ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Programa de Pós-Graduação em Engenharia da Computação

Um Benchmarking Framework para Avaliação da Manutenibilidade de Software Orientado a Aspectos

Dissertação de Mestrado

Engenharia da Computação

Marcelo Luís Machado Moura

Orientadores: Prof. Dr. Sérgio Castelo Branco Soares
Prof. Dr. Alessandro Fabrício Garcia

Recife, agosto de 2008

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Um Benchmarking Framework para Avaliação da Manutenibilidade de Software Orientado a Aspectos

Dissertação de Mestrado

Engenharia da Computação

Esta Dissertação é apresentada como requisito parcial para obtenção do título de Mestre em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Marcelo Luís Machado Moura

Orientadores: Prof. Dr. Sérgio Castelo Branco Soares
Prof. Dr. Alessandro Fabrício Garcia

Recife, agosto de 2008

UNIVERSIDADE
DE PERNAMBUCO

Marcelo Luís Machado Moura

Um Benchmarking Framework para Avaliação da Manutenibilidade de Software Orientado a Aspectos

Resumo

O desenvolvimento de software Orientado a Aspectos (OA) busca melhorar a manutenção de software através da modularização de interesses transversais que, caso contrário, são implementados de forma espalhada e entrelaçada. Entretanto, estudos empíricos sobre a avaliação da manutenibilidade de softwares OA ainda são bastante raros. Ainda pior, raramente eles são replicados por diferentes grupos de pesquisadores, dificultando a análise das suas vantagens e desvantagens, e conseqüentemente, atrapalhando o progresso da área. Um dos principais motivos é a ausência de metodologias e mecanismos adequados para apoiar a elaboração do projeto de estudos empíricos e *benchmarks*. Tais metodologias e mecanismos permitem que os estudos e *benchmarks* sejam replicados e reutilizados por toda a comunidade. Esta dissertação apresenta um Benchmarking Framework (BF) responsável por guiar pesquisadores e profissionais da área em pelo menos dois tipos de atividades: (i) no desenvolvimento, adaptação e avaliação de aplicações *benchmark* para manutenção de software OA, e (ii) na elaboração, avaliação e replicação de estudos empíricos sobre manutenibilidade de softwares OA. O BF define normas e critérios para avaliar a representatividade das características de estudos empíricos e de aplicações OA candidatas a *benchmark*, incluindo seus cenários de manutenção. A eficácia do BF é avaliada através da sua utilização em dois contextos, onde cada um deles utiliza duas aplicações OA distintas, um sistema baseado na web e uma linha de produto de software para dispositivos móveis. A sua primeira utilização é relacionada à adaptação das aplicações durante elaboração de um novo estudo empírico sobre a manutenibilidade de software OA. E a segunda é destinada a avaliação das características das aplicações, verificando se estas podem ser consideradas como *benchmark* para manutenção de software OA. Em ambos os casos, as aplicações escolhidas foram utilizadas em vários experimentos reais.

Palavras-chave: Desenvolvimento de Software Orientação a Aspectos, Manutenção de Software, Estudos Empíricos.

Abstract

The Aspect-Oriented (AO) software development aims to improve software maintenance through the modularization of crosscutting concerns, which, otherwise, are implemented in a scattered and tangled way. However, empirical studies assessing the maintainability of AO software are still very limited. Even worse, they are rarely replicated by different groups of researchers, hindering the evaluation of its benefits and drawbacks, and consequently, hampering the progress of the field. One of the key reasons is the lack of appropriate methodologies and mechanisms to support the designing of empirical studies and benchmarks. These methodologies and mechanisms enable replicating and reusing the studies and benchmarks across the community. This dissertation presents a Benchmarking Framework (BF) for guiding researchers and practitioners in, at least, two types of activities: (i) the development, adaptation and evaluation of benchmark applications for AO software maintenance, and (ii) the elaboration, evaluation and replication of empirical studies addressed to AO software maintainability. The BF defines guidelines and criteria to assess the representativeness of empirical studies and AO benchmark candidate applications characteristics, including their maintenance scenarios. The effectiveness of the BF is assessed through its appliance in to two contexts, where each one uses two different AO applications, a web-based system and a software product line to mobile devices. Its first use is the adaptation of these applications during the elaboration of a new experiment about the AO software maintainability. The second use addresses the evaluation of applications characteristics, verifying if it could be considered as benchmark applications to AO software maintenance. In both cases, the selected applications were applied at a number of real experiments.

Keywords: Aspect-Oriented Software Development, Software Maintenance, Empirical Studies.

Sumário

Índice de Figuras

Índice de Tabelas

Agradecimentos

Primeiramente, gostaria de agradecer a Deus por ter me dado saúde para chegar até aqui e também a minha família, por sempre ter acreditado em mim e me dado todo o apoio necessário para realização dos meus sonhos. Agradeço a minha namorada, Carolina Jordão, pela cumplicidade, paciência e compreensão nos momentos de mau humor.

Agradeço ao meu orientador, Sérgio Soares, que sempre esteve disponível e ajudando muito além da sua obrigação acadêmica, se mostrando um amigo de verdade. Ao meu co-orientador, Alessandro Garcia, que mesmo distante, teve uma participação fundamental no desenvolvimento deste trabalho.

Agradeço ao amigo e companheiro de mestrado, Mário Monteiro, que sempre esteve ao meu lado nessa caminhada. Ao professor Fernando Castro Filho, por dar sugestões sempre que questionado. Aos alunos de iniciação científica, Diego Araújo e Elliackin Figueiredo, por ajudarem nas coletas das métricas.

Agradeço aos companheiros de mestrado pela solidariedade e união durante essa jornada, tanto nos momentos de descontração como nos mais difíceis. Agradeço também a presteza e dedicação de Georgina, responsável por resolver todos os problemas administrativos com relação ao mestrado.

Agradeço a todos os Amigos de infância, do colégio, da graduação e da vida, pela constante motivação e exemplos de perseverança que me ajudaram na obtenção dos meus objetivos, vocês fazem parte da minha família.

A todos, o meu muito obrigado, saibam que esta conquista também é de vocês.

Capítulo 1

Introdução

Este capítulo apresenta e discute a importância da geração sistemática de estudos empíricos sobre manutenibilidade de softwares orientados a aspectos. O capítulo apresenta a motivação para a pesquisa, os objetivos desejados, as contribuições obtidas e a organização da dissertação.

1.1 Motivação

O Desenvolvimento de Software Orientado a Aspectos (DSOA) [37] ainda é relativamente recente, porém, o mesmo está sendo cada vez mais adotado na implementação de aplicações de diferentes domínios, como por exemplo, sistemas *web* [68][83][94], *middleware* [19] e linhas de produto [1][36]. Esta tecnologia apóia a simplificação da manutenção de software através da modularização de interesses transversais.

No entanto, faltam comprovações e evidências científicas que atestem sobre as vantagens e desvantagens associadas à manutenibilidade do DSOA. Avaliações neste sentido geralmente não recebem a devida importância, mesmo se tratando de uma tarefa essencial. Pois, ao considerar novas tecnologias, como DSOA, a preparação e replicação de estudos empíricos sobre manutenibilidade é um processo lento e complexo, justamente por não haver outros similares. E de fato, avaliações empíricas e suas replicações são realmente escassas, não só no contexto de manutenção de software orientado a aspectos (OA) [36][45][64], como de forma geral para a área de engenharia de software [103].

Empresas e organizações de softwares precisam de informações que apóiem suas decisões, como por exemplo, o uso ou não de uma determinada tecnologia para o

desenvolvimento do software. Essas decisões visam atender os objetivos almejados, que podem ser a melhora da qualidade do produto ou a redução dos custos associados ao seu desenvolvimento, por exemplo.

Porém, na maioria dos casos da engenharia de software, a única maneira real de avaliar essas questões é colocando-as em prática, ou seja, fazendo com que pessoas utilizem a tecnologia proposta durante o desenvolvimento de um software real, para que se possa avaliá-la posteriormente. Por isso, apesar de custosas e demoradas, pesquisas científicas, através de experimentos e estudos empíricos, são de fundamental importância durante a avaliação de novas tecnologias utilizadas no desenvolvimento, e permitem que os resultados obtidos acelerem o crescimento e disseminação do conhecimento na área.

Um obstáculo fundamental para avanço da área de engenharia de software, e mais especificamente para a tecnologia de Orientação a Aspectos (OA), está associada aos primeiros passos necessários para avaliação empírica da manutenibilidade do DSOA: a seleção, elaboração e adaptação sistemática de aplicações OA representativas. Pois, exemplos de *benchmarks* [32][91] não tiveram boa aceitação pela comunidade científica para manutenção de softwares OA.

Pois, diferentemente de outras áreas que possuem um avanço científico considerável, pesquisadores e profissionais de OA não estão equipados adequadamente com ferramentas metodológicas para guiar a construção de tais *benchmarks*. Aplicações *benchmark* são os mecanismos empíricos básicos para avançar pesquisas e disseminar tecnologias em determinados campos da engenharia de software [92][103].

No entanto, a criação de um *benchmark* para o contexto de manutenção de software OA além de lidar com as dificuldades relacionadas à área de estudos empíricos, também deve superar os obstáculos relacionados à manutenção de software OA. Por isso, a criação de tal *benchmark* é uma tarefa complicada em vários sentidos.

Em primeiro lugar, a variedade dos domínios de aplicações OA tem aumentado rapidamente. Inicialmente a Programação Orientada a Aspectos (POA) estava destinada apenas a melhorar a modularidade, e conseqüentemente a manutenibilidade, de interesses clássicos que possuíam o escopo muito abrangente em aplicações distribuídas [59][94], tais como, controle de concorrência e tratamento de erros. Mas, posteriormente a mesma foi utilizada para atender a objetivos bem diferentes, como melhorar a adaptabilidade de *middlewares* [19][28] e aumentar a estabilidade de linhas de produto [36].

Adicionalmente, os mecanismos considerados relacionados ao campo de DSOA [37] e as suas híbridas manifestações em linguagens de programação emergentes [3][71][100] estão crescendo constantemente. E, finalmente, estudos sobre a manutenibilidade de novas técnicas de desenvolvimento, como a OA, necessitam de análises detalhadas a respeito de diversos fatores tipicamente presentes em atividades de manutenção de software, tais como diferentes tipos de mudança.

Portanto, baseado nesses pontos, existe uma necessidade urgente de apoiar metodologicamente a construção de aplicações benchmark, ao invés de limitar a análise, considerando apenas poucos “casos universais”.

1.2 Objetivos

O principal objetivo desta dissertação é a definição de um Benchmarking Framework (BF) para auxiliar a preparação e padronização de avaliações empíricas sobre a manutenibilidade de técnicas de DSOA, acelerando a geração de evidências empíricas na área.

O BF define critérios, normas e diretrizes apropriadas para avaliar, através de aplicações *benchmark*, características sobre a manutenibilidade de técnicas de OA. O mesmo é responsável por guiar pesquisadores e profissionais da área na seleção, desenvolvimento e adaptação de aplicações benchmarks e seus respectivos cenários de manutenção. Com o intuito de que suas características sejam adaptadas da melhor forma para atender os objetivos experimentais. Por isso, o BF, através de seus critérios, também pode ser utilizado para guiar o planejamento de novos estudos empíricos, assim como, a replicação e avaliação de estudos empíricos existentes.

Outro objetivo deste trabalho é a realização de avaliações utilizando diferentes configurações e aplicações de domínios distintos, para analisar a eficácia do BF ao atender seus objetivos propostos.

Portanto, espera-se que esta iniciativa reduza a ausência de resultados e comprovações científicas na área de OA, uma vez que este trabalho está destinado a acelerar a geração sistemática de estudos empíricos, e suas replicações, envolvendo manutenibilidade de softwares OA. E conseqüentemente, auxiliando a tomada de decisão dos *stakeholders*, que precisam avaliar as vantagens e desvantagens da adoção de técnicas de OA em determinadas circunstâncias.

1.3 Contribuições

Os seguintes itens compõem a lista das principais contribuições deste trabalho:

- Definição de um Benchmarking Framework (BF):
  - Definição de normas, diretrizes, componentes e critérios representativos para avaliação de características sobre a manutenibilidade de softwares OA;
- Utilização do BF por Projetistas de Estudos Empíricos e Projetistas de Benchmark:
  - Guiar a seleção, criação e adaptação de aplicações benchmark e seus respectivos cenários de manutenção, dado os objetivos experimentais;
  - Guiar o planejamento de novos estudos empíricos, e auxiliar replicações e avaliações de estudos empíricos existentes;
  - Guiar avaliações das características de manutenibilidade de técnicas de DSOA;
  - Possibilidade de extensão dos componentes e critérios definidos para que atendam a domínios e objetivos experimentais não cobertos inicialmente;
- Avaliações da utilização do BF:
  - Utilizando o BF para planejar novos experimentos;
  - Utilizando o BF para identificar aplicações e cenários benchmark;

1.4 Organização da Dissertação

Esta dissertação possui 6 capítulos, incluindo este, e está organizada da seguinte forma. O Capítulo 2 apresenta questões gerais sobre estudos empíricos e motiva a sua aplicação na área de engenharia de software. O Capítulo 3 apresenta discussões relacionadas aos temas de manutenção de software, orientação a aspectos e estudos empíricos, que são destinados a avaliar e relacionar esses temas. Os capítulos 2 e 3 compõem a base necessária para o entendimento desta dissertação. O Capítulo 4 apresenta a maior contribuição deste trabalho, a definição do Benchmarking Framework (BF) para avaliar a manutenibilidade de softwares OA. O Capítulo 5 apresenta e analisa avaliações sobre a utilização do BF e sua eficácia. O Capítulo 6 apresenta as conclusões deste trabalho, incluindo a comparação do BF com alguns trabalhos relacionados e a descrição dos trabalhos em andamento e futuros.

Capítulo 2

Estudos Empíricos em Engenharia de Software

Este capítulo discute a importância da realização de estudos empíricos na área de engenharia de software, de forma a tornar mais sistemática a avaliação de técnicas e métodos propostos para esta área. Os principais conceitos associados à experimentação e aceleração da criação de evidências científicas, tais como estudos empíricos e suas replicações, *frameworks*, *benchmarks* e *testbeds*, são brevemente descritos através de exemplos específicos para engenharia de software.

2.1 Estudos Empíricos e Engenharia de Software

Apesar da criação de uma área específica, a Engenharia de Software [80][82][97], que visa facilitar e controlar o desenvolvimento de sistemas de software, as atividades envolvidas nesse processo continuam fortemente dependentes da participação e criatividade humana. Esses fatores diminuem a confiabilidade de resultados científicos na área e impedem que ela se torne uma ciência exata. Por isso é necessário utilizar métodos empíricos que auxiliam a comprovação de tais resultados.

Analogamente à definição de engenharia de software [51], experimentações permitem de maneira sistemática, disciplinada, quantificável e controlada, avaliar atividades realizadas por pessoas [104]. Essa é uma das principais razões para as pesquisas empíricas serem tão comuns em área de ciência sociais e comportamentais [87], as quais estão intimamente interessadas e relacionadas ao comportamento humano. Tal característica dificulta a avaliação de resultados, pois não é possível encontrar leis da natureza, como na Física, que guiem e provem com exatidão

a maioria das hipóteses nessas áreas. Isto justifica a necessidade de experimentos científicos e estudos empíricos para garantir a veracidade de tais resultados.

Portanto, uma observação importante é que, assim como na área de ciências sociais e comportamentais, a engenharia de software também é fortemente influenciada pelo comportamento humano, através das pessoas que desenvolvem software. Isso faz com que essas diferentes áreas se relacionem sobre esse contexto, e possibilita que a experiência existente nessas outras áreas guie novas pesquisas científicas em engenharia de software.

Além disso, experimentos e estudos empíricos são importantes para pesquisadores de qualquer área, inclusive os de engenharia de software. Novos métodos, técnicas, processos, linguagens e ferramentas não devem ser apenas sugeridas, publicadas e comercializadas. Antes de qualquer adoção é extremamente necessário que novas invenções e sugestões sejam rigorosamente avaliadas e comparadas com outras já existentes, para que os riscos associados sejam bem conhecidos. A experimentação provê esta possibilidade, tornando a área pesquisada mais científica e confiável.

2.2 Estudos Empíricos em Engenharia de Software

Empresas e organizações de softwares precisam de informações que apóiem suas decisões, como por exemplo, o uso ou não de determinado processo ou de uma tecnologia específica para o desenvolvimento do software, visando melhorar a qualidade do produto e reduzir os custos associados ao seu desenvolvimento.

Porém a única maneira real de avaliar esse tipo de questão é colocando-a em prática, ou seja, fazendo com que pessoas usem o processo ou a tecnologia proposta durante o desenvolvimento de um software real. Dessa forma, pesquisas científicas, através de experimentos e estudos empíricos, são de fundamental importância durante a avaliação desses processos que descrevem as atividades de desenvolvimento.

A atenção dada aos estudos empíricos na área de engenharia de software começou a entrar em evidência a partir do surgimento de trabalhos que explicitavam a necessidade de experimentação na área [10]. Uma década depois, o assunto foi amplamente discutido por vários trabalhos [12][103][104] que tentavam disseminar a realização de estudos empíricos durante o desenvolvimento de software.

No entanto, várias dificuldades foram encontradas devido às peculiaridades da área. Por ser uma atividade multidisciplinar, a engenharia de software envolve questões de diferentes áreas e não somente aspectos técnicos, como a maioria das engenharias. Por isso, não pode ser tratada e avaliada utilizando os mesmos métodos e procedimentos científicos adotados para as demais. Pois, além de precisar controlar inúmeras variáveis, que na maioria das vezes são conflitantes, como por exemplo, tempo de experiência acadêmica e industrial, a mesma depende intensamente do comportamento e da criatividade humana durante todo o processo de desenvolvimento de software.

Atualmente as atividades associadas à área de engenharia de software, em sua grande maioria ainda envolvem atividades humanas, pois dependem que alguém as realize, tornando-se impossível automatizá-las. Assim, esta área se difere de algumas ciências exatas, onde é possível encontrar leis universais e regras formais que possibilitam a automação de determinadas atividades. Além de implicar na provável redução de tempo e esforço gastos na execução das tarefas, a automação também unifica e garante que várias atividades sejam desenvolvidas da mesma forma, evitando inconsistências que poderiam ser geradas por diferentes pessoas.

Por não ser uma engenharia exata, devido à dificuldade e complexidade de depender do comportamento de humano, em que cada pessoa possui diferentes habilidades, experiências e conhecimentos técnicos, e de controlar variáveis conflitantes durante todo o processo de desenvolvimento; estudos empíricos sólidos e controlados ainda são muito raros na área de engenharia de software [102][106].

Podem-se listar como principais causas dessa escassez:

- A dificuldade em padronizar e encapsular todas as informações necessárias aos estudos;
- A dificuldade em controlar todas as variáveis envolvidas no desenvolvimento de software, tais como processos, tecnologias, experimentos e fatores humanos em geral;
- O envolvimento humano direto na execução dos estudos, que gera uma alta probabilidade de contaminação dos resultados, uma vez que é impossível selecionar para o estudo, pessoas que tenham exatamente as mesmas características;
- Os altos custos associados na alocação de pessoas para realizar experimentos durante todo o processo de desenvolvimento de um software;

Essa carência de estudos empíricos em engenharia de software é extremamente danosa, pois atrapalha e dificulta o surgimento de novas tecnologias, conseqüentemente atrasando sua transferência para indústria e sua adoção pela comunidade. A transferência de uma tecnologia demora muito tempo para ser realizada com sucesso, em média 18 anos até ser largamente disseminada pela indústria e comunidade técnica [85]. Por isso, a falta de resultados e de evidências científicas torna-se um obstáculo à tomada de decisão por parte dos *Stakeholders* que representam a indústria de software [52]. Isto se deve a falta de capacidade de avaliar com segurança, os reais riscos e benefícios da adoção de novas tecnologias.

2.3 Replicações de Estudos Empíricos

Com o objetivo de reduzir essa lacuna de comprovações científicas na área, e conseqüentemente acelerar o avanço na transferência de tecnologia, ações que visam o estímulo para geração de estudos empíricos são de grande valia. Pode-se citar como exemplo, a iniciativa de realizar replicação [16] de estudos já existentes através da criação de famílias de estudos empíricos [50]. Desta forma, o mesmo estudo é realizado mais de uma vez, aumentando a confiabilidade de seus resultados, e permitindo que estudos relacionados sejam executados a partir de outros já existentes.

