-270.

Code/Factor: Programming Language

Definition:

It indicates the development platform (in terms of coding) used to produce the source code.

Theoretical Basis:

Researchers have tried to establish whether there is some relationship between some characteristics of software products and and software productivity. Among these, some empirical studies investigate whether the programming languages' characteristics may affect the software productivity.

Viewpoints:

The choice of a programming language to develop completely or parts of software product can affect software productivity.

Measures found in studies:

Name	Specification
Language Generation (HUANG et al., 2015) (RODRIGUEZ et al., 2012)	Nominal Scale, assuming values such as 3GL, 4GL, etc.
Language Name (HE et al., 2008) (RODRIGUEZ et al., 2012) (TSUNODA et al., 2009) (TSUNODA & ONO, 2014)	Nominal Scale assuming values such as C,C++,Java, etc.

Articles:

References
HE, Mei et al. An Investigation on Performance of Software Enhancement Projects in China. In: **Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific**. IEEE, 2008. p. 67-74.
HUANG, Jianglin; SUN, Hongyi; LI, Yan-Fu. An empirical study of the impact of project factors on software economics. In: **Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on**. IEEE, 2015. p. 43-47.
RODRIGUEZ, D. et al. Empirical findings on team size and productivity in software development. **Journal of Systems and Software**, v. 85, n. 3, p. 562-570, 2012.
TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.
TSUNODA, Masateru; ONO, Kenichi. Pitfalls of analyzing a cross-company dataset of software maintenance and support. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on**. IEEE, 2014. p. 1-6.

Code/Factor: Project Duration

Definition:

It represents the time spent to build the software project.

Theoretical Basis:

There are empirical studies investigating the possible relationship between the project duration (in months or years) and software productivity. According to these studies, to verify the existence and magnitude of this relationship may point out to some superior limit of duration at certain level of software productivity.

Viewpoints:

The project's elapsed time can have an observable effect on software productivity over time.

Measures found in studies:

Name	Specification
Duration	Ratio scale, measured as the time lapse between start and release dates of project measured in months or years.

Articles:

References
LIU, Li; KONG, Xiaoying; CHEN, Jing. How project duration, upfront costs and uncertainty interact and impact on software development productivity? A simulation approach. **International Journal of Agile Systems and Management**, v. 8, n. 1, p. 39-52, 2015.
MAXWELL, Katrina D.; VAN WASSENHOVE, Luk; DUTTA, Soumitra. Software development productivity of European space, military, and industrial applications. **IEEE Transactions on Software Engineering**, v. 22, n. 10, p. 706-718, 1996.
TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.

Code/Factor: Requirements Volatility

Definition:

It represents the perception on how environmental changes resulting in unstable requirements may affect the software productivity.

Theoretical Basis:

One component of uncertainty is the nature of the environment changes, which results in unstable requirements. This requirements instability in turn leads to interpersonal conflict among stakeholders. This is because the users desire changes to reflect environmental changes, while developers would prefer to lock-in requirements so that the project can be delivered within the budget and on time. Meanwhile, interpersonal conflicts could lead to a lack of collaborative decision-making, causing disagreement on the system requirements among stakeholders with diverse views on requirements. Unfortunately, given the importance of conflict and requirements uncertainty in the software project success, the relationship between user–developer, interpersonal conflicts, and requirements uncertainty are overlooked in the literature (Liu et al. 2011).

Viewpoints:

An increasing in level of requirements volatility may negatively impact software productivity.

Measures found in studies:

Name	Specification
Requirements Instability (Liu et al. 2011)	the following items were scored using a five-point Likert scale ranging from “total disagreement” (1) to “total agreement” (5) : ● Requirements fluctuated quite a bit in later phases. ● Requirements identified at the beginning were quite different from those at the end. ● Requirements will fluctuate quite a bit in the future.
(Otero et al., 2012)	Ordinal scale, A dichotomous categorical variable with high and low levels, where a high level indicates the presence of a major requirement change.
(RAMASUBBU et al.,	Percentage of effort spent on rework due to change in customer requirements, holding the planned development effort before the

2015)	change occurred as the baseline.
(MAXWELL & FORSELIUS,2000)	Likert-scaled variable, based on COCOMO II (Boehm, 1981) requirement volatility driver, ranging according the following values: ● Very Low → up to 2 months; ● Low → Some changes to specifications; some new or adapted functions; some minor changes in data contents; ● Nominal → More changes to specifications, but project members can handle them, and their impact is minor (< 15% new or modified functions); ● High → Some major changes, impacting total architecture and requiring rework; 15–30% of functions new or modified; ● Very High → Continuously new requirements; lots of rework; > 30% of the functions new or modified.

