1
Introdução

Durante o desenvolvimento ou manutenção de um sistema, boas práticas de projeto modular (Gamma et al. 1995, Fowler et al. 1999) deveriam ser aplicadas visando-se maximizar o reúso e a manutenibilidade do mesmo. Entretanto, na prática, nem sempre essa recomendação é respeitada. Sistemas de software sofrem constantes mudanças de diferentes naturezas, levando à produção de uma série de versões ao longo do histórico dos mesmos. Essas alterações inadvertidamente podem levar à introdução de anomalias de modularidade (Fowler et al. 1999) que tendem a impactar negativamente o reúso e a decomposição modular do sistema (Buckley et al. 2006). Essas anomalias, com grande frequência, servem de indicadores da necessidade de refatorar o código da aplicação (Fowler et al. 1999). Nesse contexto, uma questão frequentemente discutida por pesquisadores (Marinescu 2002, Figueiredo et al. 2009, Carneiro et al. 2010a) relaciona-se a descoberta de mecanismos que contribuam com a localização dessas anomalias nos módulos de um sistema.

1.1
Contextualização

De fato, tendo em vista as restrições de tempo ou inexperiência de programadores, é inevitável que anomalias de modularidade sejam introduzidas em código na medida que novas versões do sistema são geradas. Existe uma gama de possíveis anomalias catalogadas (Fowler et al. 1999) que são observadas recorrentemente em programas. O objetivo de se catalogar tais anomalias é justamente difundir o conhecimento sobre elas entre desenvolvedores de forma que sejam evitadas durante o desenvolvimento ou manutenção de sistemas. Além disso, a localização dessas anomalias pode indicar a necessidade de se reestruturar ou refatorar o código de um sistema.

Um exemplo clássico dessas anomalias é a chamada *God Class*, frequentemente caracterizada por ser uma classe complexa e que encapsula muitas responsabilidades do sistema ao mesmo tempo (Riel 1996). Outro problema comum é quando uma determinada necessidade de alteração leva a muitas pequenas alterações em diversas classes – problema intitulado por Fowler (1999)

como *Shotgun Surgery*. Segundo pesquisadores (Fowler et al. 1999), anomalias de modularidade são indesejáveis devido ao impacto negativo relacionado ao reúso e à manutenibilidade do código.

Além de serem candidatas a prejudicar o reúso e a manutenibilidade de código, anomalias como as mencionadas também podem indicar violações básicas da arquitetura de software de um sistema. Nesse sentido, um caso comum seria uma *God Class* que, dentre suas várias preocupações, possui código relativo a responsabilidades de diferentes camadas da arquitetura do sistema. Ou, ainda, que acumula responsabilidades de diferentes interesses do padrão Modelo-Visão-Controle (MVC) (Pree e Sikora 1997). A ocorrência de problemas como esses potencialmente contribuiriam para a degradação da arquitetura do sistema.

Por motivos como os apresentados, é comum recomendar-se que anomalias de modularidade sejam sistematicamente localizadas nos diversos módulos que integram o código de um sistema. Isso possibilita que elas sejam, posteriormente, analisadas e eliminadas. Atividades como essas visam contribuir com o controle da degradação do projeto e com a diminuição das dificuldades relacionados à geração de novas versões de um sistema (Riel 1996).

1.2 O Problema

Sabe-se que é recomendado que o código de sistemas seja refatorado para remover anomalias de modularidade nocivas ao mesmo. Entretanto, uma dificuldade comum em tal processo é que, sem a utilização de mecanismos de avaliação apropriados, a localização dessas anomalias pode se tornar bastante custosa ou até mesmo impraticável (Seção 2.2). Por isso, engenheiros de software têm se dedicado em pesquisar mecanismos que auxiliem nesse processo de detecção.