Porém, apenas tais ações não são suficientes para gerar uma série de estudos empíricos, pois, além das dificuldades mencionadas sobre a área de engenharia de software, os pesquisadores encontram problemas em replicar, planejar e analisar estudos sem a utilização de guias, processos e técnicas adequadas que os auxiliem na tarefa. Por isso, existe a necessidade de se definir novos modelos e técnicas que tenham como objetivo ajudar os pesquisadores a realizar tais atividades [11][12], e conseqüentemente, aumentar a quantidade de estudos empíricos realizados na área.

Dessa forma, trabalhos são destinados exclusivamente para discutir, propor e avaliar regras de documentação e publicação de estudos empíricos, com o objetivo de promover o acesso às informações e facilitar suas replicações [16][53]. A padronização da documentação disponível de um experimento é essencial para que replicações sejam realizadas adequadamente, por exemplo, decisões de pesquisa que não são informadas devidamente dificultam o entendimento do mesmo e fatalmente atrapalham ou impedem a sua replicação.

2.4 Benchmarks

Outra forma eficiente de se obter a aceleração na geração de estudos empíricos, e bastante comum em outras áreas, é através da alteração dos ambientes de planejamento, execução e avaliação dos experimentos, para que os mesmos passem a utilizar guias, frameworks e benchmarks com enfoque em engenharia de software. Através do uso de tais estruturas, é possível melhorar a organização, documentação e encapsulamento dos experimentos. Conseqüentemente, promovendo maior facilidade de replicação e criação de famílias de estudos empíricos.

Um benchmark pode ser definido como uma maneira conveniente de encapsular as informações essenciais e procedimentos (hipóteses, tratamentos, variáveis independentes, objeto de controle, objeto do estudo, resultados e demais componentes de um experimento) utilizados na execução de estudos empíricos, podendo ser usado para responder diversas questões e diferentes objetivos [91]. Um importante benefício do uso dos benchmarks é que os mesmos tornam possível responder essas questões de forma objetiva, além facilitar a análise, comparação e replicação de diferentes experimentos.

Tichy [103] define benchmark como uma amostra de tarefas de um determinado domínio, executadas por um computador, ou por um humano e um computador. Durante essa execução realizam-se medições bem definidas sobre desempenho. Porém, este trabalho apresenta outra definição para benchmark que será detalha no Capítulo 4.

Além dos estudos apresentados, , é possível encontrar outros exemplos, bem sucedidos, de benchmarks empregados em engenharia de software [92].

2.5 Testbeds

Além de *benchmarks*, outro importante conceito relacionado é o de *testbed*. Entretanto, este termo praticamente não possui definições no contexto de engenharia de software, exceto por alguns trabalhos que tentam preencher essa lacuna [14][44][67]. Esses trabalhos propõem o esforço inicial em direção ao uso do *testbed*, porém, sem definir explicitamente o termo em seu domínio específico.

Já em outras áreas é possível encontrar definições mais detalhadas sobre *testbeds*, tais como em sistemas de previsão do tempo [30]. Dabberdt e outros definiram *testbed* de forma

genérica, onde a lacuna abaixo deve ser preenchida de acordo com área estudada, possibilitando a utilização desta definição em outros contextos. Portanto, em tal definição o termo *Testbed* seria:

> *"Uma relação de trabalho entre um framework semi-operacional e especialistas em medições, pesquisadores e indústria; que tem como objetivo resolver problemas práticos e operacionais _____ que possuam uma forte ligação com usuário final. Os resultados obtidos a partir do testbed são: sistemas mais eficazes, melhora na análise de dados, melhores serviços e produtos, e benefícios econômicos e sociais. Testbeds aceleram a tradução de pesquisa e desenvolvimento (P&D) em melhores operações, serviços, e tomadas de decisão."*

Com isso, é possível adaptar essa definição para o contexto de engenharia de software e mais especificamente para o contexto dessa dissertação, onde *Testbed* seria:

> *"Uma relação de trabalho entre um framework semi-operacional e especialistas em medições, pesquisadores e indústria; que tem como objetivo resolver problemas práticos e operacionais* ***da engenharia de software*** *que possuam uma forte ligação com usuário final.* ***O testbed possui uma coleção organizada e sistemática de benchmarks, onde cada um deles é projetado com base em um benchmarking framework, o qual se destina a tratar de problemas específicos do seu respectivo domínio****. Os resultados obtidos a partir do testbed são: sistemas mais eficazes, melhora na análise de dados, melhores serviços e produtos, e benefícios econômicos e sociais. Testbeds aceleram a tradução de pesquisa e desenvolvimento (P&D) em melhores operações, serviços, e tomadas de decisão."*

Portanto, esta definição adaptada difere da original nos seguintes pontos: (i) por ser específica a área de engenharia de software, e (ii) por indicar que o *testbed* possui um conjunto de *benchmarks* elaborado através de um benchmarking framework, específico para cada tipo de domínio avaliado, no caso desta dissertação o mesmo é focado na manutenibilidade de softwares OA, como mostrado em detalhes no Capítulo 4.

2.6 Considerações Finais

Este capítulo discutiu a importância e necessidade da realização de estudos empíricos na área de engenharia de software, assim como, mostrou exemplos de guias e modelos que podem ser utilizados para direcionar e auxiliar a geração de experimentos nesta área.

A aplicação das estruturas citadas é extremamente essencial à geração de evidências, comprovações e resultados científicos. Pois, elas proporcionam a construção de uma base conhecimento, necessária para o avanço e aceleração do nível de maturidade das tecnologias utilizadas e da engenharia de software como um todo [13].

Capítulo 3

Manutenção de Software Orientado a Aspectos

Neste capítulo são apresentadas características da manutenção de software e da orientação a aspectos, assim como, os supostos benefícios que esta técnica de programação pode promover à qualidade do software, e conseqüentemente, a atividade de manutenção. Também são descritos exemplos que avaliam cientificamente cada um desses temas e os resultados da possível colaboração existente entre eles, reduzindo assim, a ausência de estudos empíricos que comprovem tais benefícios.

3.1 Manutenção de Software

Após o término do período de desenvolvimento do software e todo esforço envolvido em sua construção, pode-se imaginar que o pior já passou quando o mesmo é entregue ao cliente, porém, na realidade não é o que acontece.

Todo software precisa sofrer alterações após seu desenvolvimento, independente do seu tamanho ou domínio. Com o passar do tempo, fatalmente os requisitos originais precisam ser modificados para se adequarem a solicitações de mudança e novas necessidades dos clientes e usuários. O ambiente de um sistema pode ser obrigado a evoluir através da utilização de novos hardwares, e conseqüentemente, o seu software precisará de mudanças para se adaptar a esse novo contexto. Além disso, questões acidentais como erros que não foram identificados durante as fases de testes, podem ser percebidos pelo usuário e necessitarem de correção.

Ou seja, manutenção consiste em todo e qualquer processo de alteração de um sistema após a sua implantação. Essas mudanças podem envolver desde simples correções no código fonte, a mudanças mais abrangentes para corrigir erros de projeto ou realizar melhorias significantes, tais como, correção de erros de especificação ou adaptação a novos requisitos (tipos de mudança serão discutidos em detalhes no Capítulo 4).

Ao observar o ciclo de vida de softwares existentes, pode-se concluir que a atividade de manutenção é mais cara e demorada de todo o processo de desenvolvimento de software [80][82][97]. Isso se explica devido à falta de planejamento antes do desenvolvimento, que conseqüentemente, gera softwares mal projetados e difíceis de manter e evoluir. Além de se agravar durante o desenvolvimento pela ausência de boas práticas de projeto, tais como padrões de projetos, reuso, baixo acoplamento e alta coesão, que se não são colocadas em prática, na maioria dos casos tendem a aumentar a complexidade do código fonte e dificultar sua manutenção.

Por isso, é essencial estimular a criação e análise de novas técnicas, métodos, processos, tecnologias e paradigmas que visem facilitar e reduzir os custos da atividade de manutenção (Seção 3.2), tais como a Programação Orientada a Aspectos (POA) [35][58] (Seção 3.3). De forma que seja possível identificar quais são as abordagens mais adequadas para atender determinados cenários de mudança, em determinados domínios.

3.2 Estudos Empíricos sobre Manutenção

Com o objetivo de melhorar a manutenibilidade do software é preciso identificar quais características do produto, ambiente e equipe afetam a manutenção. Como ilustrado pelos estudos a seguir, tal necessidade motiva especificamente a realização de estudos empíricos na área de manutenção de software.

Além dos fatores já citados, prejudiciais a manutenção de software, ainda existem outros mais gerais, que estão diretamente relacionados ao esforço envolvido nas atividades de manutenção. Por isso é importante identificá-los e analisá-los detalhadamente para facilitar o entendimento sobre os mesmo e, conseqüentemente, permitir que os softwares sejam mantidos de maneira mais eficiente. Segundo alguns estudos [7][69], esses fatores incluem características dos sistemas, tais como estrutura, tamanho, idade, quantidade de dados de entrada e saída, tipo de aplicação, e linguagem de programação. Por exemplo, sistemas grandes e complexos tendem a

exigir maior esforço de manutenção quando comparados com sistemas menores e mais simples. Isso acontece porque quanto maior a aplicação, mais tempo é necessário para entendê-la, e também devido à grande diversidade de funcionalidades que aplicações complexas costumam ter [56].

Com o intuito de ampliar a discussão sobre os fatores causadores de problemas na manutenção, outros estudos [8][9] foram realizados para investigar a relação entre determinados tipos de mudanças e os efeitos gerados por elas. Muitas vezes essas características costumam se repetir, formando padrões de causa e conseqüência, que podem ser associados aos fatores citados anteriormente. Ou seja, uma mudança específica, quando realizada em aplicações de determinados domínios ou aplicações que foram construídas em certas linguagens de programação, pode não ser bem sucedida e vir a afetar negativamente a qualidade da aplicação.

Ambos os estudos citados acima indicaram a necessidade de utilização de processos e modelos específicos durante a fase de manutenção, o que ajuda no controle das variáveis do ambiente. Porém, tais estudos não são tão abrangentes no que diz respeito à geração de evidências empíricas, bem como, em relação à análise de métodos e técnicas adequadas para manutenção. Temas mais abordados pelos estudos a seguir.

Kemerer [57] apresenta uma abordagem detalhada para a realização de estudos empíricos na área de manutenção de software, através da investigação de 23 sistemas desenvolvidos e mantidos ao longo de 20 anos (1980 a 2000). O estudo descreve um *survey* contendo a utilização de técnicas e métodos em diversas aplicações reais, e posterior análise das mesmas sob diferentes contextos. Porém, apesar de ser uma boa contribuição, o trabalho possui algumas limitações que foram levantadas pelos próprios autores, como por exemplo, a necessidade de aumentar a base de conhecimento na área através da realização de mais estudos empíricos. Possibilitando realizar avaliações e comparações, mais representativas, utilizando os resultados obtidos e, conseqüentemente, aumentando o entendimento sobre o domínio do problema.

Como já citado no Capítulo 2, a geração e replicação de estudos empíricos na área de engenharia de software é uma tarefa árdua e repleta de desafios. E se tratando de manutenção de software, a situação se complica ainda mais, pois, a mesma está entre as atividades mais caras de todo ciclo de vida do desenvolvimento de software. E, além disso, realizar experimentos sobre a manutenção de um determinado software, obviamente exige que este seja projetado e implementado previamente. Porém, na maioria dos casos, são raros os softwares que possuem características ou, sejam padronizados adequadamente para serem utilizados em vários experimentos, principalmente no caso de técnicas emergentes de engenharia de software, como

POA (Seção 3.3), a identificação dessas características se torna mais difícil pela escassez de estudos empíricos. Dessa forma, o desenvolvimento do software passa a ser associado aos custos do experimento, que precisa de aplicações alvo que atendam seus requisitos.

Por isso, com objetivo de acelerar o processo de geração de evidências empíricas, a experiência de áreas relacionadas na definição de guias, modelos e metodologias de avaliação experimental é essencial para o avanço das pesquisas em manutenção de software. A seguir são apresentados estudos que mostram a utilização de *frameworks, benchmarks e testbeds* focados nas atividades de manutenção.

3.2.1 Frameworks

Esta seção apresenta frameworks dedicados as atividades de manuteção de software. Como citado no Capítulo 2, frameworks podem definidos como um conjunto de regras e atributos responsáveis por auxiliar a realização e avaliação de processos, técnicas ou métodos de uma determinada área.

Porém, pela dificuldade em se definir um framework genérico para área de manutenção de software, que ao mesmo tempo atenda a vários domínios específicos, pesquisas têm sido realizadas a fim de construí-los num escopo mais limitado, restringindo seu uso a determinados domínios, mas aumentando seu nível de detalhamento. Exemplos de tais Frameworks [62][70] são brevemente descritos abaixo.

Kajko-Mattsson e outros [70] definem um framework contendo regras para evolução e manutenção de sistemas baseados em arquitetura orientada a serviços (Service-Oriented Architecture – SOA) [41]. Os autores adaptaram modelos convencionais de manutenção software para as necessidades específicas do domínio pesquisado e, além disso, sugeriram regras e padrões a serem seguidos para que as atividades de manutenção fossem bem sucedidas.

Já Koponen [62] apresenta um framework para guiar o processo de manutenção de softwares *Open Source* [79]. Similarmente ao trabalho anterior, é realizada uma comparação entre os processos de manutenção convencional e o *Open Source*, para que sejam identificados pontos correlacionados. Dessa forma, os autores identificaram atividades fundamentais aos processos e aquelas que são mais comuns e freqüentemente utilizadas, no contexto das aplicações de softwares *Open Source* que foram analisadas.

3.2.2 Benchmarks

Os frameworks apresentados têm como foco principal processos de manutenção de software e suas atividades, porém os mesmos possuem a deficiência de não enfatizar as tecnologias e técnicas adotadas durante tais processos. Por isso, com objetivo de suprir tal ausência e analisar a manutenção de forma mais abrangente, sem a limitação de domínio específico, outros estudos apresentam avaliações mais gerais utilizando benchmarks para manutenção e evolução de software [32][93].

Apesar do estudo de Sjøberg e outros [93] não ser intitulado de *framework* ou *benchmark*, o mesmo possui várias semelhanças, pois utiliza tais conceitos para fazer uma avaliação de tecnologias, técnicas e métodos voltados para manutenção de software. Tal avaliação é feita através de definição de critérios, formas de comparação, regras, princípios e estruturas que são utilizadas. Uma vez que é definido esse conjunto de atributos (*framework*) que possui uma taxonomia capaz de representar a maioria dos casos reais de mudança de software (manutenção). Estudos de caso são avaliados com base nesses atributos. O objetivo é identificar exemplos, que não necessariamente atendam a todos os atributos definidos, mas a um subconjunto típico e representativo (*benchmark*).

Da mesma forma, Demeyer e outros [32] apresentam um *benchmark*, que tem como objetivo permitir a comparação de diversas técnicas utilizadas na manutenção de software. Assim, é possível estabelecer uma forma mais justa e confiável de avaliar tais técnicas, uma vez que o ambiente dos experimentos é configurado de forma padronizada e adequada, além das aplicações e estudos de casos serem escolhidos de forma criteriosa e representativa. Diferentemente do uso exclusivo de frameworks, que apenas fornecem critérios, mas não costumam permitir a comparação com outros casos representativos. Com isso, aumenta-se a autenticidade da verificação do desempenho de determinadas técnicas, bem como, facilita-se a replicação de novos estudos empíricos.

3.2.3 Testbeds

Analogamente aos trabalhos já citados, outro estudo [42] apresenta uma avaliação sistemática da manutenibilidade e do desempenho de diferentes metodologias para realização da persistência de dados em softwares. A sua diferença para os demais é que esse trabalho define um *testbed* e um processo de avaliação contendo métricas específicas com o objetivo de avaliar a

manutenabilidade e desempenho das técnicas de persistência. Apesar de estar restrito a um pequeno domínio, tal estudo pode servir como base para outros exemplos mais gerais de manutenção.

Embora sejam inovadoras na área e contendo uma boa abordagem para avançar a criação de estudos empíricos controlados e com possíveis resultados cientificamente válidos, como o exemplo apresentado nesta seção e nas anteriores, pesquisas que propõem *frameworks*, *benchmarks e testbeds*, devem ser exaustivamente utilizadas, estendidas e avaliadas para que só posteriormente possam ser largamente adotadas pela comunidade. Esse processo de verificação e validação tecnológica é importante para identificar as limitações existentes e sugerir melhorias nos estudos avaliados, pois as técnicas, métodos e tecnologias utilizadas na área estão em constante evolução. Por isso, torna-se praticamente impossível definir uma estrutura, conjunto de atributos ou processo que sejam estáticos e completos. A velocidade da evolução das tecnologias exige que tais conceitos sejam definidos de forma a permitir um futuro aperfeiçoamento para um novo domínio ou uma simples correção de falhas.

3.3 Orientação a Aspectos

Atualmente, a Orientação a Objetos (OO) [15] é o paradigma de desenvolvimento de software mais comum e popular utilizado na comunidade e indústria. Porém, com o constante aumento da complexidade dos softwares, percebeu-se que o mesmo apresenta algumas limitações [77][78], como por exemplo, a dificuldade de modularizar e modificar adequadamente certos tipos de requisitos e interesses do sistema. Esses interesses que não podem ser facilmente modularizados com OO são conhecidos como interesses transversais, pois apresentam características semelhantes de espalhamento e entrelaçamento por todo código fonte, ou seja, transversais ao sistema. Algumas dessas limitações podem ser minimizadas através da utilização de padrões de projeto [39], porém tal solução não atende a maioria dos casos de interesses transversais.

As diversas decisões de projeto que precisam ser tomadas durante o desenvolvimento do software, direcionam como os requisitos não-funcionais (freqüentemente causadores da manifestação de interesses transversais) devem ser implementados no sistema. Porém, na maioria das vezes, a implementação dos mesmos exige que seu código fonte seja espalhado por todos os interesses funcionais do sistema, resultando no entrelaçamento de diferentes interesses.

Dessa forma, com o objetivo de solucionar os problemas encontrados em OO, surgiu a Orientação a Aspectos (OA) [35][58]. Essa nova técnica de desenvolvimento foi criada com a finalidade de permitir uma melhor separação de interesses, e com isso aumentar a modularidade dos interesses transversais de um sistema, característica essa que não pode ser atendida satisfatoriamente apenas com a utilização de OO e padrões de projeto.

Com a separação dos interesses transversais, a OA garante que o código destinado a implementação de cada requisito não irá se misturar e espalhar com o código destinado a outros requisitos do sistema. Os interesses transversais mais comuns em softwares são persistência, distribuição, controle de concorrência, tratamento de exceções e auditoria. Assim, pode-se afirmar que a OA aumenta a modularidade do software através da completa separação de código que pertence interesses semelhantes e que afeta diferentes partes do sistema. Essa característica é essencial para facilitar o desenvolvimento, manutenção e evolução de sistemas, pois reduz a dependência e acoplamento entre os módulos do sistema.

A OA realiza a separação dos interesses transversais através da introdução de uma nova unidade de implementação, o aspecto, que será responsável por encapsular todo código que anteriormente estava espalhado e entrelaçado por outros módulos. Assim, os aspectos passam a ser a única unidade de implementação transversal ao restante do sistema. Para isso os aspectos precisam alterar o comportamento do programa, utilizando o código que antes estava espalhado e entrelaçado, com o objetivo de que a mudança seja transparente e que as funcionalidades do sistema continuem as mesmas.

Tal alteração do comportamento pode acontecer em vários pontos bem definidos na execução de um programa (*join points* ou pontos de junção). Para isso é preciso que regras sejam definidas com intuito de determinar quais pontos do programa serão afetados com a alteração de comportamento (*pointcuts* ou pontos de atuação). O código fonte responsável pela alteração do comportamento também deve ser descrito, juntamente com as regras que foram definidas para identificar quais pontos o mesmo irá afetar (*advices* ou adendos).

Outras características mais específicas podem variar de acordo com a linguagem utilizada, pois, apesar de ser uma tecnologia relativamente nova, a OA já possui diversas linguagens de programação, onde cada uma delas estende uma linguagem de outro paradigma, como programação orientada a objetos (POO) ou procedural. Pode-se citar como exemplo de linguagens orientadas a aspectos populares e promissoras, AspectJ [59] e CeasarJ [71], em que ambas estendem a linguagem de programação Java.

3.4 Estudos Empíricos Sobre OA e Sua Manutenabilidade

O desenvolvimento de software orientado a aspectos (DSOA) [37] tem crescido rapidamente, e está sendo utilizado cada vez mais em aplicações de diferentes domínios, como por exemplo, sistemas *web* [68][83][94], *middleware* [19] e linhas de produto [1][36], com o propósito de simplificar a manutenção de tais aplicações.

