5
Conclusão

Da mesma forma que o conceito de transparência emergiu e se tornou importante em outros contextos, acreditamos que, com o tempo, o mesmo irá acontecer com o software. Nem todo software será impactado pela transparência, mas aqueles relacionados a áreas onde a demanda por transparência é crescente, serão mais suscetíveis a mudanças devido a este novo requisito.

A complexidade da transparência foi abordada em (Cappelli, 2009), onde o conceito foi estudado e entendido como uma rede de qualidades que já são discutidas há bastante tempo. De acordo com esta modelagem, contribuindo individualmente para uma destas qualidades, estamos contribuindo também para a transparência.

Desenvolver software sob os princípios de transparência, não é benéfico apenas do ponto de vista da divulgação de informações; tem impacto direto também em outras questões, há muito discutidas no âmbito da engenharia de software (ES), como entendimento do software, rastreabilidade entre seus artefatos e critérios para modularização.

Neste trabalho, guiado por estes princípios e tendo como base a ideia de que é importante manter os requisitos próximos ao código, propomos um processo para desenvolvimento de software mais transparente (PDS+T). Este processo foi criado de forma que fosse possível abordar problemas já conhecidos da ES, e que também contribuem para transparência.

De maneira geral, o processo permite a definição da estrutura do software a partir dos requisitos, mantendo a rastreabilidade entre os artefatos produzidos. A partir desta estrutura, chega-se ao código fonte, de forma que este pode ser rastreado até a estrutura e dai até os requisitos.

5.1.
Comparações com Trabalhos Relacionados

Em seu trabalho, (Bucchiarone et al., 2008) Bucchiarone descreve uma abordagem para conduzir o desenvolvimento de software a partir dos requisitos,

até a produção de código Java. O processo apresentado permite a checagem, através de modelos, entre a arquitetura do software e os requisitos, garantido sua conformidade. O processo também propõe a geração automática do código a partir da arquitetura do software.

A abordagem se inicia com a especificação dos requisitos funcionais do software, que é realizada através da linguagem PSC (Property Sequence Chart). Esta linguagem é uma extensão dos diagramas de sequência presentes na UML 2.0 (Pilone, 2005), que permite especificação gráfica de propriedades temporais. Como a tarefa de traduzir os requisitos para um conjunto de propriedades demonstrou ser muito difícil, a ferramenta W_PSC foi utilizada para este fim. Esta ferramenta guia as decisões, eliminando as incertezas e ambiguidades dos requisitos, o que sua especificação através da linguagem PSC.

A segunda etapa da abordagem compreende a especificação da arquitetura do software. Isto é feito através de uma notação baseada em UML, que é parte do framework CHARMY. Este framework foi especialmente criado para ajudar no desenho da arquitetura do software e permitir sua validação automática com os requisitos. No CHARMY, a descrição da arquitetura é feita em termos de componentes e das mensagens trocadas entre eles. Além disto, cada componente tem seu comportamento interno descrito através de máquinas de estado.

A validação entre a arquitetura do software e os requisitos do software é realizada na terceira etapa da abordagem. As notações utilizadas no CHARMY, para descrever a arquitetura, são automaticamente traduzidas para um protótipo formal. Este protótipo é executado e verificado pela ferramenta SPIN. O resultado da execução do protótipo é utilizado para validá-lo de acordo com os requisitos modelados em PSC.

Finalmente, a última etapa da abordagem consiste na geração do código Java, a partir da especificação da arquitetura validada, que é feita em dois passos. No primeiro, o código ArchJava (Aldrich et al., 2000) é obtido automaticamente a partir da especificação da arquitetura. Por fim, o compilador ArchJava, que foi desenvolvido em (Aldrich et al., 2000), é utilizado para gerar código Java executável a partir da especificação obtida no primeiro passo.

Embora este processo permita a validação da arquitetura de acordo com os requisitos, nada é dito sobre o critério para sua construção a partir dos requisitos. Consequentemente, não há nenhum rastro entre arquitetura e requisitos, o que significa que modificações nos requisitos não podem ser facilmente propagadas para a arquitetura. Além disto, não há preocupação com

os requisitos não funcionais e seu impacto na arquitetura e, portanto, também não há no código, já que este é gerado automaticamente. Ademais, a linguagem para descrição de requisitos utilizada (PSC) é formal. O próprio autor afirma que seu uso é difícil, de forma que até uma ferramenta foi proposta para facilitar a modelagem através dela.