Métricas (Chidamber e Kemerer 1994, Henderson-Sellers 1996), ferramentas de visualização de código (Demeyer et al. 1999, Carneiro et al. 2010a) e estratégias de detecção (Marinescu 2004, Lanza e Marinescu 2006) têm sido frequentemente exploradas nesse contexto. Estratégias de detecção são regras heurísticas que se baseiam em um conjunto de métricas e valores limites para capturar entidades de código em conformidade com tais valores (Lanza e Marinescu 2006). As métricas utilizadas visam capturar sintomas ou características de possíveis anomalias presentes nas aplicações.

Uma limitação observada em estratégias de detecção é que, normalmente, elas são compostas por métricas que tendem a se concentrar exclusivamente em propriedades de versões particulares dos módulos (Lanza e Marinescu 2006) .

Ou seja, apesar do desenvolvimento de sistemas ser cada vez mais incremental, elas não levam em consideração a evolução das propriedades do código para detectar possíveis anomalias. Sendo assim, elas ignoram propriedades essenciais de sistemas em evolução, tais como frequência das alterações, crescimento contínuo do tamanho ou da complexidade, decrescimento contínuo da coesão, módulos mais instáveis e assim por diante – informações que poderiam ser úteis na detecção de anomalias. Métricas e estratégias de detecção com tais características são chamadas nesta dissertação de convencionais.

A literatura relata (Ratiu et al. 2004, Figueiredo et al. 2009) que estratégias de detecção convencionais (Marinescu 2004, Lanza e Marinescu 2006) têm se apresentado contraproducentes. Frequentemente, um grande número de módulos do sistema que não possuem anomalias acabam sendo detectados – os falsos positivos. Além disso, é comum que módulos com anomalias reais não consigam ser detectados – os chamados falsos negativos. Sendo assim, iniciativas que investiguem as possibilidades de melhorar a eficácia de detecções de anomalias têm se apresentado emergentes entre pesquisadores e é nesse contexto que atua nossa pesquisa.

Observamos que uma possibilidade de estudo seria a avaliação de estratégias que considerassem a evolução das propriedades de código ao longo das versões de um sistema. Por exemplo, possibilitar a identificação de classes com acentuado aumento de tamanho ou de complexidade ao longo de suas versões poderia contribuir com a identificação de algum problema de modularidade? E quanto às classes que apresentam coesão decrescente? Nota-se que considerar a história dos módulos pode contribuir na localização de anomalias que realmente estejam influenciando na evolução do sistema, inclusive impactando em propriedades como estabilidade, crescimento controlado dos módulos, dentre outros.

Estratégias convencionais não fornecem subsídios para avaliações sensíveis à história como as mencionadas no parágrafo anterior. Além disso, sabe-se ainda que um requisito fundamental para testar novos tipos e combinações de estratégias seria o suporte adequado de uma ferramenta de detecção. Para tal ferramenta contribuir com análises comparativas entre os resultados gerados por estratégias convencionais e outras a serem propostas, essa ferramenta precisaria principalmente de: (i) automatizar a execução de diferentes configurações de estratégias e (ii) fornecer suporte a métricas e estratégias sensíveis à história. A literatura, contudo, carece tanto de propostas de métricas e estratégias sensíveis à história quanto de ferramentas capazes de dar suporte aos recursos mencionados.

1.3 Limitações de Trabalhos Relacionados

Poucos trabalhos na literatura têm investigado como informações básicas sobre a evolução dos módulos podem: (i) auxiliar na detecção eficaz de anomalias clássicas de modularidade e (ii) contornar limitações das estratégias baseadas em análises individuais de versões de programas. Existem algumas métricas sensíveis à história propostas recentemente na literatura (Ducasse et al. 2004, Ratiu et al. 2004), mas nenhum trabalho tem explorado as possíveis contribuições de estratégias de detecção sensíveis à história.