Porém, embora o DSOA vise melhorar a manutenção de software através do encapsulamento de interesses transversais, são raras as evidências empíricas que comprovem tal teoria [5]. Pois, por ser uma técnica de desenvolvimento relativamente recente, estudos que avaliem sistematicamente as vantagens e desvantagens associadas à manutenabilidade de softwares orientados a aspectos, ainda são escassos e bastante limitados na literatura.

Por isso, a preparação e replicação de estudos empíricos com essa finalidade, se tornam ainda mais complicados e demorados quando comparados com estudos que avaliam tecnologias mais difundidas, e outras atividades que sejam menos custosas do que a de manutenção. Outro ponto negativo é a ausência de replicação dos estudos por diferentes grupos de pesquisadores, que tem o propósito de aumentar a confiabilidade nos resultados obtidos. Assim, esses fatores implicam no atraso do avanço científico da área.

Como mencionado no Capítulo 2, é de fundamental importância a criação de uma base de conhecimento científica necessária para aceleração da tecnologia, nesse caso da OA. Além da construção de guias e modelos que permitam auxiliar o processo de avaliação sistemática e, conseqüentemente, de transferência de tecnologia. Dessa forma, alguns estudos apresentados a seguir tentam preencher essa lacuna através de avaliações empíricas de softwares orientados a aspectos, e de seus benefícios, que estão diretamente relacionados com a manutenção de tais aplicações.

3.4.1 Estudos empíricos

Alguns estudos descrevem especificamente como deve ser realizada a implementação de determinados interesses transversais, comuns a aplicações de diferentes domínios, como por exemplo, persistência [83][94][96], distribuição [94][96], concorrência [60] e tratamento de exceção [20][21], além de analisar os resultados e impactos obtidos por tais implementações.

Nesse mesmo contexto, outro trabalho [95] apresenta uma avaliação mais geral, contendo a definição de uma técnica específica para implementação orientada a aspectos, capaz de modularizar cada um dos interesses transversais citados.

Além dessas funcionalidades básicas, a implementação de padrões de projetos [39], tarefa que entre outras coisas visa melhorar a manutenção de software, é outra atividade que freqüentemente provoca desafios em aplicações reais, quando se utiliza mais de um padrão. Um dos principais problemas da implementação de múltiplos padrões de projetos em um sistema é que estes não se limitam a afetar apenas os módulos da aplicação (requisitos funcionais), inevitavelmente eles se afetam entre si, impactando de diferentes formas, os módulos destinados as suas implementações específicas. O que ocasiona espalhamento e entrelaçamento de código, tornando assim esses interesses transversais ao sistema. Por isso, sua separação e composição são consideradas tarefas árduas.

Dessa forma, estudos [19][40][48] apresentam avaliações de como uma abordagem utilizando OA, pode prover mais melhorias na separação e composição de múltiplos padrões de projeto utilizados em uma mesma aplicação, quando comparados com soluções de OO. Esse argumento é baseado em comparações realizadas entre padrões de projetos implementados em OO e OA. Tais estudos avaliam a utilização de aspectos para implementar os padrões de projeto, e com isso modularizá-los de forma mais adequada, controlando a interação e comunicação entre eles e com os demais módulos do sistema.

Outra abordagem diferente das anteriores reforça a validade dos resultados obtidos nos estudos acima, com relação aos efeitos da modularização de interesses transversais. A mesma é baseada num levantamento específico dos tipos de falhas que os interesses transversais podem ser causar, caso não sejam modularizados de forma adequada [34]. O estudo empírico mencionado identificou através de experimentos e avaliações de código OO, uma forte relação entre a existência de interesses transversais não modularizados e a ocorrência de defeitos. Esses resultados demonstram o quão nocivos à manutenção e evolução os interesses transversais podem ser.

Assim como os estudos citados anteriormente demonstram a importância de se realizar avaliações sobre a implementação de interesses transversais, sejam eles funcionalidades básicas ou padrões de projetos, também existe outro fator fundamental que é entender como estes se comportam após o término de suas implementações, onde inevitavelmente estarão sujeitos a sofrer diversas mudanças referentes à manutenção e evolução. Por isso, novos estudos [3][36][45][64][74] se preocupam em analisar quantitativamente e qualitativamente de que forma

os interesses transversais podem afetar futuras implementações. Essas avaliações ocorrem através da observação de determinadas características relacionadas à modularidade e estabilidade dos softwares, ou seja, qual o impacto que eles sofrem na presença de mudanças que utilizam cenários reais de manutenção e evolução, e qual a capacidade que estes apresentam para se adaptarem a cada mudança específica.

A existência limitada de evidências empíricas sobre OA provoca certa desconfiança e controvérsia entre pesquisadores com relação a eficácia da mesma [99]. Estes utilizam a justificativa de que ao implementar os interesses transversais em novos módulos, os aspectos, a dependência e acoplamento existente anteriormente continua existindo neste novos módulos, e que isso impede o desenvolvimento independente do código orientado a aspectos dos demais. Pois, os desenvolvedores de cada módulo (seja um aspecto ou um modulo convencional) precisam entender outros módulos que não sejam aqueles sob responsabilidade dos mesmos. Pode-se citar como exemplo desta necessidade: (i) saber que parte do código fonte é afetada pelos aspectos, e por isso não pode ser alterada para evitar que os aspectos deixem de funcionar corretamente; e também, (ii) identificar que tipo de comportamento o aspecto vai alterar, para que não haja replicação de código.

No entanto, pesquisas [33][46][86][100] já estão sendo realizadas no sentido de solucionar esse possível problema através da criação formas mais disciplinadas para definição de dependências entre módulos convencionais e aspectos. Tal solução é semelhante ao conceito de interfaces existente em OO, e permite que ambos os módulos sejam desenvolvidos e mantidos de forma independente, desde que sejam respeitadas as regras estabelecidas para interação e comunicação entre os módulos.

3.4.2 Ausência de guias

Diferentemente de outros exemplos apresentados no Capítulo 2, pode-se observar que a área de OA ainda é carente em termos de metodologias e ferramentas que auxiliem na elaboração e execução de estudos empíricos, uma vez que estas tarefas exigem muito tempo e esforço. Apesar de não serem denominados de guias ou modelos, os estudos apresentados neste capítulo podem ser considerados como versões iniciais de tais estruturas, pois possuem características semelhantes a elas.

São raros os trabalhos específicos para esse fim como *framewoks* e *benchmarks*. O estudo mais próximo a tal iniciativa consiste no desenvolvimento de um *testbed* focado em DSOA [44],

que tem o objetivo de gerar uma série de aplicações *benchmarks* e auxiliar a configuração dos demais fatores do ambiente experimental.

Esse trabalho contém uma base necessária para fornecer suporte a elaboração, execução e replicação de estudos empíricos na área, uma vez que contém um repositório de aplicações, métricas, resultados coletados e abordagens utilizadas por diferentes pesquisadores em diferentes experimentos. Porém, tal pesquisa se encontra em estágios iniciais, tendo em visto a enorme complexidade e esforço envolvido no seu desenvolvimento.

3.5 Considerações Finais

Este capítulo apresentou a importância da avaliação empírica no contexto da manutenção de software e da OA. Os temas se relacionam uma vez que esta técnica de programação visa melhorar a manutenibilidade de softwares através do aperfeiçoamento de sua modularização.

Os estudos empíricos citados mostraram que os resultados existentes para avaliar isoladamente cada um dos temas ainda são escassos e não podem ser generalizados, assim como, estudos específicos que investigam os benefícios que a OA oferece a manutenção de software. O principal fator que atrapalha a geração desses resultados é a falta de mecanismos e metodologias específicas para apoiar os pesquisadores durante a elaboração, avaliação e replicação de estudos empíricos sobre a manutenibilidade de softwares OA.

Por isso, a elaboração e adoção de mecanismos e metodologias, tais como o Benchmarking Framework apresentado no Capítulo 4, são essenciais para garantir o avanço da área, aumentando a geração de estudos empíricos e facilitando a tomada de decisão dos profissionais da indústria. Pois, estes precisam conhecer bem as vantagens e desvantagens associadas à utilização de uma tecnologia como a OA, que impacta diretamente na atividade considerada a mais cara de todo o ciclo de desenvolvimento de software [80][82][97].

Capítulo 4

O Benchmarking Framework

Este capítulo apresenta a principal contribuição deste trabalho, um Benchmarking Framework específico para manutenção de softwares orientados a aspectos. O mesmo é destinado a auxiliar a geração de evidências empíricas, e com isso acelerar a disseminação na área. Todos os seus objetivos, características, componentes e critérios são descritos detalhadamente ao longo deste capítulo.

4.1 Apresentação

Como descrito nos capítulos anteriores, existe uma grande lacuna de evidências científicas associadas com técnicas e métodos da Engenharia de Software, principalmente quanto se trata de novas tecnologias como Orientação a Aspectos (OA) e, atividades muito caras como a manutenção de software. Por isso, este trabalho visa reduzir as dificuldades e desafios associados à geração de tais resultados, de forma que o processo de transferência e disseminação das tecnologias utilizadas seja acelerado através da utilização de um Benchmarking Framework (BF), que será apresentado e detalhado ao longo deste capítulo.

O Benchmarking Framework (BF) pode ser definido como:

> *Conjuntos de atributos capazes de identificar características específicas de diferentes aplicações (framework), de modo que seja possível avaliá-las e compará-las (benchmarking) sob um determinado contexto (manutenibilidade de software OA).*

Portanto, como indicado na definição acima, o trabalho apresentado neste capítulo é destinado a auxiliar a avaliação da manutenibilidade de técnicas de Desenvolvimento de Software Orientado a Aspectos (DSOA). Este framework define uma estrutura adequada, para que aplicações sejam avaliadas e comparadas com relação aos atributos de manutenibilidade das técnicas utilizadas.

Dessa forma, o framework é capaz de guiar pesquisadores e profissionais da área durante os processos de seleção, elaboração ou adaptação de aplicações e seus respectivos cenários de manutenção, tornando-os mais adequados para atender os objetivos específicos de determinados experimentos ou estudos de caso sobre manutenibilidade de sistemas de software OA. Portanto, o BF pode ser usado para apoiar a elaboração, replicação e avaliação de estudos empíricos que possuam propósitos relacionados ao seu.

Além disso, sua utilização sistemática na avaliação e comparação de aplicações, e cenários de mudança, permite que o BF seja utilizado como uma ferramenta para verificação de exemplos *benchmarks*. Neste contexto, o BF facilita a identificação de aplicações e cenários suficientemente representativos, sendo considerados como *benchmarks* de determinados domínios, e estando disponíveis em um repositório para que sejam usados em futuras comparações de estudos científicos.

Com isso, os principais benefícios deste BF, focalizado em permitir a avaliacao de manutenibilidade de software OA, podem ser resumidos em:

- Auxiliar a avaliação de técnicas de desenvolvimento de software;
- Apoiar a elaboração, replicação e avaliação de estudos empíricos;
- Facilitar a identificação de aplicações e cenários de mudança considerados como *benchmarks*;
- Possibilidade de extensão dos critérios para atender a estudos e aplicações de diferentes contextos;

A definição dos critérios que compõem o BF é uma tarefa muito difícil, uma vez que os mesmos precisam ser genéricos e específicos ao mesmo tempo. Pois, eles devem ser suficientemente: (i) genéricos para avaliar diferentes características de uma mesma aplicação e/ou estudo empírico, e (ii) específicos para atender aplicações e/ou estudos empíricos de diversos domínios que utilizem técnicas OA. Além disso, outro fator complicador é o rápido surgimento de novas técnicas de OA, o que impede que a lista de critérios seja elaborada estaticamente,

exigindo, portanto, que a mesma seja dinâmica, preparada para se adaptar a mudanças impostas pelas necessidades do meio externo ou dos próprios usuários.

Por isso, o BF está estruturado de forma que os seus critérios possam ser estendidos ou adaptados para: (i) atender a objetivos específicos de experimentos sobre manutenibilidade, e (ii) dar ênfase a avaliação de técnicas de OA. A seleção de grande parte das diretrizes e critérios presentes neste trabalho foi guiada através da análise de diversos estudos empíricos relacionados com manutenção de software OA, realizados por diferentes grupos de pesquisadores ao longo dos últimos anos [1][3][19][21][40][60][74][83][88].

O Framework é composto por três componentes principais: o Processo (Seção 4.2), o Produto (Seção 4.3) e os Cenários de Manutenção (Seção 4.4). Cada um deles possui propriedades específicas que têm o objetivo de avaliar pontos relevantes sobre conceitos e técnicas de manutenibilidade de OA. Exemplos concretos da aplicação do BF são apresentados ao longo deste capítulo.

4.2 O Processo

O Benchmarking Framework define um processo que consiste em diferentes categorias de *stakeholders* e uma série de transformações de entradas em saídas, como por exemplo, um conjunto de requisitos experimentais pode servir de entrada para a geração da primeira versão do plano de um experimento. As saídas podem servir também para realimentar o framework com objetivo de aperfeiçoá-lo (*feedback*). Exemplos de aplicação deste processo estão detalhados no Capítulo 5.

Devido às diferentes finalidades que o BF possui, seus *stakeholders* podem ser classificados em dois grupos principais de acordo com seus respectivos objetivos: Projetistas de Estudos Empíricos e Projetistas de Benchmarks. O primeiro grupo se destina a aqueles que estão interessados em realizar estudos empíricos sobre manutenibilidade de software envolvendo uma ou mais técnicas de OA. Neste caso, a entrada fornecida é o conjunto de requisitos do experimento, e a saída é a configuração inicial deste experimento.

O segundo grupo é composto por pesquisadores responsáveis por gerenciar *benchmarks*, adicionando ou alterando seus artefatos, além de selecionar novas aplicações e seus cenários de manutenção. Esse grupo deve analisar as entradas, que são aplicações e cenários de manutenção, com o objetivo de verificar se os mesmos são adequados e suficientemente gerais para serem

utilizados em vários estudos. As saídas geradas contêm uma ou mais aplicações e cenários de manutenção representativos, de forma que estas sejam capazes de comparar e avaliar satisfatoriamente técnicas de OA.

Além das saídas específicas geradas por cada grupo, existe outra possível saída que é comum aos dois grupos: um *Cookbook*, o qual descreve as experiências adquiridas pelos usuários com relação tanto ao uso das técnicas e métodos avaliados, como também ao uso do próprio framework. O *Cookbook* possui dois módulos, sendo o primeiro uma coletânea de Boas Práticas. Tal coletânea descreve boas e más práticas associadas ao uso de técnicas de OA, ainda sob avaliação da comunidade científica, que objetivem prover a manutenibilidade de software. E o segundo módulo, de Lições Aprendidas, em que os usuários descrevem pontos positivos e negativos com relação ao uso contínuo do framework para atender seus objetivos.

Um exemplo de *Cookbook* similar a este pode ser observado para o contexto de reestruturação de programas Java com AspectJ [22], onde algumas aplicações *benchmarks* foram utilizadas para geração do mesmo.

Além de ser importante para disseminar o conhecimento técnico dos usuários, através da descrição das experiências que cada um teve ao identificar as vantagens e desvantagens específicas das técnicas avaliadas, o *Coobook* é fundamental para ajudar no aperfeiçoamento do Framework. Por isso, é essencial que o mesmo seja a principal saída, mas não a única, utilizada para realimentação do BF.

O objetivo de retroalimentá-lo é fazer com que seu processo de transformação de entradas em saídas possa identificar limitações e pontos de melhorias nos critérios utilizados e no *framework* como um todo. Exemplos de potenciais melhorias incluem o refinamento da classificação de interesses transversais (Seção 4.3.2) ou tipos de atividades de manutenção (Seção 4.4). Tal processo de retroalimentação e’ ilustrado na Figura 4.1. A figura tambem mostra detalhadamente a representação gráfica do processo do Benchmarking Framework, contendo seus atores, componentes, entradas e saídas.

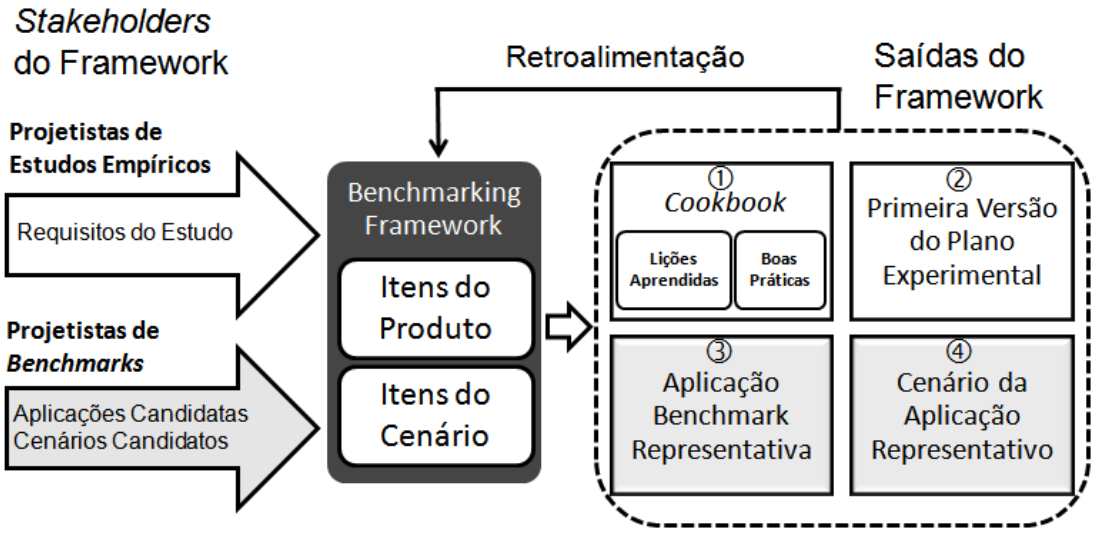

Figura 4.1. Processo definido pelo Benchmarking Framework

4.3 O Produto

Esta seção apresenta a lista de critérios responsáveis pela classificação das características relacionadas ao produto, isto é, aplicações ou sistemas de softwares. Os critérios podem ser utilizados com diferentes abordagens que variam de acordo com os objetivos específicos de cada tipo de usuário. No caso do Projetista de Benchmarks, os critérios podem ser usados para avaliar se uma determinada aplicação candidata pode ser considerada como uma aplicação benchmark. Portanto, é necessário que o usuário utilize os critérios para verificar a adequação da inclusão ou alteração de certos tipos de artefatos OA e não OA, por exemplo.

Por outro lado, o Projetista de Estudos Empíricos utiliza os critérios referentes à aplicação para auxiliar as decisões que precisam ser tomadas durante a elaboração de estudos empíricos sobre manutenibilidade de software OA. Assim, o mesmo pode se basear nos critérios do BF para iniciar a definição dos requisitos de novos estudos empíricos, os quais precisam utilizar geralmente uma aplicação específica, ou até mesmo avaliar aplicações utilizadas em estudos empíricos já existentes.

Porém, como dito na Seção 4.1, é importante salientar que é impossível dispor de um conjunto completo de critérios para todos os tipos de aplicações de domínios específicos. Por isso, essa lista de critérios deve ser constantemente evoluída e estendida para que seja possível atender a novas características e, conseqüentemente, a novos objetivos. Por reunir atributos

diversificados, a lista de critérios do produto é dividida em dois grupos: Atributos Gerais (Seção 4.3.1), que se destinam as características comuns de aplicações de qualquer tipo ou domínio, e Atributos OA (Seção 4.3.2), que contém características específicas de sistemas de software OA.

4.3.1 Atributos Gerais

Os atributos listados a seguir reúnem propriedades comuns a qualquer tipo de aplicação e são agrupados em: Identificação do Sistema, *Packaging*, Documentação do Ciclo de Vida e Técnicas de Desenvolvimento. Os exemplos apresentados em cada um dos atributos não utilizam uma mesma aplicação, de modo a aumentar a representatividade dos critérios analisados. Pois, em geral, uma única aplicação não possui todas as características analisadas nos critérios.

Identificação do Sistema

As primeiras informações de um software que precisam ser descritas são essenciais para a identificação do mesmo, assim como é descrito a seguir:

- **Nome do Sistema:** informa o nome da aplicação que está sendo analisada, e também informações relevantes sobre sua respectiva versão e fabricante. Isto facilita a busca de informações adicionais sobre as propriedades da aplicação. Exemplo:

  - *Nome do Sistema: Google Earth 4.3 (beta);*

- **Domínio do sistema:** apresenta o domínio da aplicação, que muitas vezes é um fator fundamental para a escolha de uma candidata à Aplicação Benchmark ou aplicação alvo de estudos empíricos. Além de cada tipo de domínio possuir características específicas que podem não atender a algum objetivo dos usuários. Exemplo:

  - *Domínio do sistema: Sistema Web de Software Bancário;*

Packaging

As características descritas neste atributo são relacionadas principalmente com domínio da aplicação e a configuração de seu ambiente.