Articles:

References
LIU, Julie Yu-Chih et al. Relationships among interpersonal conflict, requirements uncertainty, and software project performance. **International Journal of Project Management**, v. 29, n. 5, p. 547-556, 2011.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.
OTERO, Luis Daniel et al. A DEA–Tobit Analysis to Understand the Role of Experience and Task Factors in the Efficiency of Software Engineers. **IEEE Transactions on Engineering Management**, v. 59, n. 3, p. 391-400, 2012.
RAMASUBBU, Narayan; BHARADWAJ, Anandhi; TAYI, Giri Kumar. Software process diversity: conceptualization, measurement, and analysis of impact on project performance. **MIS Quarterly**, v. 39, n. 4, pp. 787-807, 2015.

Code/Factor: Reuse Flexibility

Definition:

It aims to represent subjectively the level of flexibility on reuse of requirements; analysis and design can benefit software projects and increase their productivity.

Theoretical Basis:

Flexible design and Reuse can be considered as a base to efficient and effective delivery customer requirements and changes. Thus, the emphasis on flexible high level analysis and design aims at the definition of structures (such as components, architectures, catalogue of requirements, standards and patterns) that can permit to achieve desirable levels of modularity, cohesion, etc and facilitates to reuse such structures in different software solutions. Thus, some product-related measures (such as complexity, modularity, etc.) may be seen as proxies or indirect variables for this construct.

Viewpoints:

According RAMASUBBU & BALAN (2007), Flexible design and Reuse in software enables developers to use standardized routines that have been stored in the organization-wide repositories and libraries to accomplish certain functionality. Usage of such standardized and pre-tested functions helps the developers to avoid “reinventing the wheel”, and to focus on the customer specific needs. Thus, the general belief is that reuse plays an important role in impacting software productivity. Reuse has other indirect influences such as its potential effect on software quality as well. However, similar to other software variables studies, we use it primarily to observe productivity.

Measures found in studies:

Name	Specification
Standards use (MAXWELL & FORSELIUS, 2000) (NGUYEN et al., 2011) (TAN et al., 2009)	Likert-based scale, based on COCOMO II, ranging from Very Low (1) to High (5), whereas each item means: ● Very low → Project managers must develop standards during project; ● Low → Some standards are available, but not familiar; project managers must develop more; ● Nominal → Project members use generally known

	standards in known environments; ● High → Project members use new standards, which other customers or software houses have applied in the same environment; ● Very High → Project members use new standards, which are not familiar in the industry, but must be followed; use is controlled.
Reuse Level (Morisio et al. 2002.)	Reuse level, ratio between reused size and total size delivered. Applies only to applications developed using the framework. Each application reuses a certain part (utilization factor U_{FWK}) of the framework (S_{FWK}) and develops new parts (S_i) $$RL_i = \frac{U_{FWK} S_{FWK}}{(U_{FWK} S_{FWK} + S_i)}$$ Ratio scale.
Code Reuse (RAMASUBBU & BALAN, 2007)	Ratio scale, number of lines of code reused / total of product's line of code.

Articles:

References
AHMED, A. et al. Agile software development: Impact on productivity and quality. In: **Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on**. IEEE, 2010. p. 287-291.
DE GEA, Carrillo et al. Co- located and distributed natural- language requirements specification: traditional versus reuse- based techniques. **Journal of Software: Evolution and Process**, 2016.
FOULDS, Les R.; QUADDUS, Mohammed; WEST, Martin. Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. In: **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. IEEE, 2007. p. 724-731.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.

MORISIO, Maurizio; ROMANO, Daniele; STAMELOS, Ioannis. Quality, productivity, and learning in framework-based development: an exploratory case study. **IEEE Transactions on Software Engineering**, v. 28, n. 9, p. 876-888, 2002.

NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.

RAMASUBBU, Narayan; BALAN, Rajesh Krishna. Globally distributed software development project performance: an empirical analysis. In: **Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. ACM, 2007. p. 125-134.

TAN, Thomas et al. Productivity trends in incremental and iterative software development. In: **Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement**. IEEE Computer Society, 2009. p. 1-10.

Code/Factor: Software Development Methodology

Definition:

It indicates the different way of developing software solutions adopted in the projects, for instance, agile, waterfall, component-based development and so on.

Theoretical Basis:

Since the software development environment and culture are different in each organization, we might observe different software development approaches applied in the industry. However, it is not clear how the distinctive properties of software development adopted in many software projects can affect the software development productivity.

It allows observing whether and how a project is performed under several development methodologies, considering different strategies for software project management, using standards for paradigms, frameworks and controls.

Software development methodologies define development processes, roles played by different stakeholders, techniques used to design systems, and artifacts that document development processes.

Development methodologies found in the technical literature can be characterized by projects following highly plan-based waterfall or V-model approach for development, using outsourcing, and, even, projects that followed an agile, cyclical or spiral approach to development.