O uso de notações similares ao diagrama de sequência da UML, propostas no trabalho, não é intuitiva para indivíduos que não são especialistas. Isto representa um obstáculo na comunicação com os *stakeholders* para validação dos requisitos. Por fim, a geração automática de código cria rastro entre arquitetura e código em apenas uma direção. Desta forma, mudanças na arquitetura podem ser facilmente propagadas para o código (basta gerar novamente), porém o inverso não é verdadeiro, isto é, uma mudança no código não pode ser propagada automaticamente para a arquitetura.

O processo PDS+T descreve heurísticas que permitem definir a arquitetura do software a partir dos requisitos. O mesmo artefato, os cenários, em diferentes níveis de abstração, são utilizados para descrever tanto a arquitetura quanto os requisitos. Isto é feito de forma que é possível a rastreabilidade em ambas as direções entre requisitos e arquitetura, o que permite a propagação de mudanças. Adicionalmente, o modelo de cenários utilizado permite a descrição de requisitos não funcionais, o que significa que o seu impacto pode ser rastreado dos requisitos para a arquitetura e desta para o código. Como dito anteriormente, no modelo de cenários utilizado, os requisitos são descritos em linguagem natural, uma característica que facilita a comunicação com não especialistas. Finalmente, no PDS+T os cenários são refinados até que possam ser operacionalizados. Os cenários operacionais permanecem junto ao código como forma de anotação, permitindo rastreabilidade em ambas as direções entre código e arquitetura. Consequentemente, as mudanças na arquitetura podem ser propagadas para o código e o inverso também é possível.

Entretanto, o trabalho de Bucchiarione permite a validação, através de modelos, da arquitetura a partir dos requisitos. Embora no PDS+T, por construção, todos os cenários são detalhados para a arquitetura e, posteriormente, para o código, não é possível realizar uma validação mais formal. Além disto, no PDS+T a atividade de geração do código e totalmente manual, enquanto no trabalho de Bucchiarione ela é automatizada.

O trabalho de Al-Otaiby (Al-Otaiby et al., 2005), propõe uma abordagem para obtenção da organização modular dos requisitos, a partir de cenários, que

são descritos em linguagem natural. Sua abordagem é dividida em três fases: (i) análise cenário-atributo, análise de cenários e clusterização de cenários. Na primeira fase, uma análise linguística é realizada. Esta análise é baseada em algoritmos de mineração de dados, e tem como objetivo identificar objetos específicos em cada cenário. Estes objetos são chamados de atributos do cenário, e são utilizados para estabelecer os relacionamentos entre os cenários de requisitos. A segunda fase consiste na definição da força dos relacionamentos identificados anteriormente. Esta atividade também é pautada nos atributos do cenário, identificados na fase anterior. Finalmente, uma técnica de clusterização hierárquica é utilizada, com o objetivo de organizar os cenários em grupos que, por sua vez, serão utilizados para definir os módulos do sistema.

Diferente do trabalho de Al-Otaiby, nossa proposta de organização modular utiliza o LAL, que mapeia o vocabulário do UdI, fornecendo a base para encontrar os relacionamentos entre seus símbolos, já que segue o princípio da circularidade. Assim, heurísticas para modularização são aplicadas sobre um documento (LAL), o qual tem, por definição, relacionamentos estruturados. Baseado nisto, é justo dizer que heurísticas baseadas no LAL devem ser mais efetivas, pois sua principal característica é o relacionamento explícito entre seus símbolos. A atividade de inferir relacionamentos a partir de documentos não estruturados é mais suscetível a falhas, pois eles não foram criados com este propósito. Portanto, podemos afirmar que conclusões obtidas a partir da análise destes relacionamentos são menos efetivas.

Em geral, abordagens baseadas em desenvolvimento dirigido por modelos (*Model-Driven Development* - MDD) (Sheng e Benatallhah, 2005; Hastbacka e Vepsalainen, 2011; Agner et al., 2012), são fundamentadas na ideia de que modelos são mais próximos aos problemas do domínio e menos relacionados a código que as linguagens de programação comuns (Brambilla, 2012). Portanto, modelos são menos dependentes de tecnologia e mais fáceis de serem entendidos e criados. Por outro lado, esta distância impede a evolução de modelos para seguir o código fonte. Consequentemente, eles perdem sua importância, pois tendem a divergir da realidade conforme o software evolui. Portanto, a principal característica do MDD é a geração automática de código a partir de modelos. Preferencialmente, de forma que seja possível manter uma rastreabilidade bidirecional, isto é, dos modelos para o código e deste para os modelos. A disciplina MDD também guia a construção de modelos específicos, com o intuito de facilitar a tarefa de modelagem, e assegurar que as informações necessárias para geração do código estão presentes.