- **Detalhes da Versão:** para aplicações com mais de uma versão, é importante saber detalhes específicos sobre a versão em uso e, o que a diferencia das demais. Exemplo:

  - Detalhes da Versão: Na Versão 3.2.1 foram implementadas novas funcionalidades para a entidade Clientes e Funcionários, como pesquisa e relatórios, bem como, corrigidos bugs no cadastro de Fornecedor, onde era possível adicionar um registro sem informar os campos obrigatórios;

- **Disponibilidade de Código Fonte:** o usuário deve informar de forma objetiva (SIM/NÃO) se a aplicação em questão tem o seu código fonte disponível para pesquisas ou consultas. Essa informação é decisiva para que se possa fazer uma análise completa das características do software. Exemplo:

  - Disponibilidade de Código Fonte: SIM;

- **Linguagens de Programação:** indica as linguagens de programação utilizadas para desenvolvimento do software. Cada tipo de linguagem tem características específicas que podem influenciar os objetivos de avaliação, portanto, é necessário que as mesmas sejam informadas. Exemplo:

  - Linguagens de Programação: Java, Java EE, AspectJ;

- **Configuração do Ambiente:** descreve as restrições que a aplicação possui relacionadas ao sistema e plataforma operacional e, os requisitos mínimos para que a mesma funcione adequadamente, tanto para hardwares e softwares. Exemplo:

- Configuração: PC com Windows XP ou Linux; Macintosh com Mac OS 10.4 (Tiger); Pentium 3 / 1 Ghz, 512MB de RAM; Conexão com a Internet de 256Kbps; Mozilla Firefox 2.0 ou Internet Explorer 6.0; Java 6.0; Resolução de 1024 X 768;

Documentação do Ciclo de Vida

Lista os artefatos disponíveis que foram elaborados durante todas as etapas do desenvolvimento do software. É fundamental analisar a documentação de um sistema para realizar adequadamente as atividades de manutenção de software e avaliar detalhadamente se o sistema é capaz de atender os objetivos propostos pelos usuários do BF. Por isso, em determinados casos, pode ser essencial que certos documentos estejam disponíveis para análise. Exemplo:

- *Documentação do Ciclo de Vida: Modelos de Análise e Projeto, Documento de Requisitos, Documento de Casos de Uso, Documento de Casos de Teste, Diagrama de Classes, Diagrama de Seqüência e Diagrama de Componentes;*

Técnicas de Desenvolvimento

Descreve as técnicas e abordagens utilizadas durante todas as etapas do desenvolvimento do software. As técnicas utilizadas na construção do software são diretamente relacionadas com a qualidade final do produto, pois as mesmas podem atuar sobre diferentes artefatos e em fases distintas. Por isso é necessário ter conhecimento de quais foram utilizadas para que seja possível avaliar os tipos de técnicas mais indicadas para determinados domínios, avaliando o desempenho específico de cada uma delas e, conseqüentemente, identificando a mais adequada para o contexto do framework e de seus usuários. Exemplo:

- *Técnicas de Desenvolvimento: RUP, Prototipação, Programação Orientada a Objetos, Programação Orientada a Aspectos,Testes Funcionais e Estruturais;*

A Tabela 4.1 apresenta um resumo dos Atributos Gerais apresentados nesta seção:

Tabela 4.1. Atributos Gerais do Produto.

Atributos Gerais	
Identificação do Sistema	Nome
	Domínio
Packaging	Detalhes da Versão
	Disponibilidade de Código Fonte
	Linguagens de Programação
	Sistema Operacional e Requisitos Mínimos
Documentação do Ciclo de Vida	
Técnicas de Desenvolvimento	

4.3.2 Atributos Orientados a Aspectos

Os atributos listados nesta seção são caracterizados por serem específicos e diretamente relacionados com as principais características de aplicações desenvolvidas utilizando OA. Os mesmos podem ser divididos nos seguintes principais grupos: Classificação de Interesses Transversais, Composição de Interesses, Escopo de Interesses Transversais, Construções de Linguagens OA.

Assim como na Seção 4.2.1, a seleção e adaptação dos critérios apresentados a seguir foram cuidadosamente analisadas e baseadas em trabalhos da literatura que tiveram boa aceitação dos especialistas da área. Assim, cada um dos atributos apresenta os respectivos trabalhos que guiaram a sua definição, e exemplos fictícios para que seja possível aumentar a representatividade dos critérios analisados.

Classificação de Interesses Transversais

A classificação de diferentes tipos de interesses transversais é um tema muito abordado por trabalhos existentes na literatura [4][29]. Por isso, a classificação adotada por este trabalho e,

apresentada neste atributo, é baseada em um subconjunto representativo definido por tais trabalhos que foram previamente mencionados.

Este atributo fornece uma classificação dos tipos de interesses transversais de acordo com diferentes dimensões. Para garantir que vários recursos de linguagens OA sejam exercitados, é preciso possuir uma grande diversidade de interesses transversais e analisá-la sob diferentes conceitos. Com isso, essa verificação também é necessária para muitos estudos empíricos relacionados com OA, bem como para facilitar a identificação do grau de representatividade de aplicações.

A classificação é feita de forma ortogonal, ou seja, um único interesse pode ser classificado em cada uma das categorias abaixo:

- **Funcional X Não-Funcional:** Analogamente a definição de Requisitos Funcionais e Não-Funcionais, um interesse é classificado como Funcional quando o mesmo é relacionado a funcionalidades do negócio da aplicação. Já os interesses Não-Funcionais referem-se às funcionalidades sistêmicas, que dizem respeito às restrições que o sistema deve atender e como o mesmo deve se comportar. Exemplo:

  - Interesse Funcional: Cadastro de Pessoa Física;
  - Interesse Não-Funcional: Tratamento de Exceção;

- **Homogêneo X Heterogêneo:** Um interesse transversal Homogêneo é caracterizado por alterar o comportamento de um programa em múltiplos *pontos de junção*, acrescentando o mesmo trecho de código em cada um dos *pontos de jução* estendidos. Por outro lado, um interesse transversal Heterogêneo também pode estender múltiplos *pontos de junção*, porém, cada um deles terá seu comportamento modificado por um trecho de código específico. As Figuras 4.2 e 4.3 mostram exemplos de Interesses Transversais Homogêneos e Heterogêneos, respectivamente, implementados em AspectJ.

Exemplo 1:

Na Figura 4.2, pode-se observar que o ponto de atuação `IT_Homogeneo` é responsável por estender a chamada (`call`) a todos os métodos da classe `Paciente` e de todos os seus subtipos, devido ao uso do *wildcard* `*`. Esta construção inclui os métodos que não foram herdados (uso do *wildcard* `+`). Além disso, cada um dos métodos pode ter qualquer quantidade e tipo de parâmetros (uso *wildcard* `..`). Dessa forma, todos os métodos estendidos são afetados pelo mesmo trecho de código;

```
// Interesse Transversal Homogêneo
public pointcut IT_Homogeneo():
    call (public void paciente.Paciente+.*(..));

// Aleterando o comportamento do pointcut
void around() : IT_Homogeneo() {
    ...
}
```

Figura 4.2. Exemplo de Interesse Transversal Homogêneo Implementado em AspectJ.

Exemplo 2:

Diferente do caso anterior, a Figura 4.3 mostra que o ponto de atuação `IT_Heterogeneo`, estende a chamada (`call`) de um único método, e que mesmo com a possibilidade de poder estender múltiplos pontos de junção, este caso sempre deve resultar na extensão de um único ponto do programa. Por isso, o método estendido é afetado por um trecho de código exclusivo, que não afeta nenhum outro ponto do programa;

```
// Interesse Transversal Hetereogêneo
public pointcut IT_Heterogeneo():
    call (public void paciente.Paciente.setSintomas(String));

// Aleterando o comportamento do pointcut
void around() : IT_Heterogeneo() {
...
}
```

Figura 4.3. Exemplo de Interesse Transversal Heterogêneo Implementado em AspectJ.

Devido à facilidade com que um método indevido pode ter o seu comportamento alterado acidentalmente, a implementação de Interesses Transversais Homogêneos exige atenção redobrada, pois os mesmo são muito poderosos. Porém, o uso disciplinado dos mesmos pode trazer benefícios, como por exemplo, a reutilização de código fonte e, conseqüentemente, a redução do tamanho do código final. Benefícios estes que não podem ser obtidos tão facilmente com a utilização de Interesses Transversais Heterogêneos.

- **Intra-Componente e Inter-Componente:** Este atributo é capaz de identificar, sob o ponto de vista de componentes (classes e aspectos), qual o impacto que um Interesse Transversal pode causar ao sistema. Assim, aqueles que são classificados como Intra-Componente afetam um ou mais pontos do sistema, desde que sejam em apenas um único componente. Já os Inter-Componentes afetam um ou mais pontos do sistema pertencentes a mais de um componente. Os Exemplos abaixo descrevem cada uma das classificações citadas.

  Exemplo 1:

  Suponha que um interesse transversal (IT1) qualquer tenha na sua implementação apenas o aspecto `AlterarLimiteConta`, e esse aspecto seja capaz de afetar dois métodos distintos da classe `Conta`. Com isso, pode-se classificar o interesse IT1 como Intra-Componente, uma vez que os módulos impactados pertencem ao mesmo componente, neste caso, a classe `Conta`;

  Exemplo 2:

  Suponha que um interesse transversal (IT2) qualquer tenha na sua implementação apenas o aspecto `RemoverTaxaCPMF`, e esse aspecto seja capaz de afetar apenas o método `movimentacao`, que possui diferentes implementações nas classes *Conta* e `ContaEspecial`. Dessa forma, o interesse IT2 deve ser classificado como Inter-Componente, uma vez que os módulos impactados pertencem a diferentes componentes, neste caso, a classe `Conta` *e* `ContaEspecial`;

Interação e Composição de Interesses Transversais

Este atributo classifica as diferentes formas de interação e composição existentes entre os interesses transversais de um sistema [19]. A comunicação entre os mesmos é comum e, dependendo de como for realizada, tem impacto direto na manutenção, pois o software pode ter o seu comportamento alterado pela influência que os interesses provocam entre si. Por isso, é interessante verificar tais características, uma vez que o objetivo é avaliar o impacto de técnicas de desenvolvimento sobre manutenibilidade de software OA.

Essa classificação pode ser dividida nos seguintes grupos:

- **Baseado em Invocação (*Invocation-Based*):** É a forma mais simples de composição entre interesses transversais, ocorre apenas através de chamadas de métodos entre os interesses. Para isso, nenhum dos interesses envolvidos pode possuir componentes (classes ou aspectos) em comum. Exemplo:

  

  - *O método M1 da classe C1, pertencente ao interesse IT1, possui uma chamada ao método M2 da classe C2, pertencente ao interesse IT2;*

- **Entrelaçamento no nível de Componentes (*Component-Level Interlacing*):** Identifica o entrelaçamento de interesses no nível de componentes (classes ou aspectos). Ocorre quando múltiplos interesses têm um ou mais componentes (módulos) em comum, porém, sem compartilhar nenhuma operação (atributos, métodos ou *adendos*) que implemente regras dos dois interesses, ou seja, apesar do compartilhamento de componentes, existem operações específicas para implementar as regras de cada interesse. Exemplo:

  - *O aspecto A1 possui a implementação de regras dos interesses IT1 e IT2, porém, utilizando atributos, métodos e adendos distintos para implementação de cada interesse;*

- **Entrelaçamento no nível de Operações (*Operation-Level Interlacing*):** Similar ao item anterior, este identifica o entrelaçamento de interesses no nível de operações (atributos, métodos ou adendos). O mesmo ocorre quando os interesses possuem uma

ou mais operações em comum. Portanto, essas operações têm diferentes trechos de código relacionados à implementação de cada interesse. A diferença entre este item e o anterior é o nível de abstração do entrelaçamento, que pode ocorrer tanto em relação a componentes como a operações e, conseqüentemente, provocando o entrelaçamento dos interesses e os tornando transversais ao sistema. Exemplo:

- *Os interesses IT1 e IT2 compartilham a implementação do método M1, porém, o código de M1 pode ser divido em dois trechos, onde cada um dos trechos é destinado a implementação das regras de apenas um interesse, IT1 ou IT2;*

- **Sobreposição (*Overlapping*):** Diferente dos demais estilos de combinação, neste caso é feita a identificação do compartilhamento total de comandos, operações e componentes, entre interesses. Na Sobreposição, os elementos compartilhados são totalmente relacionados à implementação da regra dos interesses, em contraste com os casos anteriores, em que apesar do compartilhamento, o código destinado a cada interesse era disjunto. Exemplo:

  - *Uma classe C1 é totalmente compartilhada pelos interesses IT1 e IT2, onde todas as suas operações destinam-se a implementação das regras, sem distinção entre os dois interesses;*

Escopo de Interesses Transversais

Este atributo é responsável por identificar em que fase do processo de desenvolvimento de software cada interesse transversal se manifesta, ou seja, quando é explicitamente referenciado em algum artefato que seja gerado durante a fase citada. Essa informação é importante, pois, estudos [6][25][84] comprovaram que dependendo das características do processo utilizado, os interesses transversais podem ser identificados em fases distintas do processo de desenvolvimento. Por exemplo, o interesse de tratamento de exceção pode ser descoberto durante atividades de requisitos, arquitetura, projeto ou implementação.

Por isso, é necessário identificar especificamente quando os interesses transversais são descobertos, permitindo avaliar as diferentes abordagens existentes na literatura e determinar,

mais precisamente, o impacto que uma descoberta tardia pode ao software e a sua manutenção. Exemplo:

- *Interesse de Auditoria foi descoberto durante a fase de Requisitos;*
- *Interesse de Tratamento de Exceção foi descoberto durante a fase de Projeto;*

Construções de Linguagens OA

Este atributo identifica os vários tipos de recursos de linguagem de programação utilizados para implementar aplicações com interesses transversais. Cada linguagem possui uma grande diversidade de elementos, com diferentes características e, por isso, influenciam diretamente a manutenção dos interesses existente em uma aplicação [45]. Para isso, é fundamental que as construções sejam listadas e diferenciadas com relação à variedade e tipos de construções, de forma que se possa avaliar a representatividade dos recursos contidos em uma determinada aplicação.

Dessa forma, devem-se identificar os tipos existentes de cada construção utilizada nos módulos OA. Além disso, tais construções podem ter uma classificação ortogonal, com relação à forma de atuação na aplicação: estática, caso o comportamento da aplicação mude antes de sua execução, ou dinâmica, se o fluxo da aplicação for alterado em tempo de execução. Os exemplos a seguir são baseados na linguagem AspectJ, mas poderiam utilizar qualquer outra linguagem OA:

- *Variedade de construções OA encontradas: Aspectos abstratos, adendos do tipo "after", "before" e "around";*
- *Tipos de atuação das construções encontradas: Estático (intertipo do tipo "declare parents") e Dinâmico ("cflow");*

A Tabela 4.2 apresenta um resumo dos Atributos Orientados a Aspectos apresentados nesta seção:

Tabela 4.2. Atributos Orientados a Aspecto do Produto.

Atributos Orientados a Aspectos	
Classificação de Interesses Transversais	Funcional e Não Funcional
	Homogêneo e Heterogêneo
	Intra-Componente e Inter-Componente
Interação e Composição de Interesses Transversais	Baseado em Invocação
	Entrelaçamento no Nível de Componentes
	Entrelaçamento no Nível de Operações
	Sobreposição
Escopo de Interesses Transversais	
Construções de Linguagens OA	

4.4 Cenários de Manutenção

Esta seção apresenta os critérios que descrevem as principais características sobre cenários de manutenção. Por se tratar de um Benchmarking Framework dedicado à avaliação da manutenibilidade de softwares que utilizam a tecnologia de OA, é essencial a existência de um módulo específico para analisar, detalhadamente, mudanças que comumente acontecem em tais aplicações. Dessa forma, é possível simular situações reais, em que um software já desenvolvido precisa sofrer certos tipos de mudanças, ou seja, manutenção, para se adequar as necessidades dos seus *stakeholders* ou do ambiente/domínio.

No contexto do BF, estes critérios podem guiar estudos e avaliações com diferentes objetivos, dependendo de como são utilizados. Por exemplo, Projetistas de Estudos Empíricos podem utilizar os critérios para identificar e mapear os impactos sofridos por uma aplicação, durante a implementação de diferentes cenários, que podem utilizar vários tipos de manutenção. Outro possível objetivo deste usuário é se basear nos critérios existentes para auxiliar a definição

de um novo estudo empírico sobre manutenibilidade, pois, os critérios ajudam a determinar quais características são essenciais para atender aos objetivos de tal estudo.

Por outro lado, os Projetistas de Benchmark estão interessados em utilizar os critérios relacionados aos cenários de manutenção, para avaliar a representatividade das características que cada Cenário Candidato possui e, dessa forma, facilitar a identificação de exemplos que podem ser considerados como Cenários Benchmarks, para determinados contextos ou domínios de aplicações. Por isso, os critérios são fundamentais para ajudar tais usuários a selecionar novos cenários ou adaptar cenários já existentes, com o objetivo de tornar o Benchmark o mais representativo possível.

Assim como descrito na Seção 4.1, os critérios deste módulo, e todos os demais módulos do Framework, devem ser constantemente evoluídos e adaptados, para que se adéqüem a novas necessidades e atendam novas características, neste caso, de manutenção de software. A seguir os critérios de cenários de manutenção são descritos e apresentados detalhadamente em cada um dos atributos.

Descrição do Cenário

Este atributo apresenta o nome do cenário, uma breve descrição sobre sua funcionalidade e outras informações relevantes sobre o mesmo. Esses dados e informações podem ser extremamente úteis durante a classificação dos demais atributos deste módulo. Exemplo:

- *Alteração no cadastro de Paciente, para que seja possível identificar o médico responsável por cada atendimento realizado no paciente;*

Tipo de Mudança

Este atributo classifica os tipos de mudança que a implementação de um cenário de manutenção causa à aplicação. Uma mudança pode ser classificada sob diferentes aspectos, objetivos e níveis de abstração, como definido por estudos anteriores [17][23][97][101]. Por isso, para melhor avaliá-la, este atributo se baseia parcialmente em tais estudos e foi divido para permitir uma classificação ortogonal do tipo de mudança com relação ao Objetivo da Mudança e a Natureza da Mudança.

- **Objetivo da Mudança:** Este item classifica a mudança de acordo com a motivação da manutenção, ou seja, pode ser relacionada desde uma simples mudança para corrigir de erros de codificação, até uma mudança mais complexa para acomodar novas funcionalidades. Esse tipo de classificação é ainda bastante controverso, e costuma dividir a opinião de pesquisadores da área, causando certa discordância sobre o assunto. Dessa forma, esta classificação é baseada nos trabalhos mais adotados e utilizados sobre este tema [47][72][80][82][97]. Portanto, o objetivo da mudança pode ser definido em três tipos de manutenção: Corretiva, Adaptativa ou de Aperfeiçoamento.

    - **Manutenção Corretiva (*Corrective*):** Este tipo de manutenção é destinado a corrigir erros que foram identificados no software. Esses erros podem variar desde simples erros de codificação até erros complexos de projeto, que podem provocar alteração de diversos componentes do software. Exemplo:

        - *Alteração no cadastro de usuário para que o mesmo não seja cadastrado indevidamente sem informar os campos obrigatórios de CPF e RG;*

    - **Manutenção Adaptativa (*Adaptive*):** São mudanças realizadas para que o software seja capaz de se adaptar a mudanças no seu ambiente externo e continuar funcionando em diferentes condições. Porém não envolve alteração radical de suas funcionalidades. Exemplo:

        - *Alteração de parâmetros para que o software se adapte a um novo gerenciador de banco de dados;*
        - *Alteração de regras do sistema para o que o mesmo se adapte a uma nova lei;*

    - **Manutenção de Aperfeiçoamento (*Perfective*):** Este tipo de manutenção envolve mudanças realizadas para estender os requisitos originais do software, adicionando novas funcionalidades ou alterando existentes com o objetivo de aperfeiçoá-lo. Este item ainda inclui mudanças classificadas por alguns autores [80][82] como Preventivas (*Preventive*). Tais mudanças alteram o software

visando preveni-lo de problemas antes que os mesmos aconteçam. Este trabalho defende uma única classificação para os dois tipos de manutenção, pois ambos possuem o mesmo objetivo, aperfeiçoar o sistema, que é o foco deste atributo[1]. Exemplo:

- *Alteração para permitir emissão de novos relatórios de venda ordenados por loja e vendedor;*

- **Natureza da Mudança:** Independente do objetivo de uma mudança, ela pode ser classificada de acordo com a sua natureza, ou seja, de que forma a mudança impacta a estrutura e comportamento do software.