All the development methodologies have been adopted under the expectation of achieving high quality of final products and high productivity of development teams. Thus, empirical studies exploring the comparison among some of these methodologies will use this variable.

Viewpoints:

The adoption of different development methodologies may differently affect the achievement of certain level of software productivity.

Measures found in studies:

Name	Specification
Outsourcing ratio (Tsonuda et al., 2009)	Ratio scale, measured as percentage of software outsourcing regarding total project size.
Plan-oriented management (Mizuno et al., 2000)	Ratio scale, it represents the deviation from software management plan.
Development Methodology (RAMASUBBU et al.,2011)	Nominal scale, assuming values such as waterfall, agile, RUP,etc.
Use of Software-Development Method	We planned an appropriate software process for the project. We used appropriate software methods for the project. We and our client were clear about what methods should be used during the projects. We and our client had the common understanding about software methods which were used during the project.
Five point Likert-based scale (mostly disagree (1) to mostly agree (5) (Xu & YAO, 2013) for above items.	

Articles:

References
DE SOUZA CARVALHO, William Chaves et al. A comparative analysis of the agile and traditional software development processes productivity. In: **Computer Science Society (SCCC), 2011 30th International Conference of the Chilean**. IEEE, 2011. p. 74-82.
KAMMA, Damodaram et al. Effect of Model Based Software Development on Productivity of Enhancement Tasks--An Industrial Study. In: **Software Engineering Conference (APSEC), 2014 21st Asia-Pacific**. IEEE, 2014. p. 71-77.
MIZUNO, Osamu et al. Statistical analysis of deviation of actual cost from estimated cost using actual project data. **Information and Software Technology**, v. 42, n. 7, p. 465-473, 2000.
RAMASUBBU, Narayan et al. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: **Proceedings of the 33rd international conference on Software engineering**. ACM, 2011. p. 261-270.
RAMASUBBU, Narayan; BHARADWAJ, Anandhi; TAYI, Giri Kumar. Software process diversity: conceptualization, measurement, and analysis of impact on project performance. 2015.

TSUNODA, Masateru et al. Software development productivity of Japanese enterprise applications. **Information Technology and Management**, v. 10, n. 4, p. 193, 2009.

XU, Peng; YAO, Yurong. Knowledge sharing in offshore software development: A vendor perspective. **Journal of Global Information Technology Management**, v. 16, n. 1, p. 58-84, 2013.

Code/Factor: Software Quality Implementation

Definition:

It represents an indication on whether and how software quality practices, such as inspections, verification, validation, configuration management and so on, are applied or combined throughout the software development process.

Theoretical Basis:

There are empirical studies arguing that the use of methods and techniques regarding verification, validation, testing or any other related to quality assurance could have positive effects on software productivity. It is based on implementation of some practices in key process areas related to software quality in improvement programs, like CMM, MPS.Br, PSP, etc.

Viewpoints:

The idea behind this variable is to observe whether high levels of productivity may be largely achieved through the implementation of quality-related practices. High level of software quality assurance of an organizational unit will be positively affect the software productivity.

Measures found in studies:

Name	Specification
Quality Processes (implemented) (KRISHNAN et al., 2000)	Based on five-points Likert scale, ranging from 1 (“The practice is performed rarely if ever”) to 5 (“The practice is performed more than 90% of the time”): ● Software Configuration Management ● Defect Prevention ● Peer Review ● Software Quality Assurance
Development quality (ROTHENBERGER	Ratio scale, it is the inverse defect rate based on all the defects reported during all the inspection and testing processes before release. The number of defects discovered before release is an indicator of the quality of the code during development. Therefore, it has been defined as the project size divided by the

et al. 2010)	total number of defects reported before release.
(ANDERSON & GHAVAMI,1999)	Ratio-scaled output measures based on DEA analysis: ● Defects discovered, (DEF_DIS): indicates the number of defects discovered. ● Defects corrected, (DEF_COR): is the number of defects corrected.
Prevention QMA (RAMASUBBU & BALAN, 2007)	Ratio scale, % of total project hours in prevention-based approach (Prevention-based quality management practices in software development involve activities such as training that are primarily done to avoid the occurrence of errors. A score was computed a score for this approach based on the percentage of total development effort spent on training, project planning and configuration management activities).
Appraisal QMA (RAMASUBBU & BALAN, 2007)	% of total project hours in appraisal-based approach (Appraisal-based activities involve proactively assessing progress performance and quality of intermediate artifacts at various stages of development. A score was computed for this approach based on the percentage of total development effort spent on peer reviews of requirement, design, status reviews and code inspection).
Failure QMA (RAMASUBBU & BALAN, 2007)	% of total project hours failure-based approach (Failure-based quality approaches include testing the adherence of applications to customer specifications and subsequent defect correction activities. A score was computed for this approach based on the percentage of total development effort spent on unit tests, module integration tests and system tests).