Em trabalhos como os de (Ducasse et al. 2004) e de (Ratiu et al. 2004), os autores até consideram algumas métricas de evolução para melhorar a eficácia de duas estratégias de detecção convencionais. Entretanto, as informações sensíveis à história objetivam apenas a redução de falsos positivos obtidos por estratégias convencionais. As anomalias são primeiramente detectadas por estratégias convencionais. Em seguida, dependendo do comportamento evolutivo dos módulos detectados, eles são classificados como prejudiciais ou inofensivos. Ou seja, os módulos não detectados pela abordagem convencional continuam sem ser detectados pela abordagem sensível à história proposta nesses trabalhos.

Quanto às ferramentas de detecção (Together 2009, inCode 2009, iPlasma 2009, inFusion 2009), não existe nenhuma de nosso conhecimento que permita ao programador definir, de forma flexível, diferentes configurações de estratégias. Além disso, elas não dão suporte a métricas sensíveis à história e, dessa forma, não disponibilizam estratégias que permitam considerar o histórico evolutivo das propriedades do código. Tal fato contribui para que não exista qualquer evidência relacionada às possíveis contribuições de estratégias sensíveis à história na redução de falsos positivos e de falsos negativos. Tal negligência ocorre até mesmo para detecção de anomalias de modularidade clássicas como *God Classes* (Riel 1996).

Ferramentas que não disponibilizam a configuração de estratégias, de forma flexível, desconsideram necessidades de diferentes grupos de usuários. Por exemplo, para pesquisadores da área de detecção de anomalias é interessante que uma ferramenta viabilize a aplicação e avaliação das diferentes estratégias que eles possivelmente venham a propor. Já para usuários finais que executam o papel de inspetores de código, tal flexibilidade também é um requisito importante. Esse grupo de usuários, por exemplo, pode identificar necessidades de customização de estratégias em situações bastante comuns, como:

1. uma anomalia a ser detectada ou até mesmo um padrão de codificação a

ser evitado por uma dada empresa não estão contemplados na ferramenta de detecção. Como considerá-los?

2. um limite que identifica uma classe como grande em um determinado domínio precisa ser alterado. Como ajustar esse valor limite?

3. uma nova combinação de métricas foi descoberta como mais eficiente na detecção de uma dada anomalia. Como substituir na ferramenta a combinação de métricas considerada para tal detecção?

Além disso, seria desejável que todas essas possibilidades de configuração pudessem ser realizadas em alto nível. Ou seja, estratégias de detecção deveriam poder ser criadas, alteradas e ainda avaliadas sem que fosse necessário alterar explicitamente a implementação em nível de código da ferramenta de detecção. Entretanto, isso não é comum entre as ferramentas.

Normalmente, ferramentas atuais e até bastante conhecidas, como o caso da Together (Together 2009), limitam as decisões sobre anomalias, métricas e valores limites ao grupo de desenvolvedores que as desenvolveu. Ou seja, qualquer necessidade de alteração remete à necessidade de alterações a serem realizadas diretamente no código e apenas por esses profissionais. Avaliadores de código ou pesquisadores da área de detecção não têm a possibilidade de especificar em alto nível e de forma flexível as estratégias requeridas por cada um.

Sendo assim, podemos dizer que a falta de suporte a métricas e estratégias de detecção sensíveis à história associadas à falta de flexibilidade para criar e alterar estratégias são algumas das principais limitações das ferramentas atuais. Tais ferramentas acabam limitando ou não considerando necessidades particulares de usuários na adaptação e criação de estratégias de detecção, além de dificultarem a pesquisa e avaliação de novas possibilidades de estratégias.

1.4 Visão Geral da Solução

A principal questão discutida neste trabalho é "estratégias que consideram a evolução das propriedades do código podem ser um recurso eficaz na detecção de anomalias de modularidade de código?" Como mencionamos na seção anterior, alguns fatores dificultam a realização de pesquisas que abordem tal questão. Alguns desses fatores são a falta de métricas e estratégias sensíveis à história e de ferramentas para automatização desses recursos.