  - **Estrutural:** Uma mudança que altera a estrutura do software. Tal mudança pode envolver as seguintes operações: adicionar um novo elemento (classe, método, atributo, aspecto, pontos de atuação, intertipo ou adendo), alterar um elemento existente ou remover um elemento existente. A maioria das mudanças estruturais utiliza mais de uma das operações citadas e, geralmente, também afeta o comportamento do software. Exemplo:

    - *Adicionar o método saldo da classe Conta e alterar o método debitar da classe Conta;*

  - **Comportamental:** Uma mudança que altera o comportamento do software. Mudanças semânticas podem alterar ou preservar o comportamento de uma aplicação. Em muitos casos as mudanças que alteram o comportamento não podem ser automatizadas e são realizadas manualmente. Já as mudanças que preservam o comportamento, na maioria das vezes podem ser automatizadas completamente ou parcialmente. Exemplo:

[1] Extensões do BF podem criar novos atributos para separar os dois tipos de manutenção, caso exista uma necessidade específica para isso. A Seção 4.4 dá mais detalhes sobre como estender o BF.

- *Aplicar o recfatoring Rename Method* [38] *no método debitar da classe Conta;*

Nível das Mudanças

Este atributo documenta as modificações realizadas em artefatos de diferentes fases do processo de desenvolvimento, pois, cada um deles pode sofrer um impacto específico para cada mudança de um determinado cenário de manutenção. Por isso, é necessário especificar mudanças que ocorrem nas principais fases do desenvolvimento, como por exemplo, no nível de requisitos, análise e projeto e implementação, uma vez que mudanças em tais fases implicam em alterações de artefatos associados a outras fases, freqüentemente.

- **Mudanças no Nível de Requisitos:** Detalha mudanças ocorridas em artefatos gerados na fase de Requisitos. Mudanças em requisitos podem afetar vários artefatos de fases subseqüentes do processo de desenvolvimento, portanto, é importante documentá-las e especificá-las com o objetivo de rastear melhor os impactos que as mesmas podem causar. Outro ponto relevante é que esses efeitos podem variar de acordo com a escolha da abordagem de engenharia de requisitos [98].

- **Mudanças no Nível de Análise e Projeto:** Descreve mudanças realizadas em artefatos gerados na fase de Análise e Projeto. A grande maioria destas mudanças são críticas, pois, tais artefatos fazem parte do núcleo das atividades do desenvolvimento de software, podendo comprometer a aplicação como um todo e, conseqüentemente, impactar sua modularidade e manutenibilidade.

- **Mudanças no Nível de Implementação:** Identifica mudanças ocorridas em artefatos da fase de implementação. Tais artefatos devem refletir exatamente as regras descritas nas fases de requisitos e de análise e projeto, por isso, é importante identificar e diferenciar quais módulos são afetados por mudanças.

A Tabela 4.3 apresenta um resumo dos Atributos do Cenário de Manutenção apresentados nesta seção:

Tabela 4.3. Atributos do Cenário de Manutenção.

Atributos do Cenário de Manutenção		
Descrição do Cenário		
Tipo de Mudança	Objetivo da Mudança	Manutenção Corretiva
		Manutenção Adaptativa
		Manutenção de Aperfeiçoamento
	Natureza da Mudança	Estrutural
		Comportamental
Nível das Mudanças	Requisitos	
	Análise e Projeto	
	Implementação	

4.5 Extensão do Framework

Os critérios do Framework que foram apresentados neste capítulo devem ser constantemente evoluídos, como mencionado na Seção 4.1. Um grande motivo para tal é o fato da área de OA ainda ser relativamente nova e está em fase de desenvolvimento. Por isso, é essencial que o BF acompanhe a evolução das tecnologias que ele se propõe a avaliar. Além disso, é impossível definir critérios originais para o BF, que sejam suficientemente genéricos para avaliar completamente toda e qualquer aplicação mesmo neste domínio específico, ainda atendendo a qualquer tipo de objetivo dos seus usuários. De fato, esta possibilidade de extensão permite que o BF seja instanciado para utilização em outros domínios e com outros objetivos específicos.

Esta seção apresenta exemplos de como estender os critérios do BF para atender a diferentes domínios e objetivos de usuários. A extensão deve ser realizada quando alguma característica presente na aplicação, cenário ou objetivo do experimento não está presente nos

critérios, se tornando um possível ponto de extensão. Assim, nesse tipo de situação o framework deve ser estendido para se adequar aos pontos de extensão identificados para cada caso.

Os critérios do framework podem ser estendidos tanto no componente de Produto como no de Cenário de Manutenção, como mostram, respectivamente, as Seções 4.5.1 e 4.5.2, através de exemplos práticos. Assim como o componente de Processo, caso seja necessário adaptá-lo a alguma nova restrição dos usuários. A escolha da abordagem de exemplos ao invés da criação de normas gerais para representar a extensão, foi devida a vasta gama de possibilidades que a criação de um guia apresenta e pela dificuldade em projetar qual caso tem maior probabilidade de acontecer no futuro. Assim, a fim de evitar descrever informações que possam ter pouca aplicabilidade prática, exemplos foram utilizados para retratar situações reais de extensão.

4.5.1 Extensão do Produto

A extensão dos critérios do Produto está diretamente relacionada à ausência de determinadas características de uma aplicação. Dessa forma, esta seção mostra a adequação do framework para atender a aplicações de domínio específico, como o de Linhas de Produto de Software (LPS) [27][81], que não são cobertos pelos critérios originais do framework.

A escolha do domínio de LPS é baseada na sua ligação com a OA. Vários estudos [1][2][54][73] defendem a implementação deste tipo de aplicações utilizando técnicas de OA, com objetivo de atender a variabilidade de funcionalidades e prolongar a estabilidade de uma linha de produto.

Com o objetivo de considerar características sobre o domínio de linhas de produto ao componente de Produto do Framework, é necessário adicionar um novo módulo, que pode ser chamado pelo mesmo nome do domínio, Linhas de Produto de Software, contendo atributos que permitam a classificação e avaliação de suas características. Assim como os demais critérios do framework, os critérios apresentados neste exemplo de extensão foram escolhidos com base em estudos empíricos relevantes [1][63] no contexto de LPS. Com isso, os critérios selecionados auxiliam os usuários do BF, a avaliar e decidir, se a aplicação em questão é apropriada para atender os seus objetivos.

Portanto, o novo módulo, Linhas de Produto de Software, do componente Produto contém os atributos que estão descritos abaixo e listados na Tabela 4.4:

- **Estratégias de Adoção:** Este atributo identifica a abordagem utilizada pelos *Stakeholders* para construir uma linha de produto. Pois, cada abordagem possui um nível de esforço envolvido e diferentes características. As abordagens podem ser classificadas em três formas ou estratégias de adoção. A primeira, Proativa, é identificada quando uma nova linha de produto passa a ser elaborada e desenvolvida desde o início. A segunda, Reativa, acontece quando uma linha de produto já existente é evoluída de forma incremental, para se adaptar a novos produtos ou requisitos. E por último, a abordagem Extrativa ocorre quando um produto existente dá origem a uma linha de produto, através da extração de possíveis pontos de variação e da reutilização do código em comum a partir de um produto já existente. É importante salientar que essas abordagens não são necessariamente mutuamente exclusivas, pois em alguns casos podem ser combinadas, como por exemplo, na criação de uma linha de produto a partir de um produto existente (Extrativa) e, na evolução incremental da mesma (Reativa).

- **Pontos de Variabilidade:** Identifica quais e quantos são os pontos de variabilidade encontrados na linha de produto. Esta informação é útil para analisar a complexidade da linha.

- **Mecanismo para Implementação de Variabilidades:** Este atributo descreve os tipos de mecanismos utilizados para implementar o gerenciamento de variabilidades da LPS. Cada mecanismo utilizado tem características específicas, que afetam diretamente a linha de produto com suas qualidades e defeitos, por isso, diferentes mecanismos ou combinações deles podem influenciar na qualidade, esforço e manutenibilidade da LPS. Um exemplo de mecanismos seria a utilização de OA e compilação condicional para implementar as variabilidades da linha.

A Tabela 4.4 mostra um resumo da extensão dos atributos do Produto para o domínio de LPS, apresentados nesta seção:

Tabela 4.4. Atributos do Produto para o Domínio de Linhas de Produto de Software.

Atributos para Linhas de Produto de Software
Estratégias de Adoção
Pontos de Variabilidade
Mecanismos para Implementação de Variabilidades

4.5.2 Extensão do Cenário de Manutenção

Da mesma forma que os critérios do framework podem ser estendidos para atender novas características do produto (Seção 4.5.1), o mesmo acontece para o Cenário de Manutenção. Esta seção apresenta uma possível extensão dos critérios definidos originalmente na Seção 4.3, que é o detalhamento dos impactos sofridos no nível de implementação, para cada um dos interesses existentes na aplicação.

Essa nova informação permite analisar os impactos, nas unidades de implementação (métodos, classes, aspectos e etc.), provocados por cada um dos cenários de manutenção, não só a aplicação como um todo, mas também especificamente a cada um dos interesses nela existente, sejam eles transversais ou não. Portanto, o objetivo desse detalhamento é identificar uma possível relação entre os módulos dos interesses afetados e os cenários de mudança, de forma que essa informação seja usada para observar pontos de melhoria na implementação de cada interesse e, conseqüentemente, na sua manutenção.

Para isso não é preciso criar um novo atributo, uma vez que já existe um para descrever as mudanças ocorridas nas unidades de implementação da aplicação (atributo Mudanças no Nível da Implementação). Assim, é possível adaptar o atributo existente para passar a descrever as mudanças sofridas por cada interesse, além do total de mudanças sofridas na aplicação.

A Tabela 4.5 mostra um exemplo da descrição deste novo atributo para um cenário de manutenção hipotético, onde é realizada uma correção no interesse de tratamento de exceção.

Tabela 4.5. Mudanças no Nível de Implementação Detalhas por Interesse.

Interesse	Natureza da Mudança / Unidade de Implementação		Adicionado	Alterado	Removido
Tratamento de Exceção	OA	Aspecto	1	0	0
		Ponto de Atuação	1	4	1
		Adendo	3	2	2
		Intertipo	1	0	0
		Método	0	0	0
		Atributo	1	1	1
	OO	Classe	0	4	0
		Método	0	8	0
		Atributo	0	1	0

4.6 Considerações Finais

Este capítulo apresentou detalhadamente um BF específico para avaliar a manutenibilidade de softwares OA, assim como, seus principais componentes, tais como, Processo, Produto e Cenário de Manutenção, específicos para avaliar pontos relevantes dos conceitos abordados. O mesmo define uma estrutura adequada para que aplicações, e seus respectivos cenários de manutenção, sejam avaliados e comparados com relação aos atributos de manutenibilidade das técnicas utilizadas através dos critérios existentes em cada um de seus componentes.

Dessa forma, o BF é capaz de guiar pesquisadores e profissionais da área, com diferentes objetivos empíricos, durante (i) a seleção, elaboração ou adaptação de suas aplicações, e (ii) o planejamento, avaliação ou replicação de estudos empíricos. Com isto é possível acelerar a geração de evidências empíricas na área e aumentar o suporte a tomada de decisão.

Capítulo 5

Avaliação do Benchmarking Framework

Este capítulo apresenta diferentes avaliações da utilização do Benchmarking Framework, bem como discute os resultados obtidos. Cada avaliação utilizou duas aplicações, baseadas em estudos empíricos recentes, como alvo das análises. Também são descritos pontos positivos e negativos identificados durante os procedimentos de avaliação sobre os estudos, aplicações e o próprio Benchmarking Framework.

5.1 Avaliando o Benchmarking Framework

Este capítulo é destinado a avaliar o Benchmarking Framework (Capítulo 4) e de que forma o mesmo atinge os seus objetivos propostos. Devido à diversidade de finalidades do Benchmarking Framework (BF), é necessário realizar dois diferentes tipos de avaliação, onde cada uma utiliza duas aplicações, HealthWatcher [94] e MobileMedia [105]. As avaliações precisam ser divididas em dois grupos, para que seja possível simular a utilização do framework pelos seus dois tipos de usuários (Seção 4.2), o Projetista de Estudos Empíricos e o Projetista de Benchmarks, uma vez que ambos possuem objetivos específicos. Porém, apesar das avaliações envolverem estudos em diferentes circunstâncias, o principal objetivo de ambas é analisar a eficácia do BF. As avaliações se concentram em determinar se as aplicações citadas podem ser utilizadas em estudos empíricos para padronizar a comparação e avaliação (*benchmark*) de técnicas OA.

Dessa forma, os dois grupos de avaliações se destinam a analisar:

- Se as características das aplicações alvo atendem as especificações dos critérios do Benchmarking Framework;
- Como o a utilização do Benchmarking Framework pode ser útil na identificação de falhas no planejamento ou execução de estudos empíricos;

A escolha das aplicações alvo, utilizadas nos dois tipos de avaliação, foi baseada em diversos critérios essenciais para a experimentação. Primeiramente, as aplicações precisam ser exaustivamente analisadas e observadas para que possam ser empregadas em diferentes contextos empíricos. Tanto o HealthWatcher (HW), Sistema Web OA, quanto o MobibleMedia (MM), Linha de Produto de Software OA, já foram utilizados em vários estudos empíricos [26][36][45][94][105].

Entretanto, dois grupos de pesquisa (SPG[2] e AOSE[3]) estão elaborando novos experimentos, com objetivo de avaliar a manutenibilidade de tais aplicações. Assim, o BF pode ser utilizado nesse contexto, auxiliando os Projetistas de Estudos Empíricos a elaborarem uma versão inicial dos planos dos experimentos. Este novo documento representa uma das saídas do framework (Seção 4.2), ilustrado na Figura 4.1 como ②.

E, além disso, existe uma grande necessidade de se criar aplicações *benchmark* para acelerar a geração de evidências empíricas relevantes na área, como mencionado nos Capítulo 2 e 3. Por isso, o BF pode ser usado para guiar a definição de como esses sistemas, e suas diferentes versões, podem ser aperfeiçoados para que sejam considerados como aplicações e cenários *benchmark* em seus respectivos domínios. A Figura 4.1 ilustra tais saídas do framework (Seção 4.2), respectivamente, como ③ e ④.

O segundo motivo para que essas aplicações tenham sido escolhidas é que as mesmas possuem inúmeras características diferenciadas, capazes de torná-las candidatas representativas da evolução de sistemas de software OA. Ambas as aplicações foram desenvolvidas utilizando diferentes linguagens de programação, tais como, Java, AspectJ e CaesarJ. As duas aplicações foram especialmente adaptadas com o objetivo de funcionarem como *benchmark* para comparação e avaliação da manutenibilidade de técnicas OO e OA [94][105]. Existem inúmeras versões disponíveis para o HW e MM, onde cada uma delas exercita tipos heterogêneos de mudanças.

---

[2] http://www.cin.ufpe.br/spg

[3] http://www.comp.lancs.ac.uk/computing/aod/index.htm

Terceiro, as aplicações selecionadas são de diferentes domínios e desenvolvidas originalmente por diferentes grupos de pesquisa. O HW é um Sistema de Informação Web, baseado em uma arquitetura de n-camadas [18], enquanto o MM é uma linha de produto de software, baseado em uma arquitetura MVC [39], destinado a aplicações de manipulação de dados que executam em dispositivos móveis. O primeiro, HW [49], é um sistema real, desenvolvido pelo *Software Productivity Group* (SPG) da Universidade Federal de Pernambuco. Enquanto o segundo, MM, é um protótipo acadêmico desenvolvido pela Universidade de British Columbia, no Canadá [105].

Apesar das duas aplicações compartilharem reusabilidade e manutenibilidade através do controle de requisitos não-funcionais, elas têm diferentes fundamentos de portabilidade, desempenho, privacidade, disponibilidade, e outros [95][105]. Por fim, e não menos importante, as aplicações selecionadas e suas respectivas versões foram originalmente projetadas sem qualquer tipo de acesso as informações e objetivos propostos neste trabalho.

Este capítulo está organizado da seguinte forma: a Seção 5.2 apresenta a avaliação do uso do Benchmarking Framework para auxiliar a elaboração de novos estudos empíricos, a Seção 5.3 analisa e discute os resultados obtidos na Seção 5.2, a Seção 5.4 mostra a avaliação do uso do Benchmarking Framework para identificar Aplicações Benchmark, a Seção 5.5 analisa e discute os resultados obtidos na Seção 5.4, a Seção 5.6 indica algumas limitações das avaliações e a Seção 5.7 resume os resultados do capítulo.

5.2 Avaliação do Caso 1: Elaborando novos experimentos com o BF

Esta seção apresenta a avaliação do primeiro caso de utilização do Benchmarking Framework, quando o mesmo é usado para o contexto de planejamento de experimentos. Neste caso os usuários, Projetistas de Estudos Empíricos, utilizam o Benchmarking Framework para guiar a configuração da aplicação alvo e de seus cenários, de acordo com os requisitos específicos do experimento que está sendo planejado.

Assim, o estudo apresentado nesta seção tem como objetivo avaliar como o Benchmarking Framework pode ser útil no planejamento de novos experimentos empíricos. Por isso, o mesmo irá reproduzir o planejamento de um novo experimento. As aplicações

selecionadas para participarem deste novo estudo são HW e MM, devido aos motivos já citados anteriormente e por possuírem estudos empíricos com objetivos semelhantes [45][36].

Tais trabalhos servem como base dos requisitos experimentais para este novo estudo empírico que está sendo planejado, e tem como principal objetivo analisar a estabilidade arquitetural de sistemas de softwares OA na presença de mudanças heterogêneas. Similarmente a aquelas que acontecem durante as atividades de manutenção, ou seja, dentro do mesmo contexto do Benchmarking Framework.

Portanto, a análise da eficácia que o Benchmarking Framework possui, para auxiliar o planejamento de tal experimento, é baseada na comparação de dois documentos independentes contendo versões preliminares do plano experimental, como é descrito a seguir.

Configuração do Estudo: Usuário do Framework vs. Especialista

Este estudo foi estruturado de forma que dois planos experimentais, elaborados de diferentes formas e por diferentes pessoas, fossem gerados para o mesmo experimento. O primeiro plano foi projetado com o auxílio do Benchmarking Framework por um estudante de pós-graduação, que não é especialista em análise arquitetural e possui apenas conhecimentos básicos na modelagem de estudos empíricos.

Por outro lado, o segundo plano foi feito sem nenhum tipo de ajuda do Benchmarking Framework, baseando-se exclusivamente na experiência de um especialista em análise arquitetural, que já desenvolveu uma série de estudos empíricos estudos empíricos sobre OA nos últimos cinco anos.

Essa abordagem foi utilizada com intuito de verificar sistematicamente, como a utilização do Benchmarking Framework pode ser útil no planejamento de estudos sobre manutenção de softwares OA. Dessa forma, ao comparar os planos gerados pelo usuário do Framework e pelo especialista, deve ser possível identificar benefícios e limitações no Benchmarking Framework.

Como dito anteriormente, os dois planos foram baseados nos mesmos objetivos experimentais, similares aos encontrados em estudos empíricos recentes e relevantes [45][36] que utilizam as mesmas aplicações, HW e MM, adotadas em ambos os planos elaborados.

5.3 Análise dos Resultados para o Caso 1: Elaborando novos experimentos com o BF

Os critérios do Benchmarking Framework foram observados com intuito de avaliar se eram realmente eficazes na análise, e adaptação das aplicações e de seus cenários, baseado nos objetivos específicos do experimento planejado. Assim, a avaliação do Benchmarking Framework foi direcionada aos critérios do Produto e do Cenário de Manutenção (Capítulo 4).

Tal direcionamento permitiu que após a execução do estudo e comparação dos planos gerados, fosse possível identificar características relacionadas ao Produto e aos seus Cenários, que não são cobertas pelos critérios do Benchmarking Framework. Porém, tais limitações podem ser informadas e corrigidas, conforme a indicação de retroalimentação do BF na Figura 4.1.

Os resultados do desta avaliação, tanto para o HW quanto para o MM, apontam que o uso do Benchmarking Framework pode ser vantajoso até mesmo para especialistas em avaliações empíricas sobre técnicas de OA. Como pode ser observado a seguir, o primeiro plano, gerado pelo estudante de pós-graduação, possui considerações relevantes que não foram atendidas pelo outro plano, elaborado pelo especialista.

5.3.1 Analisando Características da Aplicação (Produto)

Dados os objetivos do experimento planejado, ambos os Projetistas, tanto o estudante de pós-graduação quanto o especialista, tinham o objetivo de analisar e descrever se o HW e MM eram aplicações adequadas para avaliarem a estabilidade arquitetural de um software OA. Portanto, cada um o fez a sua maneira.