Articles:

References

ANDERSON, Timothy R.; GHAVAMI, Peter K. Using Data Envelopment Analysis for evaluating alternative software development process configurations. In: **Management of Engineering and Technology, 1999. Technology and Innovation Management. PICMET'99. Portland International Conference on**. 1999.

KRISHNAN, Mayuram S. et al. An empirical analysis of productivity and quality in software products. **Management science**, v. 46, n. 6, p. 745-759, 2000.

RAMASUBBU, Narayan; BALAN, Rajesh Krishna. Globally distributed software development project performance: an empirical analysis. In: **Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. ACM, 2007. p. 125-134.

ROTHENBERGER, Marcus A.; KAO, Yi-Ching; VAN WASSENHOVE, Luk N. Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects. **Information & Management**, v. 47, n. 7, p. 372-379, 2010.

Code/Factor: Software Risk Exposure

Definition:

It represents the level of project uncertainty regarding organizational, environmental, users, requirements, planning and control factors having observable impact on the extent to which the software is able to support new or changes of functions in response to business needs (Nidumolu 1996 apud DI TULLIO & BAHLI, 2006).

Theoretical Basis:

Risk exposure has been studied as residual performance risk, which is defined as the difficulty in estimating performance-related outcomes during later stages of software development projects. Project uncertainty increases residual performance risk while both project uncertainty and residual performance risk have a direct negative effect on project performance (Nidumolu 1995; Nidumolu 1996 apud DI TULLIO & BAHLI, 2006).

Project management risks were also found to be significantly related to both the process performance and the product performance of software development projects. Indeed, such factors as organizational environment risk, user risk, requirements risk, project complexity risk, planning and control risk, as well as team risk, have a significant negative impact on both the process and product performance of software development projects.

Viewpoints:

In general, the level of risk exposure may indicate a negative impact on both the process and product performance dimensions and thus stress the need to address the software development risks when considering key organizational concerns such as the performance of software development projects (Wallace et al. 2004 apud DI TULLIO & BAHLI, 2006).

Measures found in studies:

Name	Specification
Risk exposure (ANDERSON & GHAVAMI,1999)	Ratio scale, measured as the relative measure of risk proportional to the project duration and defects remained in the software.

Software Development Risk (DI TULLIO & BAHLI, 2006)	It is comprised of of 46 items organized grouped along five dimensions: technological acquisition, project size, degree of expertise, organizational environment, and application complexity. All measured using a 7-point Likert-type scale with anchors ranging from “Strongly disagree” and “No expertise” to “Strongly Agree” and “Outstanding Expertise”.
Risk Resolution (NGUYEN et al., 2011)	Six point Likert-based scale, ranging from Very low (1) to Extra High(6) for items below: ● Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR. ● Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan ● Percent of development schedule devoted to establishing architecture, given general product objectives ● Percent of required top software architects available to project ● Tool support available for resolving risk items, developing and verifying architectural specs ● Level of uncertainty in Key architecture drivers: mission, user interface, COTS, hardware, technology, performance. ● Number and criticality of risk items

Articles:

References

ANDERSON, Timothy R.; GHAVAMI, Peter K. Using Data Envelopment Analysis for evaluating alternative software development process configurations. In: **Management of Engineering and Technology, 1999. Technology and Innovation Management. PICMET'99. Portland International Conference on**. 1999.

DI TULLIO, Dany; BAHLI, Bouchaib. The impact of software process maturity and software development risk on the performance of software development projects. **ICIS 2006 Proceedings**, p. 90, 2006.

JEET, Kawal; BHATIA, Nitin; MINHAS, Rajinder Singh. A model for estimating the impact of low productivity on the schedule of a software development project. **ACM SIGSOFT Software Engineering Notes**, v. 36, n. 4, p. 1-6, 2011.

NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.

Code/Factor: Software Team Diversity

Definition:

It subjectively represents the heterogeneity within the team in terms of individual team members' attributes, such as age, gender, ethnic background, education, functional background, tenure, and technical abilities (Williams and O'Reilly 1998 apud Lee & XIA, 2010).

Theoretical Basis:

Some empirical studies suggests that a software team's internal variety should match the variety and complexity of the environment and that the diversity of skills amplify the internal variety that enables the team to respond to the changing environment (Highsmith 2004 appud Lee & XIA, 2010; Nerur and Balijepally 2007 appud Lee & XIA, 2010).

Although the conflict is the inevitable companion of diversity, some empirical results suggests that software teams need to bring together a variety of skills and perspectives to see problems and pitfalls, to think of multiple ways to solve problems, and to implement the solutions (Coad et al. 1999 appud Lee & XIA, 2010).

Viewpoints:

Some level of heterogeneity within the team in terms of individual attributes can affect software productivity.