Nesse contexto estão algumas das contribuições desta dissertação. Com o objetivo de suprir as limitações discutidas na seção anterior, este trabalho

apresenta a proposição de um conjunto inicial de (i) métricas sensíveis à história, (ii) estratégias de detecção sensíveis à história e o (iii) desenvolvimento de uma ferramenta de detecção que considera tanto estratégias convencionais quanto estratégias sensíveis à história. Além disso, (iv) estudos empíricos também foram realizados para avaliação das estratégias de detecção promovendo discussões importantes sobre as detecções geradas.

Uma das grandes contribuições da ferramenta proposta, a Hist-Inspect, é que ela permite criar novas estratégias de detecção de forma declarativa e em elevado nível de abstração. Como já havíamos destacado, em ferramentas clássicas como o Together (Together 2009) ou mesmo a inCode (inCode 2009), apenas os desenvolvedores dessas ferramentas são capazes de realizar as alterações necessárias para a definição de uma nova estratégia de detecção. Isso porque nessas ferramentas, tal necessidade implica na definição de novos algoritmos de detecção que precisam ser implementados em nível de código com novas classes, métodos e assim por diante.

Já na Hist-Inspect tais alterações podem ser realizadas por usuários finais. Para isso, eles especificam as estratégias em alto nível utilizando o conhecimento próprio desse domínio: siglas de métricas, operadores de comparação (por exemplo, $>$, $<$), valores limites, dentre outros elementos que integram tal domínio (Seção 3.2). Ou seja, novos algoritmos de detecção passam a ser imediatamente considerados pela ferramenta a partir de especificações em alto nível realizadas de acordo com as necessidades de cada usuário.

Tal benefício foi obtido por meio da utilização de uma linguagem específica de domínio (DSL) a qual chamamos de DSSL, Linguagem de Especificação de Estratégias de Detecção. Essa decisão visou solucionar limitações relacionadas aos problemas 1, 2 e 3 mencionados na seção anterior. O usuário, dessa forma, passa a ter mais liberdade quanto às estratégias consideradas em avaliações de código, pois ele mesmo passa a especificar as anomalias e estratégias de acordo com suas próprias necessidades. A Hist-Inspect também oferece uma visão panorâmina da evolução de propriedades do código através de gráficos de evolução. De nosso conhecimento, os recursos da Hist-Inspect listados não são encontrados em outras ferramentas de suporte a estratégias de detecção.

Na etapa de realização dos estudos empíricos, nos concentramos na avaliação sistemática de estratégias sensíveis à história para detectar três anomalias clássicas de modularidade. As estratégias consideraram a detecção das anomalias *God Class* (GC), *Divergent Change* (DC) e *Shotgun Surgery* (SS) (Fowler et al. 1999) ao longo de 16 versões de 2 sistemas de domínios heterogêneos. Além disso, os sistemas também são marcados por históricos de evolução diferenciados: (1) um deles dominado por vários incrementos e

remoções de funcionalidades, e (2) o outro por modificações estruturais no projeto e código ao longo das versões geradas. A avaliação de eficácia foi realizada considerando-se medidas de precisão e revocação das estratégias (Moha et al. 2010)(Rijsbergen 2001). Resultados das estratégias sensíveis à história também foram comparados aos de estratégicas convencionais.

No sistema com o histórico de evolução do tipo (1), as estratégias sensíveis à história apresentaram excelentes resultados de detecção. Por exemplo, em algumas versões foi possível a detecção de 100% das anomalias do tipo *God Class*, sem a ocorrência de falsos positivos. Já no caso (2), em que o sistema sofreu menos alterações ao longo de suas versões, as estratégias sensíveis à história não se apresentaram tão eficazes. Por exemplo, nas detecções da anomalia *God Class* o valor máximo de revocação foi de 33%. Apesar disso, tanto a média de revocação quanto a média de precisão das estratégias sensíveis à história foram superiores às médias das estratégias convencionais.