O estudante de pós-graduação considerou que instabilidades arquiteturais podem ser identificadas quando existe a presença de diferentes características relacionadas com a OA, como por exemplo, as que são listadas nos critérios do produto do BF (Seção 4.3). No caso do HW, o BF cobre satisfatoriamente estas possíveis características. Porém, no caso do MM, algumas de suas características essenciais não estão presentes no BF, assim, não podem ser identificadas facilmente. Por isso, existe a necessidade da extensão dos critérios quando é preciso se adequar a uma aplicação de domínio específico, como citado na Seção 4.5. Com a utilização de tais critérios estendidos é possível avaliar, satisfatoriamente, as instabilidades arquiteturais existentes no MM.

O especialista analisou, em seu plano, algumas características das aplicações, baseado na sua experiência em experimentação e OA. No caso do HW, ele considerou apenas duas características referentes ao produto: a diversidade de interesses transversais funcionais e não-funcionais. Uma avaliação muito limitada quando comparada aos critérios do BF referentes ao produto (Seção 4.3). Por outro lado, um ponto positivo, que merece destaque no seu plano, é a consideração de informações sobre a utilização de métricas para verificar possíveis instabilidades arquiteturais. No caso do MM, o plano do especialista continha informações mais detalhadas sobre o domínio específico da aplicação, como por exemplo, a diversidade de funcionalidades Obrigatórias, Opcionais e Alternativas.

Considerações finais: como foi descrito acima, os planos gerados por ambos os projetistas possuem pontos positivos e negativos. O resultado desta avaliação indica que o uso do BF facilita a elaboração de estudos empíricos, porém em nenhum momento, este substitui a experiência de um pesquisador. Portanto, pode-se concluir que até mesmo um especialista pode ter seu plano otimizado quando utilizar o BF, e este por sua vez, pode ter os seus critérios e componente refinados e evoluídos através do *feedback* da experiência do especialista, como descrito na Seção 4.2.

A Tabela 5.1 mostra exemplos relacionamento entre os atributos do BF e as características de cada aplicação. Essas informações são essenciais para ajudar a tomada de decisão do projetista, que precisa analisá-las para verificar quais características desejáveis ainda não estão presentes na aplicação.

Tabela 5.1. Presença dos Atributos OA nas Aplicações.

Atributos Orientados a Aspectos \ Aplicações		HW	MM
Classificação de Interesses Transversais	Funcional	-	X
	Não-Funcional	X	X
	Homogêneo	X	X
	Heterogêneo	X	X
	Intra-Componente	X	X
	Inter-Componente	X	X
Interação e Composição de Interesses Transversais	Baseado em Invocação	X	X
	Entrelaçamento no Nível de Componentes	X	X
	Entrelaçamento no Nível de Operações	X	X
	Sobreposição	X	X

5.3.2 Analisando Características dos Cenários (Cenários de Manutenção)

Assim como feito para as características do produto, os planos foram elaborados de forma a analisar também as características dos cenários de manutenção. As duas aplicações são alvo de estudos que apresentam cenários de manutenção específicos para cada uma delas [45][36]. Com isso, os projetistas analisam em seus planos o tipo de manutenção de cada cenário, com objetivo de verificar se os mesmos exercitam diferentes características da aplicação e da sua estabilidade arquitetural. Os planos foram elaborados como descrito a seguir:

O estudante de pós-graduação classificou o tipo dos cenários de manutenção de acordo com a classificação existente no BF (Seção 4.4), e identificou que os cenários são bastante similares quanto ao tipo de manutenção. Por isso foram enquadrados na mesma categoria de manutenção, Aperfeiçoamento.

O especialista usou sua própria terminologia para classificar os cenários, diferentemente do estudante de pós-graduação, que usou os termos existentes no BF. Com isso, o especialista utilizou termos referentes a tipos de manutenção, que ainda não são largamente aceitos pela classificação de mudanças de software bem conhecida na literatura [97][101]. Por outro lado, um ponto positivo, que merece destaque no seu plano, é a consideração de informações sobre a utilização de métricas para antecipar a verificação de possíveis instabilidades arquiteturais causadas pelos cenários.

Considerações finais

De fato, na prática não existe consenso na categorização de tipos de manutenção [97]. Por isso, essa imprecisão faz com que a existência e utilização de um framework sejam fundamentais para evitar equívocos, e manter a padronização dos termos utilizados durante a análise, replicação e avaliação de estudos empíricos na área de manutenção de software OA. Assim, tais problemas de ambigüidade podem ser atenuados através do uso do BF, que é capaz de guiar projetistas de diferentes experimentos a analisar as características de aplicações de forma coerente. Porém, o mesmo ainda não é capaz de atender a pontos importantes, como os apontados no plano do especialista, sobre a criação de métricas para avaliar instabilidades arquiteturais.

As Tabelas 5.2 e 5.3 (adaptadas dos estudos originais [45][36]) mostram, respectivamente, a classificação dos cenários de manutenção do HW e MM, encontradas nos planos do estudante de pós-graduação (Caso 1) e do especialista (Caso 2). É possível notar que os termos utilizados na classificação feita pelo especialista mudam de acordo com o domínio da aplicação, o que não acontece no plano do estudante de pós-graduação. Mais um fator que dificulta a avaliação e comparação, de propriedades que não consideram as características específicas de tal domínio, como por exemplo, o tipo de manutenção (Seção 4.4).

Como citado anteriormente, a classificação do BF identificou que todos os cenários possuem a mesma categoria de manutenção, o que implica na ausência de diversidade dos cenários, de ambas as aplicações, quando avaliados sob esse ponto de vista. Portanto, o BF auxilia também a identificação de possíveis pontos de melhoria durante o planejamento de experimentos, que nem sempre são facilmente identificados. Neste caso, para exercitar instabilidades arquiteturais relacionadas a todos os tipos de manutenção de software é preciso adaptar os cenários existentes ou criar novos.

Tabela 5.2. Análise do Tipo de Manutenção para os Cenários do HW [45].

Cenários	Descrição dos Cenários do HealthWatcher	Caso 1	Caso 2
R1	Decompor vários *Servelts* para melhorar a extensibilidade.	Aperfei.	Refact.
R2	Garantir que o estado do objeto *Complaint* não será atualizado depois de ser fechado, evitando múltiplas atualizações.	Aperfei.	Corret.
R3	Encapsular as operações de atualização para melhorar a manutenibilidade.	Aperfei.	Refact.
R4	Melhorar o encapsulamento do interesse de distribuição para permitir reuso e customização.	Aperfei.	Refact.
R5	Generalizar o mecanismo de persistência para permitir reuso e extensão.	Aperfei.	Refact.
R6	Remover dependências nos objetos de *Servlet Response* e *Request*, para facilitar o processo de adição de novas GUIs.	Aperfei.	Refact.
R7	Generalizar o mecanismo de distribuição para permitir reuso e extensão.	Aperfei.	Refact.
R8	Nova funcionaliade adicionada para permitir consulta de mais tipos de dados.	Aperfei.	Evolu.
R9	Modularizar o interesse de tratamento de exceção e incluir um comportamento de recuperação de erros mais eficaz nos controladores.	Aperfei.	Aperfei.

* Aperfei.: Aperfeiçoamento, Refact.: *Refactoring*, Corret.: Corretiva e Evolu.: Evolucionária

Tabela 5.3. Análise do Tipo de Manutenção para os Cenários do MM [36].

Cenários	Descrição dos Cenários do MobileMedia	Caso 1	Caso 2
R1	Versão base do MobilePhoto [28].	-	-
R2	Inclusão do tratamento de exceção.	Aperfei.	Adição de requisito não-funcional.
R3	Novas funcionalidades adicionadas para contar e ordenar fotos pelo número de vezes que a mesma foi visualizada; e para editar a descrição das fotos.	Aperfei.	Adição de funcionalidade opcional; e adição de funcionalidade obrigatória.
R4	Nova funcionalidade adicionada para permitir que os usuários definam e vejam suas fotos favoritas.	Aperfei.	Adição de funcionalidade opcional.
R5	Nova funcionalidade adicionada para permitir que os usuários façam várias cópias de uma mesma foto; e aplicação de *Refactoring* nas regras do Controlador MVC.	Aperfei.	Adição de funcionalidade opcional; *Refactoring*.
R6	Nova funcionalidade adicionada para mandar fotos para outros usuários por SMS.	Aperfei.	Adição de funcionalidade opcional.
R7	Nova funcionalidade adicionada para controlar arquivos de música; e alteração do gerenciamento de fotos para uma funcionalidade alternativa.	Aperfei.	Alteração de uma funcionalidade obrigatória em duas alternativas.
R8	Nova funcionalidade adicionada para controlar arquivos de vídeo.	Aperfei.	Adição de funcionalidade alternativa.

* Aperfei.: Aperfeiçoamento.

5.4 Avaliação do Caso 2: Identificando aplicações e cenários *benchmark*

Esta seção apresenta a avaliação do segundo caso de utilização do BF, como mencionado anteriormente nas Seções 4.2 e 5.1. Neste caso, o mesmo é utilizado por Projetistas de Benchmarks, com o objetivo de guiar a inclusão ou alteração de aplicações e cenários a um Benchmark para o contexto de manutenção de software OA. Portanto, através do uso BF, os usuários podem avaliar as aplicações e cenários em questão, com relação à representatividade das características apresentadas por ambos os objetos de análise, aplicação e seus cenários de manutenção.

Sendo assim, o estudo realizado nesta seção, tem como objetivo avaliar se o BF pode ser útil na identificação e/ou alteração de aplicações e cenários, candidatos a *benchmark* de manutenção de software OA. Dessa forma, esta avaliação simula um Projetista de Benchmark utilizando o BF. O mesmo deve analisar se e como, HW, MM e seus respectivos cenários, precisam ser alterados para serem considerados *benchmarks* para estudos de manutenção de software OA. Como citado na Seção 5.1, as aplicações foram selecionadas, entre outros motivos, por pertencerem a diferentes domínios, terem diferentes características e serem alvos de estudos empíricos recentes [45][36]. Tais estudos disponibilizaram as listas de cenários utilizadas nesta avaliação.

Esses estudos têm como objetivo analisar a estabilidade arquitetural das aplicações quando estão sujeitas a cenários reais de manutenção. A escolha dos cenários foi baseada, exclusivamente, na experiência dos projetistas dos estudos, selecionando mudanças específicas que costumam afetar (positivamente e negativamente) a estabilidade da arquitetura de tais aplicações.

No caso do estudo descrito nesta seção, a utilização do BF como uma ferramenta para facilitar a identificação de *benchmarks*, permite não só verificar se os candidatos a *benchmark* preenchem os requisitos para tal. Assim como, avaliar indiretamente os estudos empíricos que elaboraram e adotaram tais objetos analisados pelo BF. Pois, ao classificar suas características específicas através dos critérios do BF, é possível identificar pontos de melhoria nas aplicações, cenários e, conseqüentemente, nos estudos que as adotaram.

Porém, nem sempre tais pontos de melhoria significam deficiências no estudo Dependendo dos seus objetivos, essas possíveis carências podem ter sido planejadas como as outras decisões experimentais. Em tais casos é preciso investigar detalhadamente os planos experimentais para descobrir a motivação dessas decisões. Uma prova disto é que ambos os estudos empíricos analisados apresentam conclusões bastante interessantes, independentemente dos resultados desta avaliação, que são apresentados a seguir.

A seção a seguir mostra os resultados da avaliação divididos por aplicação (produto), que utiliza os critérios da Seção 4.3 e cenários (cenários de manutenção), que utilizam os critérios da Seção 4.4.

5.5 Análise dos Resultados para o Caso 2: Identificando aplicações e cenários *benchmark*

Esta seção apresenta os resultados obtidos na utilização dos critérios do BF para avaliar as aplicações e suas versões, candidatas a *benchmark*, na presença de vários tipos de mudança. Os resultados da avaliação têm como principal objetivo, investigar a adequação e representatividade das versões dos sistemas para servirem de aplicações *benchmark* em futuros estudos empíricos, de acordo com os critérios do BF (Capítulo 4). A avaliação da representatividade dos sistemas selecionados foi divida em duas partes, equivalentes aos critérios do BF:

- Análise das características dos sistemas em termos dos critérios do produto (Seção 5.5.1)
- Análise das características dos cenários de manutenção existentes de acordo com os critérios dos cenários de mudança (Seção 5.5.2).

Apesar das aplicações analisadas possuírem vários pontos e boa representatividade em algumas circunstâncias, os resultados deste estudo indicam que ambas as aplicações possuem algumas lacunas com relação à diversidade de suas características. Tais pontos poderiam ter sido identificados e evitados facilmente, caso o BF tivesse sido utilizado em seu planejamento.

Contudo, os resultados indicam como as aplicações e seus cenários podem ser adaptados, para que passem a atender a tais características que ainda não estão presentes. Esta informação auxilia na replicação de novos estudos empíricos que se baseiem nos estudos analisados por este

trabalho, e bem como, facilitando a geração de aplicações *benchmark*. Além da análise dos resultados, as seguintes seções também introduzem algumas discussões reflexivas sobre as lições aprendidas com a utilização do BF.

Outra observação importante é que apenas parte dos critérios do BF foi considerada nesta avaliação, pois nem todos estão diretamente relacionados com o objetivo principal dos estudos, que é analisar a modularidade do sistema no nível arquitetural. Esta e outras limitações desta avaliação estão descritas na Seção 5.6.

As seções a seguir descrevem as duas partes desta avaliação.

5.5.1 Avaliando o Produto

A fim de investigar a possibilidade de HW e MM serem consideradas aplicações *benchmark*, o seguinte subconjunto de critérios do BF foi utilizado para avaliar as características específicas de cada uma das aplicações: Classificação de Interesses Transversais, Classificação de Interação e Composição de Interesses, Escopo de Interesses Transversais e Construções para Linguagens OA.

Classificação de Interesses Transversais

A Tabela 5.4 mostra que o HW não abrange a todos os tipos de interesses transversais descritos nos critérios do BF. O único tipo de interesse transversal que não está presente é o de interesses transversais funcionais. Isso é um fator bastante relevante, pois, é possível concluir que as regras de negócio existentes no HW, não forçam situações de instabilidades arquiteturais, como sistemas que possuem regras que contenham interesses transversais funcionais.

Uma abordagem interessante para solucionar este problema seria a criação de um cenário de manutenção para implementar tal tipo de interesse transversal. Porém, observando a descrição dos cenários (segunda coluna) na Tabela 5.2, tal característica não está presente em nenhum dos cenários elaborados pelo estudo original [45].

Já no caso do MM, de acordo com a Tabela 5.4, a aplicação possui sem exceção, todos os tipos de interesses transversais descritos no BF, como, Funcional e Não-Funcional, Homogêneo e Heterogêneo, e Intra-Componente e Inter-Componente.

Tabela 5.4. Classificação dos Tipos de Interesses Transversais.

	Funcional	Não-Funcional	Homogêneo	Heterogêneo	Intra-Componente	Inter-Componente
HW	-	X	X	X	X	X
MM	X	X	X	X	X	X

Classificação de Interação e Composição entre Interesses

Observando a Tabela 5.5, percebe-se que HW e MM possuem boa representatividade em termos de diferentes tipos de interação e composição entre interesses, pois ambos atendem a todos os critérios definidos no BF, como Baseado em Invocação, Entrelaçamento no Nível de Componentes, Entrelaçamento no Nível de Operações e Sobreposição.

Tabela 5.5. Classificação dos Tipos de Interação e Composição de Interesses de Interesses Transversais.

	Baseado em Invocação	Entrelaçamento no Nível de Componentes	Entrelaçamento no Nível de Operações	Sobreposição
HW	X	X	X	X
MM	X	X	X	X

Escopo de Interesses Transversais

Tanto no HW quanto no MM, todos os interesses transversais se manifestam durante as atividades de projeto e implementação. Por isso, seria interessante que fossem adicionados cenários que apresentassem manifestação de interesses transversais em diferentes atividades do desenvolvimento dos softwares, como por exemplo, durante as atividades de requisitos. Dessa forma, seria possível avaliar de forma mais abrangente como a arquitetura do software seria impactada pela manifestação de interesses transversais ocorrendo em diferentes atividades.

Construções para Linguagens OA

As versões analisadas de HW e MM foram implementadas utilizando AspectJ. Portanto, a avaliação se baseou nas construções desta linguagem. Como mostra a Tabela 5.6, foram investigados os artefatos gerados pela primeira e última versão de cada aplicação, ou seja, R1 e R9 para o HW, e R2 e R8 para o MM. Neste caso, a versão considerada como inicial não é a R1, pois em tal versão só existe o código original OO. O objetivo desta avaliação é verificar a presença de diferentes construções no produto final e verificar se os cenários foram ou não responsáveis por adicioná-los. Isto auxilia a decisão se as aplicações finais e seus respectivos cenários podem ou não ser considerados *benchmarks*.

As versões finais de HW (R9) e MM (R8) apresentam uma vasta variedade de construções de linguagem OA, incluindo:

- Declaração de intertipos, como por exemplo, *declare parents*, *declare soft*, *declare precedence* e declaração de novos membros;
- Aspectos abstratos e privilegiados;
- Vários tipos de designadores de pontos de atuação, como por exemplo, chamada e execução de métodos, além de pontos de atuação abstratos;
- Diferentes tipos de adendos, como por exemplo, *before*, *around* (com e sem *proceed*), *after*, *after returning* e *after throwing*;

Porém, nem todos os tipos de construções são encontrados em ambas as aplicações, como por exemplo, apenas o HW apresenta o uso de modificador do controle de fluxo *cflow*, e já o *cflowbelow* não é encontrado em nenhuma das aplicações. Outros exemplos são as construções *within* e *withincode*, onde apenas a primeira está presente no HW e apenas a segunda no MM.

Ainda sim, mesmo com essas ausências existem vários tipos de mecanismos de programação de AspectJ presentes no HW e MM, ou seja, sob esse ponto de vistas elas são bastante representativas. No entanto, quando essas construções são avaliadas em termos de como elas afetam o código fonte, tais como, construções homogêneas e heterogêneas, é possível verificar algumas questões interessantes, como as exibidas na Tabela 5.6.

Tabela 5.6. Construções Homogêneas e Heterogêneas de Linguagens OA.

Tipos de Construções \ Aplicações			HW		MM	
			R1	R9	R2	R8
Ponto de Atuação	Homogêneo	NOO	7	15	0	0
		LOC	10	28	0	0
	Heterogêneo	NOO	5	28	19	70
		LOC	6	47	38	147
Adendo	Homogêneo	NOO	9	17	0	0
		LOC	60	134	0	0
	Heterogêneo	NOO	6	29	19	66
		LOC	51	283	129	608
Intertipo	Homogêneo	NOO	10	14	1	1
		LOC	19	28	1	1
	Heterogêneo	NOO	6	16	21	84
		LOC	18	30	21	282

NOO: Número de Ocorrências (Number of Occurrences) e LOC: Linhas de Código (Lines of Code).

Estes resultados foram coletados em termos de Número de Ocorrências (NOO) e da quantidade de Linha de Código (LOC) associada a cada uma dessas ocorrências. Com base nos dados apresentados, pode-se concluir que, sob esse aspecto, o MM não é representativo o suficiente, pois as suas versões, inicial e final, não são possuem construções homogêneas, existe apenas uma ocorrência de intertipo e nenhuma de ponto de atuação e adendo. Por outro lado, o HW possui uma boa diversidade de construções homogêneas e heterogêneas.

Os resultados, no caso do MM, indicam que o código fonte relacionado à implementação dos aspectos não está sendo reusado, uma vez que todas as construções OA (exceto uma) são heterogêneas, afetando apenas um único ponto de junção para cada ponto de atuação (Seção 4.3.2). O tamanho final do código fonte também é diretamente influenciado, isto acontece devido ao uso exclusivo de construções heterogêneas, que reduzem a vantagem do reuso e, conseqüentemente, aumentam a repetição de código fonte e dificultam a manutenção do software.

5.5.2 Avaliando os Cenários

Os cenários das aplicações HW e MM, apresentados na segunda coluna das Tabelas 5.2 e 5.3, são avaliados nesta seção com objetivo de verificar se possuem as características necessárias a um cenário *benchmark*. Esses cenários foram submetidos à avaliação de um subconjunto de critérios do BF, responsável por investigar as características específicas dos cenários candidatos a benchmark. Tal subconjunto é composto por: Tipo de Mudança e Nível de Mudança.

Tipo de Mudança

A Tabela 5.7 apresenta os resultados da classificação dos tipos de mudança presentes em cada um dos cenários de HW e MM de acordo com os critérios presentes no BF. Nesta avaliação tanto o objetivo quanto a natureza da mudança são considerados para avaliar a representatividade dos cenários implementados.