Measures found in studies:

Name	Specification
Software team diversity (LEE & XIA, 2010)	Likert-scale ranging from 1 (strongly disagree) to 7 (strongly agree), for following items: 1. The members of the project team are from different areas of expertise 2. The members of the project team have skills that complement each other 3. The members of the project team have a variety of different experiences 4. The members of the project team vary in functional backgrounds
Diversity (Chen & Chen, 2012)	Likert-scale ranging from 1 (strongly disagree) to 5 (strongly agree), for following items: 1. The members of the project team are widely vary in their areas of expertise. 2. The members of the project team vary in functional backgrounds and experiences. 3. The members of the project team have skills and abilities that complement each other.
Team Diversity (GÜNSEL, Ayşe & AÇIKGÖZ, 2013)	Likert-scale ranging from 1 (strongly disagree) to 5 (strongly agree), for following items: ● The members of the project team had skills that complemented each other ● The members of the project team had a diversity of different experiences ● The members of the project team varied in functional backgrounds

Articles:

References
CHEN, Pei-Chi; CHERN, Ching-Chin; CHEN, Chung-Yang. Software project team characteristics and team performance: Team motivation as a moderator. In: **Software Engineering Conference (APSEC), 2012 19th Asia-Pacific**. IEEE, 2012. p. 565-570.
GÜNSEL, Ayşe; AÇIKGÖZ, Atif. The effects of team flexibility and emotional intelligence on software development performance. **Group Decision and Negotiation**, p. 1-19, 2013.
LEE, Gwanhoo; XIA, Weidong. Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. **Mis Quarterly**, v. 34, n. 1, p. 87-114, 2010.

Code/Factor: Stakeholder Participation

Definition:

It represents the subjective perception on the participation of the stakeholder in the software development process.

Theoretical Basis:

Some studies emphasizes on how a strong the relationship between customers and developers could impact the software productivity. With the expanding internet technology and development of integrated solutions, the business demands faster delivering of projects to its clients and it can be best achieved with the continuous involvement of the stakeholders who have a stake in the project completion. This phenomenon is severe in offshoring, primarily because client teams and vendor teams generally have weaker ties than those in traditional environments (Gupta et al. 2009). Thus, the two teams may feel less motivated and obligated to share knowledge (Herbsleb et al. 2003). Prior research reveals that personal feelings, such as commitment and enjoyment, can change a participant's attitude and motivate individuals to engage in the knowledge sharing process (Malhotra et al. 2008). When two parties have established bonds, they tend to be more willing to engage in knowledge sharing (Bock et al. 2005). In OSD, if the vendor can maintain a close relationship with the client, it will greatly reduce the communication obstacles and motivate the contextual information sharing.

Viewpoints:

The active participation of customers in software development process may contribute to facilitate dynamic changes in requirements and, then, positively affect software productivity.

Measures found in studies:

Name	Specification
User participation in analysis and design (FOULDS & QUADDUS, 2007)	Five-point Likert scale, ranging from “not adopted” to “highly adopted”.

Customer Participation (MAXWELL & FORSELIUS, 2000)	Five-point Likert scale, from 1 (“Very low”) to to 5 (“Very high”).
Client Support (MOHAPATRA, 2011)	As perceived by project manager, seven-point Likert scale, ranging from 1 (not available) to 7 (highly available)**.**
Customer Involvement (RAMASUBBU et al. 2015)	It is calculated as the percentage of total project effort spent by a software team in engaging with end-users, parsing the daily time logs submitted by software project members that were utilized for billing and cost accounting purposes.
Relationship with Client (XU & YAO, 2013)	It employs a seven-point Likert scale, ranging from Strongly Disagree (1) to Strongly Agree (7) for the following items: ● During the development process, there was close interaction between the client and us. ● During the development process, the relationship between the client and us was characterized by mutual respect. ● During the development process, the relationship between the client and us was characterized by mutual trust. ● During the development process, close ties characterized the relationship between the client and us. ● During the development process, the relationship between the client and us was characterized by high reciprocity among team members.
Relational social capital (HSU & HUNG,2013)	Five point Likert-based scale, ranging from very low (1) to very high (5) for following items: ● There is mutual respect between developers and users ● There is mutual trust between developers and users ● There is personal friendship between developers and users ● There is high reciprocity among developers and users

Articles:

References
FOULDS, Les R.; QUADDUS, Mohammed; WEST, Martin. Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. In: **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. IEEE, 2007. p. 724-731.
HSU, Jack Shih-Chieh; HUNG, Yu Wen. Exploring the interaction effects of social capital. **Information & Management**, v. 50, n. 7, p. 415-430, 2013.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.
MOHAPATRA, Sanjay. Maximising productivity by controlling influencing factors in commercial software development. **International Journal of Information and Communication Technology**, v. 3, n. 2, p. 160-179, 2011.
RAMASUBBU, Narayan; BHARADWAJ, Anandhi; TAYI, Giri Kumar. Software process diversity: conceptualization, measurement, and analysis of impact on project performance. 2015.
XU, Peng; YAO, Yurong. Knowledge sharing in offshore software development: A vendor perspective. **Journal of Global Information Technology Management**, v. 16, n. 1, p. 58-84, 2013.