Nossa proposta se assemelha muito ao MDD em certas características. Também propomos a geração de código a partir de modelos (cenários). Porém, no nosso caso, os modelos (cenários) passam por refinamentos consecutivos antes de o código ser produzido. Não obstante, a geração do código é manual, diferente da proposta do MDD, porém é possível automatizar algumas de suas etapas, tornando-a semi-automatizada. Embora o MDD seja considerado independente de tecnologia, um de seus problemas é que ele é mais apropriado para uma categoria específica de problemas. Isto devido a complexidade da geração automática de código e dificuldade de se representar os detalhes necessários no modelo. **Ademais, algumas abordagens MDD permitem apenas a rastreabilidade do modelo para o código**, dificultando a manutenção da atualidade destes modelos. O processo PDS+T, proposto neste trabalho, é de certa forma dependente de tecnologia, pois é orientado a software Web organizado através do padrão MVC, porém, não é restrito a domínios específicos. Além disto, o código é obtido a partir de modelos em linguagem natural, mais fáceis de criar do que os modelos semelhantes a arquitetura, utilizados nas abordagens MDD. Os modelos em linguagem natural podem ser facilmente validados com pessoas que não são especialistas, o que aumenta as chances de produzir software de acordo com as expectativas do usuário. No processo PDS+T, a rastreabilidade entre o código e os cenários é bidirecional, devido a presença dos cenários operacionais integrados ao código. Consequentemente, as mudanças no código podem ser propagadas para a arquitetura e, posteriormente, para os requisitos. O caminho inverso pode ser percorrido da mesma forma.

5.2. Contribuições do Trabalho

Na primeira atividade do processo proposto, são definidas heurísticas que permitem a organização das funcionalidades do software em módulos, a partir do Léxico Ampliado da Linguagem, que descreve a linguagem utilizada no UdI. Além da criação dos módulos, nesta etapa também é realizada a integração das funcionalidades do software e uma visão abstrata do software é criada. Esta característica é importante, pois permite a definição de uma estrutura essencial, que guiará o processo de construção do software, a partir das únicas informações que estão disponíveis até então: os requisitos.

Na segunda atividade do processo proposto, são estabelecidas heurísticas que permitem a criação da arquitetura do software, seguindo o padrão MVC, a partir dos cenários que descrevem suas funcionalidades. Este aspecto é importante, pois estabelece um critério para definição da arquitetura, segundo um padrão já consolidado (MVC), a partir de artefatos obtidos da elicitação de requisitos.

Durante a construção da arquitetura, os cenários iniciais são decompostos e dão origem a outros cenários, mais refinados, que passam a integrar a arquitetura do software. Os cenários refinados, que podem ser vistos como componentes da arquitetura, também são descritos em linguagem natural. Além disso, sua descrição é especializada de acordo com a camada do software a qual pertence, segundo o padrão MVC. A narração contida nestes cenários é extremamente útil, pois serve como uma explicação da arquitetura do software. Adicionalmente, os cenários permitem a descrição de requisitos de qualidade, permitindo sejam detalhados na arquitetura, o que normalmente não é possível.

A última etapa do processo proposto, consiste no refinamento dos cenários da arquitetura em cenários operacionais, que são finalmente codificados. Para esta atividade, são definidas heurísticas, tanto para o refinamento dos cenários da arquitetura, quanto para escrita do código. Apesar de não permitir a geração automática de código, esta característica é importante, pois impõe uma disciplina durante a escrita do código fonte.

Como um dos produtos do software, obtemos os cenários operacionais integrados ao código fonte. Isto é importante, pois serve como uma forma de anotação do código, e é determinante pelos mesmos motivos que justificam a inclusão de comentários no código. Porém, os cenários são ainda mais interessantes, pois possuem uma estrutura definida, centrada na descrição de situações.