Os resultados mostram que os cenários de HW e MM são pouco representativos com relação à diversidade dos objetivos da mudança, pois, ambas as aplicações possuem um único tipo, Aperfeiçoamento, que é equivalente a apenas 33% das possíveis categorias listadas no BF. Porém, como mostra a Figura 5.1, as outras duas categorias existentes nos critérios, Corretiva e Adaptativa, representam respectivamente 17% e 18% das mudanças reais que acontecem em sistemas de software. Conseqüentemente, a categoria de Aperfeiçoamento, que está presente nas aplicações, é responsável por 65% do total, segundo estudos realizados sobre como é a real distribuição dos tipos de manutenção em diferentes softwares [66][76][97].

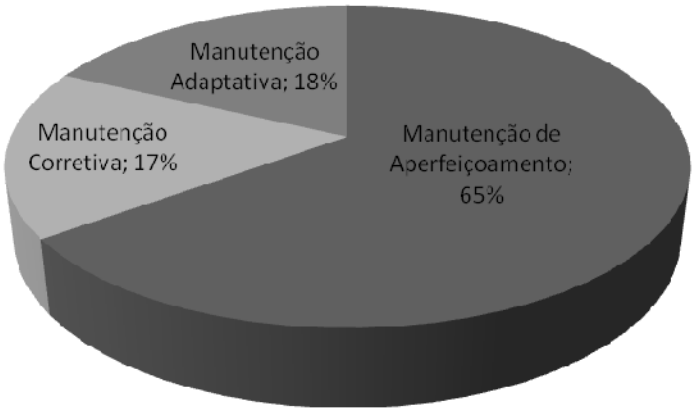

Figura 5.1. Representação Real da Distribuição dos Tipos de Manutenção [97].

Outro ponto que não é coberto satisfatoriamente é relacionado à natureza das mudanças, mais especificamente ao comportamento do sistema na presença das mudanças. Neste caso, os cenários de HW e MM apresentam características opostas, enquanto o primeiro apresenta apenas 2 cenários com alteração de comportamento (R2 e R8) e outros 7 com preservação, o segundo possui 6 cenários com alteração e apenas 1 com preservação de comportamento (R2). Dessa forma pode-se concluir que os cenários do HW são pouco representativos em mudanças que alterem o comportamento da aplicação, e por outro lado, os cenários do MM são pouco representativos em mudanças que preservem o comportamento da aplicação. Logo, o projeto de um novo estudo empírico relacionado a estes poderia considerar o uso de ambas as aplicações, já que elas possuem características distintas e complementares.

Os resultados também são negativos quando analisados em termos da representatividade dos cenários com relação ao total de possíveis combinações entre os tipos de objetivo e natureza das mudanças. Por exemplo, todos os 16 cenários analisados se concentram em apenas duas possíveis classificações: o primeiro grupo possui 8 cenários com o objetivo de manutenção de Aperfeiçoamento e mudanças que preservam o comportamento da aplicação, enquanto o segundo grupo, também com 8 cenários, possui o objetivo de manutenção de Aperfeiçoamento e as mudanças alteram o comportamento da aplicação.

Portanto, de um total de 6 possíveis categorias, os cenários analisados atendem a apenas duas delas, conseqüentemente, não atendem as outras 4 possíveis categorias (colunas 2, 3, 4 e 5 na Tabela 5.7), equivalente a 66% do total de categorias.

Tabela 5.7. Classificação dos Tipos de Mudança (Objetivo e Natureza) Para Cada Cenário de HW e MM.

Cenários	Tipo da Mudança					
	Corretiva		Adaptativa		Aperfeiçoamento	
	Preserva o Comportamento	Altera o Comportamento	Preserva o Comportamento	Altera o Comportamento	Preserva o Comportamento	Altera o Comportamento
HW						
R1					X	
R2						X
R3					X	
R4					X	
R5					X	
R6					X	
R7					X	
R8						X
R9					X	
MM						
R1	-	-	-	-	-	-
R2					X	
R3						X
R4						X
R5						X
R6						X
R7						X
R8						X

Nível da Mudança

A Tabela 5.8 mostra os resultados de todas as mudanças estruturais ocorridas nas aplicações durante a implementação dos primeiros cenários. Os valores foram coletados durante a transição de R1 para R2 no HW e R2 para R3 no MM. Dessa forma, pode-se observar exatamente o impacto estrutural causado a cada elemento da aplicação (em termos de adição, alteração ou remoção), pelo desenvolvimento de um cenário específico, neste caso, R2 para o HW e R3 para o MM.

Tabela 5.8. Mudanças no Nível de Implementação.

Unidade de Implementação \ Natureza da Mudança			Adicionado	Alterado	Removido
HW	OA	Aspecto	3	5	0
		Ponto de Atuação	6	4	0
		Adendo	5	4	0
		Intertipo	2	0	0
		Método	7	0	0
		Atributo	25	0	0
	OO	Classe	23	2	19
		Método	28	0	22
		Atributo	9	5	5
MM	OA	Aspecto	1	0	0
		Ponto de Atuação	6	4	0
		Adendo	6	3	0
		Intertipo	4	4	0
		Método	2	0	0
		Atributo	2	0	0
	OO	Classe	1	6	0
		Método	8	10	0
		Atributo	7	4	0

Os resultados apresentam, para ambas as aplicações, uma boa representatividade em termos de elementos adicionados, pois são adicionados todos os tipos presentes na classificação. Com relação aos elementos alterados, existe uma representatividade intermediária, uma vez que

alguns elementos não possuem alteração, como por exemplo, métodos e atributos OA para as duas aplicações. E por último, pode-se observar uma fraca representatividade de elementos removidos, pois, apenas o HW apresenta dois tipos de elementos removidos, que apesar de serem muitos são exclusivamente classes e métodos OO.

Esta avaliação apresentou os resultados referentes a uma única transição, mas para avaliar detalhadamente todos os cenários seria interessante avaliar a transição de todos eles, para que seja possível identificar pontos de melhoria em cada um deles.

5.6 Limitações da Avaliação

Mesmo tendo realizado diferentes avaliações (Seção 5.2 a 5.4) que confirmaram os supostos benefícios do BF através de uma investigação preliminar da sua utilidade, os procedimentos realizados tiveram algumas limitações. Tais limitações irão contribuir para futuros estudos utilizando outras aplicações candidatas a *benchmark*, e diferentes procedimentos experimentais como alvo.

Nem todos os elementos do BF foram avaliados detalhadamente, por exemplo, ainda não foi feita uma avaliação sobre como a utilização do BF pode ser efetiva para auxiliar a geração de *cookbooks* (saída ① do BF na Figura 4.1) para manutenção de sistemas de software OA [22]. Da mesma forma que nem todos os critérios do BF foram exaustivamente empregados nas avaliações apresentadas neste capítulo, sendo mais um ponto a ser focado por trabalhos futuros, assim como o refinamento de tais critérios.

Outro ponto importante são deficiências identificadas durante a avaliação de comparação dos planos de estudos empíricos (Seção 5.3). O plano do especialista continha algumas informações que não estavam presentes no BF, tais como, um conjunto de métricas para avaliar as instabilidades arquiteturais, que podem gerar um novo módulo do BF através da sua extensão. Tais aperfeiçoamentos incluem também a criação de um novo módulo, para auxiliar uma avaliação mais quantitativa da eficácia e completude que o BF possui ao atender os seus objetivos, que é uma dificuldade freqüente na área de Engenharia de Software.

Outro fator que poderia ser definido como uma limitação é o fato dos orientadores deste trabalho terem se envolvido na elaboração dos estudos que foram selecionados como alvo para participarem desta avaliação [36][45]. Porém, o autor deste trabalho não participou em nenhum momento ou foi influenciado por tais estudos durante a elaboração do BF. Assim como, grande

parte dos critérios existentes no BF foram selecionados através de trabalhos de autoria independente, sem que houvesse qualquer possibilidade influência.

5.7 Considerações Finais

Esta seção descreve resumidamente as informações contidas neste capítulo. Os resultados das avaliações do BF comprovam os seus supostos benefícios, mostrando que o mesmo pode ser útil para atender a diferentes objetivos, tipos de usuários e domínios de aplicação, dentro do contexto de manutenção de software OA. Porém, apesar dos benefícios comprovados, o mesmo ainda possui algumas limitações, como mencionado na seção anterior.

Os dois tipos de avaliações: (i) elaboração novos experimentos com o BF (Seção 5.2) e, (ii) identificação de aplicações e cenários *benchmark* (Seção 5.4) indicam que a utilização do BF é vantajosa, inclusive para especialistas na área, sejam projetistas de estudos empíricos ou projetistas de benchmarks. Uma vez que as avaliações mostraram que o BF é capaz de suprir algumas deficiências dos especialistas, porém em nenhum momento, substitui a experiência dos mesmos.

Capítulo 6

Conclusões

Este capítulo apresenta um comparativo entre o Benchmarking Framework (BF) e alguns raros trabalhos relacionados encontrados na literatura (Seção 6.1), assim como, conclusões obtidas ao término da realização deste trabalho de mestrado (Seção 6.2). Por fim, são descritos e discutidos temas sobre o BF que estão sendo pesquisados em estudos já em andamento e considerados em algumas propostas de trabalhos futuros (Seção 6.3).

6.1 Trabalhos Relacionados

São raros os trabalhos diretamente relacionados ao Benchmarking Framework (BF), pois se trata de uma pesquisa multidisciplinar dentro da área de Engenharia de Software, além de estar diretamente relacionada com estudos empíricos, manutenção software e OA, conforme discutido nos Capítulos 2, 3 e 4. Desta forma, compará-lo com outro estudo é uma tarefa complicada, uma vez que não existe tanta semelhança com qualquer outro.

Por isso, com o objetivo de relacionar o BF com outros estudos, alguns desses já apresentados em capítulos anteriores, os mesmos são agrupados por temas e comparados com o BF nas seções a seguir.

6.1.1 Ausência de Benchmarking Frameworks para DSOA

Como documentado nos capítulos anteriores, uma grande variedade de trabalhos influenciou o desenvolvimento do BF apresentado no Capítulo 4. No entanto, ainda são raros exemplos existentes na literatura, que visam criar e consolidar uma metodologia de elaboração de *benchmarks* para manutenção de softwares em geral. E, tais exemplos se restringem ainda mais quando se trata de softwares OA.

Os casos de benchmarking frameworks existentes na área de engenharia de software se limitam a fornecer uma ferramenta de apoio à tomada de decisão, exclusivamente, para criação e adaptação de processos para a área [24][31]. Estes trabalhos apresentam abordagens que tem como principal objetivo prover meios para avaliar e aperfeiçoar processos de desenvolvimento de software, enquanto, o BF definido neste trabalho, tem ênfase na avaliação das propriedades que os softwares possuem, e naquelas que eles deveriam possuir para atender os diferentes objetivos dos *stakeholders* (Seção 4.2).

Por outro lado, o trabalho apresentado por Demeyer [32] descreve um *benchmark* destinado a evolução de software, que possui objetivos muito similares aos apresentados pelo BF proposto nesta dissertação. Entretanto, ele apenas inicia uma discussão superficial sobre a necessidade da existência de tal *benchmark*, sem descrevê-lo em detalhes. Outra diferença entre os trabalhos é que ao contrário do BF, que possui foco específico de softwares OA, enquanto o outro *benchmark* é bastante genérico, destinado a softwares desenvolvidos em qualquer contexto. Essa característica dificulta a seleção de critérios relevantes, capazes de avaliar satisfatoriamente as propriedades de dois softwares desenvolvidos de formas distintas.

Diferentemente dos demais trabalhos citados nesta seção, Kienzle e Gélineau [61] propõem um conjunto de atributos responsável por avaliar a expressividade de linguagens OA baseado em aspectos que implementam as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) [43]. Os autores afirmam que esta lista de atributos pode ser naturalmente derivada analisando a composição e dependência entre tais aspectos, e com isso, usá-los como *benchmarks* para avaliar a expressividade de tais linguagens. Contudo, esse *benchmark* apresenta o problema de ter enfoque em um domínio muito específico, o de propriedades transacionais, limitando assim a sua aplicabilidade.

Já o BF, proposto neste trabalho, não considera apenas as atividades de manutenção, como também pode ser utilizado em vários domínios, e não é focado na avaliação de um único

atributo, como o de propriedades transacionais. Essa categorização de aspectos utilizada por Kienzle e Gélineau, é similar a utilizada pelo BF, e ambas são remanescentes da Categorização de Aspectos [55], que visa classificar os aspectos de acordo com os efeitos que eles causam no núcleo da aplicação. Algumas destas categorias se sobrepõem, ou seja, são redundantes, a alguns critérios definidos no BF, como por exemplo, Interação e Composição de Interesses Transversais (Seção 4.3.2). Porém, os trabalhos se diferenciam devido a Katz focar prova de correção ao invés da avaliação.

6.1.2 Aperfeiçoamento da Base de Conhecimento sobre Manutenção de Software OA

Shull e outros [89] ressaltam a importância e necessidade da disseminação e replicação de informações sobre estudos empíricos na área de engenharia de software. Ponto este, que o BF considera fundamental, contribuindo significativamente no sentido de alcançar esses objetivos. Além disso, esta dissertação deixa evidente que existe uma forte necessidade de se documentar e padronizar um conjunto de normas e diretrizes destinadas à criação de estudos empíricos sobre DSOA. Uma vez que tais estudos têm sido realizados e conduzidos, na sua maioria, de forma isolada e utilizando diferentes práticas, sem seguir nenhum padrão.

Laird e Brennam [65] apresentam uma tentativa de disseminação de informações sobre *benchmarks* para a área de engenharia de software, através de uma lista contendo os resultados obtidos por exemplos de tais *benchmarks*. Porém, as abordagens e resultados apresentados não são comentados ou aprofundados. Outras iniciativas nesse sentido, para aumentar o compartilhamento e padronização de informações, podem ser encontradas para publicação de experimentos [53] e replicação de estudos empíricos [16].

Ambos os trabalhos apresentam frameworks para guiar as respectivas atividades durante a realização de experimentos, o primeiro destina-se a padronizar a publicação e apresentação de estudos empíricos, facilitando o entendimento, avaliações e futuras replicações. Enquanto o segundo está direcionado na criação de regras para a replicação de experimentos. Esses trabalhos e outros estão contidos em um guia avançado para estudos empíricos sobre engenharia de software [90].

Portanto, o uso do BF associado a esses conceitos permite que a realização de estudos empíricos promova um aumento do intercâmbio de informações, e motive os profissionais da área de DSOA a conduzir e replicar mais estudos sobre manutenibilidade.

6.1.3 Relacionamento entre *Testbeds* e Benchmarking Frameworks para DSOA

Greenwood e outros [44] apresentam uma das iniciativas mais relacionadas ao BF proposto nesta dissertação. Tal trabalho consiste no desenvolvimento, em longo prazo, de um *testbed* específico para DSOA com o objetivo de permitir a criação de uma série de aplicações *benchmark* e de outros artefatos de apoio. Além desses itens criados, o repositório do *testbed* também possui um conjunto de métricas e de resultados coletados, com o propósito de permitir que outros pesquisadores ou desenvolvedores de softwares usem tais dados para testar suas abordagens.

De fato, a proposta do BF foi influenciada e está diretamente relacionada a esta iniciativa, pois, a mesma desempenha um papel fundamental neste contexto por: (i) identificar as características essenciais que uma aplicação precisa ter para se tornar um *benchmark*, (ii) auxiliar à tomada de decisão durante o planejamento de estudos empíricos através de um conjunto de critérios que guiam a seleção, criação e adaptação de artefatos apropriados para determinados estudos empíricos, e, (iii) promover o compartilhamento de informações que permitem e facilitam replicações de estudos empíricos.

6.2 Considerações Finais

Este trabalho discutiu a importância e carência de estudos empíricos na área de engenharia de software, mais especificamente sobre o DSOA. Assim como, propôs um BF para atender a tais necessidades com o objetivo de acelerar o processo de geração de evidências empíricas, e conseqüentemente, permitir o avanço da manutenibilidade de softwares OA. Este BF foi avaliado de diferentes formas e utilizando aplicações de propósitos distintos, como HW e MM, e seus respectivos cenários de manutenção.

Os resultados destas avaliações foram interessantes, pois, permitiram a reflexão sobre alguns pontos, tais como, o avanço da manutenibilidade de softwares OA (Seção 6.2.1), a utilização do BF em conjunto com a experiência de projetistas e pesquisadores da área (Seção 6.2.2) e, por último, a abrangência do conjunto de critérios do BF (Seção 6.2.3).

6.2.1 BF: Avanço da manutenibilidade de softwares OA

A construção deste BF representa um passo muito significante em direção a criação de uma metodologia compacta e padronizada para facilitar a elaboração de *benchmarks* sobre para softwares OA. A transferência de tecnologias promissoras geralmente leva muito tempo para ser realizada, em média 18 anos [52]. E, baseado nos resultados das avaliações, é possível concluir que o BF é uma contribuição eficaz para reduzir esse atraso, através de várias maneiras.

Em primeiro lugar, o BF mostrou que oferece soluções sistemáticas para facilitar de forma eficaz, a elaboração de estudos empíricos sobre manutenibilidade no campo do DSOA (Seção 5.2). Além disso, o mesmo é capaz de orientar a seleção, criação ou adaptação de aplicação e cenários de manutenção, representativos para serem utilizados em tais estudos ou em suas replicações (Seção 5.4). Em adição, essa iniciativa ajuda a comunidade a acelerar o aperfeiçoamento da base de conhecimento existente sobre DSOA, e apoiar melhor o julgamento daqueles que são responsáveis a tomar as decisões por parte da indústria.

Finalmente, o BF fornece meios para extensão e replicação de estudos empíricos, tarefas que sempre costumam ser demoradas e custosas [85]. Por isso, o BF pode ser empregado com o objetivo de economizar tempo e esforço associados à avaliação de estudos empíricos já existentes.

6.2.2 Interação Harmônica entre o BF e Especialistas

Apesar de propor o uso do BF, em nenhum momento foi mencionado que o mesmo substituiria, sob qualquer circunstância, a criatividade e experiência de projetistas de estudos empíricos e *benchmarks*. Muito pelo contrário, os resultados das avaliações apontam que os especialistas podem ser mais eficazes e precisos se utilizarem o BF, como por exemplo, no caso de incluir certos tipos de mudança (ex: *refactorings*) que não são explicitamente padronizados e adotados por classificações populares de tipos de mudança.

Além disso, o uso restrito do BF poderia limitar e inibir a presença da criatividade durante a elaboração de projetos experimentais, caso não seja aplicado adequadamente. Assim como, reduziria o potencial do BF afetando alguns de seus módulos, como por exemplo, a geração de *cookbooks* e o processo de retroalimentação.

Um exemplo de boa prática da utilização do BF seria usá-lo apenas depois da elaboração da primeira versão do plano experimental, com isso o BF não influenciaria as idéias inicias dos especialistas.

6.2.3 Abrangência do BF

Aplicações de diferentes domínios foram selecionadas para participar da avaliação, com o objetivo de investigar a abrangência e representatividades dos critérios definidos no BF. Entretanto, é praticamente impossível garantir a definção de uma lista de critérios capaz de englobar todas as questões relevantes para o contexto de aplicação benchmark para manutenção de software OA.

Os critérios definidos no BF são o mais genérico possível para que possam cobrir diferentes características, e ao mesmo, são específicos para avaliar adequadamente essas características. No entanto, o seu conjunto de critérios é complementar aos critérios específicos de cada interesse ou domínio de aplicação, por isso existe a possibilidade de extensão da lista.

Diferentes estudos sobre a manutenibilidade do DSOA possuem objetivos específicos que implicam em restrições nas características das aplicações utilizadas. Porém, a idéia é que o conjunto de critérios possa ajudar a acelerar e otimizar a geração da primeira versão do plano do experimental.

Por isso, é possível observar nas avaliações que algumas características relevantes para o contexto dos estudos foram facilmente esquecidas quando o BF não foi utilizado, pois, a definição das características essenciais a uma aplicação se torna uma tarefa difícil quando não se utiliza um guia. Também foi observado que existem alguns critérios no BF que são ignorados por não estarem diretamente relacionados ao objetivo do estudo.

Portanto, o BF oferece de forma coerente e padronizada a documentação de características importantes para facilitar a replicação de estudos empíricos. Além de permitir diversas extensões para atender a características ou objetivos não previstos inicialmente.