Code/Factor:Team Autonomy

Definition:

It subjectively represents the extent to which the software team is empowered with the authority and control in making decisions to carry out the project.

Theoretical Basis:

Team autonomy refers to the degree of discretion and independence granted to the team in scheduling the work, determining the procedures and methods to be used, selecting and deploying resources, hiring and firing team members, assigning tasks to team members, and carrying out assigned tasks (Breaugh 1985 apud LEE and XIA, 2010). It decentralizes decision-making power to those who will actually carry out the work (Tatikonda and Rosenthal 2000 apud LEE and XIA, 2010).

Viewpoints:

It is believed that team autonomy can positively contribute to software productivity.

Measures found in studies:

Name	Specification
Software team autonomy (reflective) (CHEN et al., 2012) (GÜNSEL & AÇIKGÖZ, 2013)	Items measured by 5-point Likert scales ranging from "strongly disagree" (1) to "strongly agree" (5) : • The project team was allowed to freely choose tools and technologies. • The project team had control over what they were supposed to accomplish.
Software team autonomy (LIA & XIA, 2010)	Item rated using a five-point Likert scale format ranging from 1='Strongly Disagree' to 7='Strongly agree'. 1. The project team was allowed to freely choose tools and technologies (AUTO1) 2. The project team had control over what they were supposed to accomplish (AUTO2) 3. The project team was granted autonomy on how to handle user requirement changes (AUTO3) 4. The project team was free to assign personnel to

	the project (AUTO4)

Articles:

References
CHEN, Pei-Chi; CHERN, Ching-Chin; CHEN, Chung-Yang. Software project team characteristics and team performance: Team motivation as a moderator. In: **Software Engineering Conference (APSEC), 2012 19th Asia-Pacific**. IEEE, 2012. p. 565-570.
GÜNSEL, Ayşe; AÇIKGÖZ, Atif. The effects of team flexibility and emotional intelligence on software development performance. **Group Decision and Negotiation**, p. 1-19, 2013.
LEE, Gwanhoo; XIA, Weidong. Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. **Mis Quarterly**, v. 34, n. 1, p. 87-114, 2010.

Code/Factor: Team Cohesion

Definition:

It represents the level at which team members are willing to work together in a software project. It refers to the level of mutual trust, respect, reciprocity, and relationship closeness between users and developers when performing the software project and it is different from the concept of sharing the same level of experience, competence or skills.

Theoretical Basis:

The general belief is that a development team needs to act as one cohesive unit to increase the efficiency of development. Thus, team members should work closely together and support one another for effective development.

This variable emphasizes whether the team members recognizes the specific potentials (strengths and weaknesses) of their individuals, whether they are contributing to the achievement of the team's goals in accordance with their specific potential and whether the imbalance of their contributions caused conflicts within the team or project performances, independently from team location or differences among member's skills, experience, motivation, etc (adapted from Liang et al., 2012).

Indeed, this is not about location (distributed or local development).

Viewpoints:

Increasing in the level of mutual trust, respect, reciprocity, and relationship closeness between users and developers (team cohesion) may positively affect the software productivity.

Measures found in studies:

Name	Specification
Relational social capital (HSU et al., 2013)	Based on a Likert-based scale, ranging from 1 (strongly disagree) to 5 (strongly agree) in the items below: ● There is mutual respect between developers and users; ● There is mutual trust between developers and users; ● There is personal friendship between developers and users; ● There is high reciprocity among developers and users;
Balance member of contributions (LIANG et al., 2012)	Based on a Likert-based scale, ranging from 1 (strongly disagree) to 5 (strongly agree) in the items below: ● The team recognize the specific potentials (strengths and weaknesses) of individual team members. ● The team members is contributing to the achievement of the team's goals in accordance with their specific potential. ● Imbalance of member contributions causes conflicts in the team.
COCOMO II's Team Cohesion driver (NGUYEN et al., 2011) (TAN et al., 2009)	Based on COCOMO II, a six point Likert-based scale ranging from 1 (very low) to 6 (extra high) for following items: ● Consistency of stakeholder objectives and cultures; ● Ability, willingness of stakeholders to accommodate other stakeholders' objectives; ● Experience of stakeholders in operating as a team; ● Stakeholder team building to achieve shared vision and commitments.
Personnel Imbalance (RAMASUBBU et al., 2011)	Extent of unevenness in distribution of personnel across locations, measured as below: Personnel Imbalance = Standard Deviation (na, nj,...,nk)] / N; where N = $\sum_{i=0}^{k}$ n*i* an n*i* is the team size at location i.