Parte das avaliações deste trabalho contou com a colaboração de pesquisadores de outras instituições: um pesquisador e um professor da Universidade Federal da Bahia (UFBA) e um professor da Universidade Federal de Minas Gerais (UFMG). A colaboração desses pesquisadores foi de extrema importância durante a elaboração dos oráculos de detecção, ou seja, na identificação manual das anomalias nos sistemas avaliados. Tal idenficação possibilitou avaliar os resultados gerados por cada estratégia com anomalias detectadas manualmente por especialistas em modularização de sistemas. Além disso, planejamos e realizamos com esses pesquisadores um estudo exploratório que permitiu analisar como ferramentas de visualização também tinham deficiências e que possivelmente poderiam ser contornadas por detecções sensíveis à história.

O estudo exploratório realizado em parceria com tais colaboradores, possibilitou nesta dissertação análises comparativas entre as detecções baseadas em estratégias sensíveis à história e as baseadas em recursos de visualização de código. Nas análises foi observado, de fato, que deficiências nas detecções resultantes da abordagem visual poderiam ser suprimidas pela utilização da abordagem sensível à história e que o inverso também se aplicava.

Por exemplo, de forma particular, na detecção de *God Classes* enquanto em todas as versões avaliadas a estratégia sensível à história não detectou nenhum falso positivo, a detecção baseada na ferramenta de visualização teve falsos positivos superiores à quantidade de acertos em todas as versões. Em algumas dessas versões, o número de falsos positivos obtidos pela abordagem visual chegou a cinco vezes o número de acertos. Em contrapartida, os recursos visuais possibilitaram a localização de anomalias reais que não conseguiram ser detectadas pela abordagem sensível à história. Optamos justamente por con-

siderar esses diferentes tipos de abordagens (convencional, sensível à história e visual) com o objetivo de facilitar futuros estudos sobre mecanismos de detecção alternativos. Por exemplo, diferentes abordagens poderiam ser combinadas, de forma que uma pudesse minimizar as deficiências da outra.

Como resumo das avaliações, percebemos que estratégias sensíveis à história podem trazer importantes contribuições em detecção de anomalias. Identificamos que estratégias sensíveis à história podem não apenas contribuir com detecções eficazes de anomalias de modularidade como também podem se apresentar mais eficazes que estratégias convencionais difundidas pela literatura. Além disso, elas podem ser utilizadas para diminuir a incidência de falsos positivos gerados por abordagens de detecção baseadas em recursos de visualização de código. Essa pesquisa, portanto, fornece evidências iniciais que motivam outros estudos que associem à utilização de informações sensíveis à história com diferentes recursos de detecção de anomalias de código.

1.5 Estrutura do Texto

O restante deste trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta uma visão geral de conceitos pertecentes ao domínio deste trabalho. Tais conceitos compreendem manutenção de sistemas, refatoração de código, anomalias de modularidade e avaliação sensível à história de código. No Capítulo 3, o trabalho é contextualizado com o estado da arte em pesquisas e publicações que estejam relacionados ao nosso trabalho. Neste caso, enfatizamos principalmente métricas para avaliação de código, estratégias de detecção e ferramentas relacionadas.

No Capítulo 4, são apresentadas as métricas sensíveis à história propostas nesta dissertação. No Capítulo 5, são apresentados alguns exemplos de estratégias sensíveis à história que podem ser formuladas com o uso das métricas definidas no Capítulo 4. Tais estratégias podem ser aplicadas tanto para recuperar um grupo de módulos do sistema que tenham um mesmo comportamento evolutivo (por exemplo, coesão decrescente em cada nova versão de um sistema) quanto para identificação de anomalias de modularidade difundidas pela literatura.

No Capítulo 6, são apresentadas as principais características da ferramenta de medição e avaliação proposta nesta dissertação: a Hist-Inspect. No Capítulo 7, discutimos resultados de avaliações relacionados à aplicação de estratégias de detecção sensíveis à história, convencionais e o uso de ferramentas de visualização ao longo de versões de dois sistemas. Finalmente, no Capítulo 8 são resumidas as principais contribuições deste trabalho bem como

possibilidades de trabalhos futuros.