O processo de desenvolvimento proposto foi baseado na evolução de cenários (Bergmann et al., 2003; Breitman et al., 2005). Este processo foi elaborado de forma que os cenários produzidos mantivessem seus rastros, através dos diferentes níveis de abstração em que foram descritos. Como os requisitos, a arquitetura e o código possuem cenários associados, obtemos uma rastreabilidade, em ambos os sentidos, desde os requisitos do software, até seu código fonte.

Como consequência do processo proposto, temos os cenários descrevendo os componentes do software, o que inclui seus requisitos, arquitetura e código. Os cenários são descritos em linguagem natural, e sua

estrutura foi pensada de forma a melhorar a comunicação entre quem os escreve e quem os lê. Desta forma, acreditamos que, com esta característica, estamos contribuindo para facilitar o entendimento do software como um todo.

5.3. Limitações do Trabalho

Uma das limitações deste trabalho é o esforço de utilização do processo PDS+T. Embora nenhum estudo empírico tenha sido feito neste sentido, fica claro para nós que sua utilização requer um trabalho extra. Entretanto, há um consenso que uma das atividades mais trabalhosas do ciclo de vida do software é sua evolução. Acreditamos que seja possível justificar este esforço inicial na construção do software através da diminuição dos esforços na atividade de evolução. Além disto, uma ferramenta para apoiar as atividades do processo reduziria consideravelmente este esforço.

As mesmas restrições impostas ao software que pode ser construído, como a interface Web e arquitetura MVC, criadas com o intuito de permitir a definição de heurísticas mais precisas para sua construção, acabam limitando a utilização do processo PDS+T. Porém, já vislumbramos algumas possibilidades de realizar pequenas modificações no processo para acomodar uma amplitude maior de software. Uma destas possibilidades é uma descrição mais genérica de interfaces através de cenários, possibilitando o uso de outras tecnologias para compor a interface do software.

Fica muito difícil de usufruir de algumas características do PDS+T, como rastreabilidade de cenários de requisitos até os do código, sem apoio de uma ferramenta. Isto poderia ser resolvido através da integração da ferramenta C&L com um ambiente de programação, de forma que pudéssemos navegar dos cenários do C&L até os presentes nos documentos de código.

Por contemplar boa parte das fases do desenvolvimento do software, incluindo definição da modularização, arquitetura e escrita do código, o processo PDS+T é bastante extenso. Desta forma, seu uso não é trivial e requer um treinamento prévio.

O processo PDS+T não trata explicitamente dos RNF. Apesar dos cenários permitirem a sua representação através de restrições, seria necessário utilizar uma abordagem específica para tratar esta questão, como por exemplo, aspectos (Filman et al., 2004).

Neste trabalho não contemplamos uma visão explicita dos dados manipulados pelo software. Porém, apesar de não possuir um modelo com este foco, o processo lida com os dados, que estão presentes nos recursos dos cenários que detalham as situações do software.

5.4. Trabalhos Futuros

Como trabalho futuro, no âmbito da ferramenta C&L, vislumbramos a automatização de algumas partes do processo proposto. Dentre as possibilidades de automatização, temos a análise automática dos relacionamentos entre símbolos do LAL e das heurísticas para agrupamento dos cenários, de forma que auxilie ao usuário realizar os ajustes necessários. Outra possibilidade de trabalho futuro é a integração do software C&L com um ambiente de desenvolvimento, de forma que possamos usufruir plenamente da rastreabilidade entre os diferentes níveis de abstração dos cenários, que é proporcionada pelo processo PDS+T. Além disto, podemos automatizar as subatividades de verificação, presentes em cada atividade do processo, de forma que o usuário seja alertado quando um problema for detectado.

Quanto ao processo de desenvolvimento em si, pensamos em evoluí-lo com o propósito de contemplar a construção de software além daqueles que tenham a arquitetura MVC como base. Outra evolução possível seria sua adaptação ao uso de frameworks para desenvolvimento Web, que facilitam o desenvolvimento de acordo com o padrão MVC como, por exemplo, Struts (Struts, 2013) e Spring (Spring, 2013).

Em relação à avaliação do processo proposto, vislumbramos a evolução do trabalho com a realização de novos experimentos. Estes experimentos seriam executados com o intuito de avaliar a execução do processo em si, comparando-o com outros processos voltados para o desenvolvimento de software. Além disto, podemos ampliar os estudos empíricos já realizados, utilizando outros software como base, e realizar uma comparação com outras abordagens que tratam os mesmos conceitos, que são rastreabilidade, dependência e detalhamento.