Articles:

References
HSU, Jack Shih-Chieh; HUNG, Yu Wen. Exploring the interaction effects of social capital. **Information & Management**, v. 50, n. 7, p. 415-430, 2013.
LIANG, Ting-Peng et al. The impact of value diversity on information system development projects. **International Journal of Project Management**, v. 30, n. 6, p. 731-739, 2012.
NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.
RAMASUBBU, Narayan et al. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: **Proceedings of the 33rd international conference on Software engineering**. ACM, 2011. p. 261-270
TAN, Thomas et al. Productivity trends in incremental and iterative software development. In: **Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement**. IEEE Computer Society, 2009. p. 1-10.

Code/Factor: Team Dispersion

Definition:

It intends to represent the relationship between sites/teams location, capturing differences in time zone and geographical difference.

Theoretical Basis:

Team Dispersion (or Work Dispersion) describes how distributed a project's development process is. It refers to spatial and temporal separation among the team's members participating in a development project.

Temporal dispersion refers to the time zone differences among project members, and its key impact on distributed work is quite distinct from spatial dispersion. Temporal dispersion reduces the possibilities of synchronous interaction, which is a critical communicational attribute for real-time problem solving and design activities (RAMASUBBU et al. 2011). Managing the flow of information in asynchronous interactions can be quite complex. For this reason, temporal dispersion makes misunderstandings and errors significantly more likely to happen. On the other hand, temporal dispersion may allow distributed groups to accelerate the completion of development tasks using approaches such as "follow the sun" (RAMASUBBU et al. 2011).

The key implication of spatial dispersion is the reduction in impromptu communication, which leads to lower levels of awareness of what the distributed project members are doing, the decisions that they are making, or the problems that they might have. Such a barrier to communication and awareness could result in coordination breakdowns and software integration problems, which in turn could lead to longer time to complete development tasks, increase in rework, and/or higher numbers of defects.

Viewpoints:

Increasing in spatial and/or temporal separation among the software team's members may negatively affect software productivity.

Measures found in studies:

Name	Specification
COCOMO II's Multi site development driver (NGUYEN et al. ,2011)	Six point Likert-based scale, assuming the following values: ● Very low → Some phone, mail; ● Low → Individual phone, FAX; ● Nominal → Narrowband email; ● High → Wideband electronic communication; ● Very high → Wideband elect. communication, occasional video conf.; ● Extra high → Interactive multimedia.

Dispersion Measure	**Description**	**Formula** i-j: a location pair, K– no of location pairs, n_i – team size at location i, n_j – team size at location j, N – total team size in the project	**Example of calculation** Team size 10, distributed across 3 locations (location 1=New York; 2=Frankfurt; 3=Bangalore) with 5 people located in New York, 3 in Frankfurt, and 2 in Bangalore.
Separation *(spatial)*	Geographic distance among team members	$\frac{\sum_{i-j}^{k} Miles_{i-j} * n_i * n_j}{(N^2 - N)/2}$	Separation Difference = ((3858*5*3) + (8316*5*2)+ (4607*3*2)) / ((100-10)/2) = 3748.27
Time Zone *(temporal)*	Time difference among team members	$\frac{\sum_{i-j}^{k} Time\ Zone_{i-j} * n_i * n_j}{(N^2 - N)/2}$	Time Zone Difference = ((5*5*3)+(9*5*2)+(3*3*2))/((100 -10)/2) = 4.07
Number of Sites *(configurational)*	Number of locations where team members work	K = Total number of development sites used in the project	Number of Sites = 3
Personnel Imbalance *(configurational)*	Extent of unevenness in distribution of personnel across locations	[Standard Deviation (n_i, n_j,… ,n_k)] / N	Imbalance Difference = [Standard deviation (5,3,2)] / 10 = 0.15
Experience Spread *(configurational)*	Extent of unevenness in work experience of personnel across locations	[Standard Deviation (Average Team Experience at location i, Average Team Experience at location j,…,Average Team Experience at location k)] / N Note: The experience value for an individual is defined as the number of years that the individual has worked in professional software development.	Assuming that the average team experience at New York = 10, Frankfurt = 15, Bangalore =5, Experience Difference = [Standard Deviation (10,15,5)]/10 = 0.5

(RAMASUBBU & BALAN, 2007)

(RAMASUBBU et al. 2011.)

Articles:

References

AVRITZER, Alberto; LIMA, Adailton. An empirical approach for the assessment of scheduling risk in a large globally distributed industrial software project. In: **Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on**. IEEE, 2009. p. 341-346.

NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. **An analysis of trends in productivity and cost drivers over years. In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.

RAMASUBBU, Narayan; BALAN, Rajesh Krishna. Globally distributed software development project performance: an empirical analysis. In: **Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. ACM, 2007. p. 125-134.