6.3 Trabalhos Futuros

Este trabalho é apenas o ponto de partida para uma série de estudos que serão baseados nele. Alguns deles já estão em andamento, como por exemplo: (i) o planejamento de estudos utilizando o BF para criação de várias aplicações *benchmark*, e (ii) elaboração de um questionário para validar os critérios do BF junto a outros especialistas da área de estudos empíricos. O primeiro trabalho visa selecionar uma grande quantidade de aplicações, como alvo para estudos de diferentes objetivos, e aplicá-las ao BF com intuito de criar ou adaptar aplicações *benchmark* que serão utilizadas em futuros estudos.

O segundo trabalho, por sua vez, é destinado a realizar um novo tipo de avaliação dos critérios do BF, onde especialistas de diferentes grupos de pesquisa irão responder um questionário sobre quais características, eles julgam ser essenciais, para categorizar uma determinada aplicação (ou um conjunto delas) dado um determinado objetivo experimental. Seja ele relacionado ao planejamento de novos experimentos ou a identificação de aplicações *benchmark*. Dessa forma é possível avaliar e comparar o resultado gerado pelos especialistas com conjunto de critérios existentes no BF. Portanto, com os resultados de tais trabalhos, acredita-se que seja possível refinar os critérios do BF e avaliar de forma mais quantitativa a eficácia dos mesmos.

Além desses trabalhos, existe a intenção de relacionar o BF com aplicação de métricas específicas, como por exemplo, as que visam quantificar o impacto de violações arquiteturais em softwares OA quando estes sofrem a ação de mudanças evolutivas [75], assim como, estender o BF para domínios diferentes da manutenibilidade de software OA.

Porém, o trabalho futuro mais ambicioso e desafiador é estender o BF para que o mesmo se torne um *Testbed* para DSOA, dando continuidade também a outro trabalho já iniciado [44]. O desenvolvimento de tal trabalho traria inúmeros benefícios em potencial, os quais são fundamentais para estimular a geração de novos estudos empíricos na área de engenharia de software, suprindo a carência existente atualmente. Entretanto este é um objetivo audacioso, que requer o envolvimento de várias pessoas durante um longo período tempo nas pesquisas.

Bibliografia

[1] Alves, V. Implementing Software Product Line Adoption Strategies, Ph.D. thesis. UFPE, March 2007.

[2] Alves, V. et al. Extracting and evolving code in product lines with aspect-oriented programming. Trans. on AOSD: Special Issue on Software Evolution, 2007.

[3] Apel, S. et al. Aspectual Mixin Layers: Aspects and Features in Concert. Proc. ICSE'06, Shanghai, China, May 2006.

[4] Apel, S., et al. On the Structure of Crosscutting Concerns: Using Aspects or Collaborations?. Aspect-Oriented Product Line Engineering Workshop, GPCE.06, October 2006.

[5] Apel, S., Kastner, C., and Trujillo, S. 2007. On the Necessity of Empirical Studies in the Assessment of Modularization Mechanisms for Crosscutting Concerns. In Proceedings of the International Conference on Software Engineering Workshops (ICSEW), page 161. IEEE Computer Society Press, May 2007.

[6] Batista, T., et al: Reflections on architectural connection: seven issues on aspects and adls. International workshop on Early aspects at ICSE, New York, NY, USA, ACM (2006) 3-10.

[7] Banker, R.D., Datar, S.M., Kemerer, C.F. Factors affecting software maintenance productivity: An exploratory study. In Proceedings of the 8th International Conference on Information Systems (ICIS), Pittsburgh, PA (1987) pp. 160–175.

[8] Banker, R. D., Datar, S. M., Kemerer, C. F., Zweig, D. Software Errors and Software Maintenance Management. Inf. Technol. and Management 3, 1-2 (Jan. 2002), 25-41.

[9] Barry, E., Slaughter, S., Kemerer, C. F. 1999. An empirical analysis of software evolution profiles and outcomes. In Proceedings of the 20th international Conference on information Systems (Charlotte, North Carolina, United States, December 12 - 15, 1999).

International Conference on Information Systems. Association for Information Systems, Atlanta, GA, 453-458.

[10] Basili, V. R., Selby, R. W., and Hutchens, D. H. 1986. Experimentation in software engineering. IEEE Trans. Softw. Eng. 12, 7 (Jul. 1986), 733-743.

[11] Basili, V. R. 1993. The Experimental Paradigm in Software Engineering. In Proceedings of the international Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions (September 14 - 18, 1992). H. D. Rombach, V. R. Basili, and R. W. Selby, Eds. Lecture Notes In Computer Science, vol. 706. Springer-Verlag, London, 3-12.

[12] Basili, V. R. 1996. The role of experimentation in software engineering: past, current, and future. In Proceedings of the 18th international Conference on Software Engineering (Berlin, Germany, March 25 - 29, 1996). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 442-449.

[13] Basili, V. R., Shull, F., and Lanubile, F. 1999. Building Knowledge through Families of Experiments. IEEE Trans. Softw. Eng. 25, 4 (Jul. 1999), 456-473.

[14] Boehm, B., Bhuta, J., Garlan, D., Gradman, E., Huang, L., Lam, A., Madachy, R., Medvidovic, N., Meyer, K., Meyers, S., Perez, G., Reinholtz, K., Roshandel, R., and Rouquette, N. 2004. Using Empirical Testbeds to Accelerate Technology Maturity and Transition: The SCRover Experience. In Proceedings of the 2004 international Symposium on Empirical Software Engineering (August 19 - 20, 2004). International Symposium on Empirical Software Engineering. IEEE Computer Society, Washington, DC, 117-126.

[15] Booch, G. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, second edition, 1994.

[16] Brooks, A., Roper, M., Wood, M., Daly, J., Miller, J. Replication's role in software engineering. In F. Shull, J. Singer, and D. I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 365--379. Springer, 2008.

[17] Buckley, J., et al. Towards a Taxonomy of Software Change. Journal on Software Maintenance and Evolution: Research and Practice, pp. 309-332, 2005.

[18] Buschmann, F., et al. Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[19] Cacho, N. et al. Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming. AOSD.06., March 2006.

[20] Castor Filho, F., Rubira, C., Garcia, A. A Quantitative Study on the Aspectization of Exception Handling. Proc. ECOOP Workshop on Exception Handling in OO Systems, Glasgow, Scotland, July 2005.

[21] Castor Filho, F., et al. Exceptions and Aspects: The Devil is in the Details. FSE.06, pp. 152-162. ACM Press, November 2006.

[22] Castor Filho, F., Garcia, A., Rubira, C. Extracting Error Handling to Aspects: A Cookbook. Proc. ICSM.07, Paris, France, October 2007.

[23] Chapin, N., et al. Types of software evolution and software maintenance. Journal on Software Maintenance and Evolution: Research and Practice, 13:3–30, 2001.

[24] Chiew, V., et al. Software Engineering Process Benchmarking. PROFES.02, pp. 519-531. LNCS, v. 2559, 2002.

[25] Chitchyan, R., et AL. Survey of Aspect-Oriented Analysis and Design. AOSD-Europe Project Deliverable No: AOSD-Europe-ULANC-9. 2006.

[26] Chitchyan, R., et al. Early Aspects at ICSE 2007. In Companion, pp.127-128. ACM Press, May 2007.

[27] Clements, P., Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[28] Colyer, A., et al.: Large-scale AOSD for Middleware. AOSD '04. New York, NY, USA, ACM. 2004. 56-65.

[29] Constantinides C., et al. Reasoning About a Classification of Crosscutting Concerns in Object-Oriented Systems. Workshop on AOSD. Bonn, February 2002.

[30] Dabberdt, W., Schlatter, T., Carr, F. Multifunctional Mesoscale Observing Networks. Bulletin of the American Meteorological Society, 86(7):961-982, July 2005.

[31] Daoudi, F., et al. A Benchmarking Framework for Methods to Design Flexible Business Processes. Software Process: Improvement and Practice, 12(1):51-63, 2006.

[32] Demeyer, S., Mens, T.,Wermelinger, M. 2001. Towards a Software Evolution Benchmark. In Proceedings of the 4th international Workshop on Principles of Software Evolution (Vienna, Austria, September 10 - 11, 2001). IWPSE '01. ACM, New York, NY, 174-177.

[33] Dósea, M., Costa Neto, A., Borba, P., Soares, S. Specifying design rules in aspect-oriented systems. In I Latin American Workshop on Aspect-Oriented Software Development - LA-WASP'2007, affiliated with SBES'07, pages 67-78, João Pessoa-PB, Brazil, October 2007.

[34] Eaddy, M., et al. Do Crosscutting Concerns Cause Defects? IEEE Transaction on Software Engineering, 2008.

[35] Elrad, T., Filman, R., Bader, A. Aspect-oriented programming: Introduction, Communications of the ACM, v.44 n.10, p.29-32, Oct. 2001.

[36] Figueiredo, E., et al. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. ICSE.08. May 2008.

[37] Filman, R., et al. Aspect-Oriented Software Development. Addison-Wesley, Boston, 2005.

[38] Fowler, M. et al. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

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

[40] Garcia, A., et al. Modularizing Design Patterns with Aspects: A Quantitative Study. Proc. AOSD.05, Chicago, March 2005.

[41] Gold, N., Mohan, A., Knight, C., Munro, M. Understanding service-oriented software. Software, IEEE,Vol. 21, No. 2, 2004, 71-77.

[42] Goldschmidt, T., Reussner, R., Winzen, J. 2008. A case study evaluation of maintainability and performance of persistency techniques. In Proceedings of the 30th international Conference on Software Engineering (Leipzig, Germany, May 10 - 18, 2008). ICSE'08. ACM, New York, NY, 401-410.

[43] Gray, J., and Reuter, A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[44] Greenwood, P., Garcia, A., Rashid, A., Figueiredo, E., Sant'Anna, C., Cacho, N., Sampaio, A., Soares, S., Borba, P., Dosea, M., Ramos, R., Kulesza, U., Bartolomei, T., Pinto, M., Fuentes, L., Gamez, N., Moreira, A., Araujo, J., Batista, T., Medeiros, A., Dantas, F., Fernandes, L., Wloka, J., Chavez, C., France, R., and Brito, I. 2007. On the Contributions of an End-to-End AOSD Testbed. In Proceedings of the Early Aspects At Icse: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 8.

[45] Greenwood, P., et al. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. Proc. 21st ECOOP.07, July 2007, Berlin, Germany.

[46] Griswold, W., Sullivan, K., Song, Y., Shonle, M., Tewari, N., Cai, Y., Rajan, H. Modular Software Design with Crosscutting Interfaces. IEEE Software, Special Issue on Aspect-Oriented Programming, January/February 2006.

[47] Grubb, P., Takang, A. Software Maintenance: Concepts and Practice. World Scientific Publishing Co., Singapore, 2nd edition, 2005.

[48] Hannemann,J., Kiczales,G. Design Pattern Implementation in Java and AspectJ. Proc. OOPSLA'02, Nov 2002.

[49] Health Watcher. Implementação do HealthWatcher. Disponível em: http://www.comp.lancs.ac.uk/~greenwop/tao/implementation.htm

[50] L. Hochstein, T. Nakamura, F. Shull, N. Zazworka, M. Voelp, M.V. Zelkowitz, and V. Basili. An Environment of Conducting Families of Software Engineering Experiments. University of Maryland, CS-TR-4851 and UMIACS-TR-2007-28, May 2007.

[51] IEEE, "Standard Glossary of Software Engineering Terminology", IEEE Std 610.12-1990, 1990.

[52] Jedlitschka A., et al. Relevant Information Sources for Successful Technology Transfer: A Survey Using Inspections as an Example. ESEM 2007, pp. 31-40. IEEE, September 2007.

[53] Jedlitschka, A.; Ciolkowski, M.; Pfahl, D.: Reporting Controlled Experiments in Software Engineering. In Shull, F., Singer, J., Sjøberg, D. (Editors). Guide to Advanced Empirical Software Engineering; Springer 2008.

[54] Kästner, C., Apel, S. and Batory, D. A Case Study Implementing Features using AspectJ. In Proc. of Intl. Software Product Line Conference (SPLC'07), Japan, 2007.

[55] Katz, S. Aspect categories and classes of temporal properties. Transactions on Aspect-Oriented Software Development, 3880, 2006.

[56] Kemerer, C. F. Software Complexity and Software Maintenance: A Survey of Empirical Research. Annals of Software Engineering (1), September 1995, pp. 1-22.

[57] Kemerer, C. F., Slaughter, S. 1999. An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering. 25, 4 (Jul. 1999), 493-509.

[58] Kiczales, G., Lamping, J., Mendhekar, M., Maeda, C., Lopes, C., Loingtier, JM and Irwin, J. Aspect-Oriented Programming. In proc. European Conference on Object-Oriented Programming (ECOOP'97). Springer-Verlag LNCS n.1241. 1997.

[59] Kiczales, G., et al.: Getting Started with AspectJ. Communications of the ACM 44(10), 59–65. (2001).

[60] Kienzle, J., et al. AOP Does It Make Sense? The Case of Concurrency and Failure. ECOOP.02, pp. 37–61. Springer–Verlag LNCS 2374, June 2002.

[61] Kienzle, J., et al. AO challenge - Implementing the ACID Properties for Transactional Objects. AOSD.06, March 2006.

[62] Koponen, T., Hotti, V. 2005. Open source software maintenance process framework. In Proceedings of the Fifth Workshop on Open Source Software Engineering (St. Louis, Missouri, May 17 - 17, 2005). 5-WOSSE. ACM, New York, NY, 1-5.

[63] Krueger, Charles. Easing the transition to software mass customization. In Proceedings of the 4th International Workshop on Software Product-Family Engineering., pages 282–293, Bilbao, Spain, October 2001.

[64] Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., von Staa, A., and Lucena, C. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. In Proceedings of the 22nd IEEE international Conference on Software Maintenance. ICSM. September 2006.

[65] Laird, L., Brennan, M. Software Measurement and Estimation: a Practical Approach (Quantitative Software Engineering Series). Wiley-IEEE Computer Society Pr. 2006.

[66] Lientz, B. P., Swanson, E. B. Software Maintenance Management. Addison-Wesley Longman Publishing Co., Inc. 1980.

[67] Lindvall M., I. Rus, F. Shull, M. V. Zelkowitz, P. Donzelli, A. Memon, V. R. Basili, P. Costa, R. T. Tvedt, L. Hochstein, S. Asgari, C. Ackermann, and D. Pech, An Evolutionary Testbed for Software Technology Evaluation, NASA Journal of Innovations in Systems and Software Engineering 1(1) (2005) 3-11.

[68] Lopes, C., et al. An Analysis of Modularity in Aspect-Oriented Design. AOSD.05, pp. 15-26. ACM Press, March 2005.

[69] Martin, J., McClure, C. L. 1983 Software Maintenance: the Problems and its Solutions. Prentice Hall Professional Technical Reference.

[70] Kajko-Mattsson, M., Lewis, G. A., Smith, D. B. 2007. A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems. In Proceedings of the 29th international Conference on Software Engineering Workshops (May 20 - 26, 2007). ICSEW. IEEE Computer Society, Washington, DC, 117.

[71] Mezini, M., Ostermann, K. Conquering Aspects with Caesar. Proc. AOSD'03, pp. 90-99, Boston, USA, 2003.

[72] Mens, T., Demeyer, S. Software Evolution. 1st edition, Springer Publishing Company, 2008.

[73] Mezini, M., Ostermann, K. Variability Management with Feature-Oriented Programming and Aspects. In Proceedings of FSE'2004, pp.127-136, 2004.

[74] Molesini, A., et al. On the Quantitative Analysis of Architecture Stability in Aspectual Decompositions. WICSA.08, February 2008, Vancouver, Canada.

[75] Monteiro, M., Moura, M., Soares, S., Castor Filho, F. Towards an Analysis of Layering Violations in Aspect-Oriented Software Architectures. ADI 2008 - Workshop on Aspects, Dependencies, and Interactions at 22nd European Conference on Object-Oriented Programming, ECOOP 2008. Paphos, Cyprus, July 2008.

[76] Nosek, J. T., Palvia, P. Software maintenance management: changes in the last decade. Journal of Software Maintenance 2, 3 (Sep. 1990), 157-174.

[77] Ossher, H., Kaplan, M., Katz, A., Harrison, W., Kruskal, V. Specifying suject–oriented composition. TAPOS, 2(3):179–202, 1996. Special Issue on Subjectivity in OO Systems.

[78] Ossher, H., Tarr, P. Using subject–oriented programming to overcome common problems in object–oriented software development/evolution. In International Conference on Software Engineering, ICSE'99, pages 698–688. ACM, 1999.

[79] Open Source Initiative: The Open Source Definition. Open Source Initiative 2008. http://www.opensource.org/docs/osd

[80] Pfleeger, S. Software Engineering: Theory and Practice. Prentice Hall. 2001.

[81] Pohl, K., Böckle, G., and Linden, F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc, 2005.

[82] Pressman, R. S. Software Engineering: a Practitioner's Approach. McGraw-Hill Science/Engineering/Math. 2004.

[83] Rashid, A., et al. Persistence as an Aspect. AOSD.03. ACM Press, 2003.

[84] Rashid, A., at al.: Modularisation and composition of aspectual requirements. AOSD. 2003.

[85] Redwine, S. T., Riddle, W. E. 1985. Software technology maturation. In Proceedings of the 8th international Conference on Software Engineering (London, England, August 28 - 30, 1985). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 189-200.

[86] Ribeiro, M., Dósea, M., Bonifácio, R., Costa, N, C., Borba, P., Soares, S. Analyzing class and crosscutting modularity with design structure matrixes. In XXI Brazilian Symposium

on Software Engineering - SBES'07, pages 167-181, João Pessoa-PB, Brazil, October 2007.

[87] Robson, C. Real World Research: A Resource for Social Scientists and Practitioners-Researchers, Blackwell, 1993.

[88] Sampaio, A., et al. A Comparative Study of Aspect-Oriented Requirements Engineering Approaches. ESEM'07, LOCAL, 2007.

[89] Shull, F., et al. Knowledge-Sharing Issues in Experimental Software Engineering. Empirical Softw. Engg. 9, 1-2 (Mar. 2004), 111-137.

[90] Shull, F., Singer, J., Sjøberg, D. Guide to Advanced Empirical Software Engineering. Springer, 2008.

[91] Sim, S. E., Holt, R. C., Easterbrook, S. 2002. On Using a Benchmark to Evaluate C++ Extractors. In Proceedings of the 10th international Workshop on Program Comprehension (June 27 - 29, 2002). IWPC. IEEE Computer Society, Washington, DC, 114.

[92] Sim, S. E., Easterbrook, S., Holt, R. C. 2003. Using benchmarking to advance research: a challenge to software engineering. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 74-83.

[93] Sjøberg, D., Welland, R., Atkinson, M., Jørgensen, M., Martinussen, J. P., Maus, A. Evaluating Software Maintenance Technology. In Proceedings of the 6th International Workshop on Persistent Object Systems. 1996-11 : 49-61.

[94] Soares, S., et al. Implementing Distribution and Persistence Aspects with AspectJ. OOPSLA.02, pp. 174–190, November 2002.

[95] Soares, S. An Aspect-Oriented Implementation Method. Ph.D thesis, UFPE 2004.

[96] S. Soares, P. Borba, E. Laureano. Distribution and Persistence as Aspects. Software: Practice & Experience, 2006.

[97] Sommerville, I. Software Engineering. 7th Edition, Addison-Wesley, 2006.

[98] Sommerville, I., et al. Requirements Engineering: Processes and Techniques. John-Wiley & Sons, 1998.

[99] Steimann, F. The paradoxical success of aspect-oriented programming. In OOPSLA 2006: Proceedings of the 21st International Conference on Object oriented Programming Systems, Languages, and Applications, ACM SIGPLAN Notices, pages 481–497, New York, NY, Oct. 2006. ACM.

[100] Sullivan, K. et al. Information Hiding Interfaces for Aspect-Oriented Design. Proc. FSE-13, Sept 2005, Lisbon, Portugal.

[101] Swanson, E. The Dimensions of Software Maintenance. ICSE.76, pp. 492-297. IEEE Computer Society Press, October 1976.

[102] Tichy, W. F., Lukowicz, P., Prechelt, L., and Heinz, E. A. 1995. Experimental evaluation in computer science: a quantitative study. J. Syst. Softw. 28, 1 (Jan. 1995), 9-18.

[103] Tichy, W. Should Computer Scientists Experiment More? IEEE Computer, 31(5):32–40, May 1998.

[104] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. 2000 Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers.

[105] Young, T. Using AspectJ to Build a Software Product Line for Mobile Devices. MSc Dissertation. University of British Columbia, August 2005.

[106] Zelkowitz, M.V., Wallace, D.R. Experimental models for validating technology. IEEE Computer, 31(5): 23-31, May 1998.