RAMASUBBU, Narayan et al. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: **Proceedings of the 33rd international conference on Software engineering**. ACM, 2011. p. 261-270.

Code/Factor: Team Size

Definition:

It represents the number of developers involved in the software project.

Theoretical Basis:

In this context, the observations (studies) under the lens of the Brook's law or others theoretical statements concerning team size must be considered.

Viewpoints:

The number of developers and its variation over time can affect the software productivity. Moreover, it comprises the understanding if there is an optimal value of team size that maximizes software productivity.

Measures found in studies:

Name	Specification
Team size (MORASCA & RUSSO, 2001)	Ratio scale, measured as the number of developers in the software project team.
Inter-releases Team Size (SCHOLTES et al., 2016)	Number of developers who have committed their contributions in the software source code between releases.
Average team size (RAMASUBBU et al., 2011)	Ratio scale, measured as the effort (Person month) divided by project duration.
Maximum Team Size (JEFFERY, 1987) (MAXWELL et al., 1996) (RODRIGUEZ et al., 2012) (WANG et al., 2008)	Ratio scale, measured as maximum number of developers in the software project team.

Articles:

References
AHMED, A. et al. Agile software development: Impact on productivity and quality. In: **Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on**. IEEE, 2010. p. 287-291.
FARSHCHI, Mostafa; JUSOH, Yah Yusmadi; MURAD, Azmi Azrifah Masrah. Impact of personnel factors on the recovery of delayed software projects: A system dynamics approach. **Computer Science and Information Systems**, v. 9, n. 2, p. 627-652, 2012.
JEFFERY, D. Ross. A software development productivity model for MIS environments. **Journal of Systems and Software**, v. 7, n. 2, p. 115-125, 1987.
MAXWELL, Katrina D.; VAN WASSENHOVE, Luk; DUTTA, Soumitra. Software development productivity of European space, military, and industrial applications. **IEEE Transactions on Software Engineering**, v. 22, n. 10, p. 706-718, 1996.
MORASCA, Sandro; RUSSO, Giuliano. An empirical study of software productivity. In: **Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International**. IEEE, 2001. p. 317-322.
RAMASUBBU, Narayan et al. Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: **Proceedings of the 33rd international conference on Software engineering**. ACM, 2011. p. 261-270.
RODRIGUEZ, D. et al. Empirical findings on team size and productivity in software development. **Journal of Systems and Software**, v. 85, n. 3, p. 562-570, 2012.
SCHOLTES, Ingo; MAVRODIEV, Pavlin; SCHWEITZER, Frank. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. **Empirical Software Engineering**, v. 21, n. 2, p. 642-683, 2016.
WANG, Hao; WANG, Haiqing; ZHANG, Hefei. Software productivity analysis with CSBSG data set. In: **Computer Science and Software Engineering, 2008 International Conference on**. IEEE, 2008. p. 587-593.

Code/Factor: Use of Tools

Definition:

It represents the indication of the used software technologies supporting the software development process.

Theoretical Basis:

Some empirical studies have tried to find the correlation among the characteristics of product and software productivity. One of these characteristics is use of tools related to programming, design, integration practices, and it may be a factor affecting software productivity in software development.

Viewpoints:

The use of different tools in software development can produce an observable effect on software productivity.

Measures found in studies:

Name	Specification
Use of Software Tools (MAXWELL et al., 1996) (MAXWELL & FORSELIUS,2000) (NGUYEN et al., 2011)	Based on COCOMO II, a five point Likert-based scale ranging from 1 (very low) to 5 (very high), as follows: ● Very Low → simple use of tools for edit, code and debug; ● Low → simple frontend, backend CASE and little integration; ● Nominal → basic lifecycle tools, moderately integrated; ● High → strong, mature lifecycle tools, moderately integrated; ● Very High → strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse.
Software Testing tools (MOHAPATRA, 2011)	Five point Likert-based scale ranging from 1 to 7, fulfilled by managers using questionnaires.

Articles:

References
MAXWELL, Katrina D.; VAN WASSENHOVE, Luk; DUTTA, Soumitra. Software development productivity of European space, military, and industrial applications. **IEEE Transactions on Software Engineering**, v. 22, n. 10, p. 706-718, 1996.
MAXWELL, Katrina D.; FORSELIUS, Pekka. Benchmarking software development productivity. **Ieee Software**, v. 17, n. 1, p. 80-88, 2000.
MOHAPATRA, Sanjay. Maximising productivity by controlling influencing factors in commercial software development. **International Journal of Information and Communication Technology**, v. 3, n. 2, p. 160-179, 2011.
NGUYEN, Vu; HUANG, LiGuo; BOEHM, Barry. An analysis of trends in productivity and cost drivers over years. In: **Proceedings of the 7th International Conference on Predictive Models in Software Engineering**. ACM, 2011. p. 3.