Rafael de Pinho André

Avaliação do Uso de Análise Estática na Detecção de Conflitos Semânticos em Tipos de Dados

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Centro Técnico Científico da PUC-Rio.

Orientador: Prof. Arndt von Staa

Rio de Janeiro

Abril de 2013

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael de Pinho André

Avaliação do Uso de Análise Estática na Detecção de Conflitos Semânticos em Tipos de Dados

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Arndt von Staa
Orientador
Departamento de Informática – PUC-Rio

Prof. Julio Cesar Sampaio do Prado Leite
Departamento de Informática – PUC-Rio

Profª. Simone Diniz Junqueira Barbosa
Departamento de Informática – PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 4 de Abril de 2013

Rafael de Pinho André

Graduou-se em Engenharia de Computação na Pontíficia Universidade Católica do Rio de Janeiro (PUC-Rio) em 2005. Em 2006, tornou-se especialista em Gerenciamento de Projetos de Software pela PUC-Rio. Em 2007, tornou-se especialista em Criptografia e Segurança em Redes pela Universidade Federal Fluminense e o Instituto Militar de Engenharia. Desde 2006 trabalha como consultor em gerenciamento de projetos e gestão de políticas de segurança da informação. É coordenador acadêmico do curso de Programação do Instituto de Tecnologia ORT. Suas principais áreas de interesse são Engenharia de Software e Sistemas Distribuídos.

Ficha Catalográfica

André, Rafael de Pinho

Avaliação do uso de análise estática na detecção de conflitos semânticos em tipos de dados / Rafael de Pinho André ; orientador: Arndt Von Staa. – 2013.
95 f. : il. (color.) ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2013.
Inclui bibliografia

1. Informática – Teses. 2. Análise estática. 3. Ferramentas de software. 4. Notação. 5. Qualidade. 6. Semântica. 7. Verificação. I. Staa, Arndt von. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

"There is the name and the thing; the name is a sound which sets a mark on and denotes the thing. The name is no part of the thing nor of the substance; it is an extraneous piece added to the thing, and outside of it."

- Michel Eyquem de Montaigne

Agradecimentos

A Deus, por tudo que tenho e por tudo que ainda me será concedido. Seu amor e generosidade sempre presentes me concederam maravilhosas oportunidades.

À minha família, especialmente minha mãe Regina e meu padrasto Renato, pelo apoio que recebi em toda a minha vida e o grande esforço para me proporcionar uma educação digna. Foram os exemplos em casa que me guiaram até este momento.

A Daniella, minha mulher, por estar ao meu lado em todos os momentos. Sem seu apoio e dedicação este trabalho não seria possível.

Ao meu orientador prof. Arndt von Staa, por dividir comigo seu conhecimento, entender as dificuldades que enfrentei e ajudar a tornar este trabalho possível. É uma honra poder fazer parte de sua história.

À PUC-Rio e ao Instituto de Tecnologia ORT, onde passei grande parte de toda a minha vida adulta. Recebi uma educação de excelência e lições que me transformaram como pessoa e profissional.

Aos muitos mestres que tive na vida e deixaram um pouco de sua experiência e sabedoria comigo.

Ao prof. Markus Endler e o LAC, por todo o apoio ao longo do curso e a oportunidade de ingressar na pesquisa acadêmica.

Aos meus amigos, por me incentivarem a enfrentar as dificuldades, pelo amparo nos momentos difíceis e pelas alegrias de todos os momentos bons que compartilhamos.

Aos colaboradores e parceiros deste trabalho, tão pacientes e disponíveis para participar do estudo e oferecer ajuda. Vocês acreditaram neste trabalho em todos os momentos, e me emprestaram coragem era difícil encontrá-la. Obrigado.

Resumo

André, Rafael de Pinho; von Staa, Arndt. **Avaliação do Uso de Análise Estática na Detecção de Conflitos Semânticos em Tipos de Dados.** Rio de Janeiro, 2012. 95p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Em um sistema de informação, falhas podem ocorrer pela diferença de entendimento das partes envolvidas em relação ao significado de um dado. Este é um problema bem conhecido pela engenharia de software, e defeitos deste tipo já foram responsáveis por falhas catastróficas, como a do Mars Climate Orbiter em 1999. O atual cenário de intercâmbio e processamento de dados, com grande volume de informação e heterogeneidade de participantes, cria um estado de suscetibilidade a estes defeitos. Entretanto, as técnicas de garantia de qualidade de software são tipicamente dirigidas à estrutura e às propriedades físicas dos dados, e não são eficientes ao observar questões semânticas. Este trabalho tem como intuito avaliar o uso de análise estática na detecção de conflitos semânticos em tipos de dados, e para validar sua eficácia esta abordagem foi comparada com outras técnicas de garantia de qualidade em um estudo qualitativo. A ferramenta de análise estática VERITAS (VERIficador estTÁtico Semântico) e a notação SemTypes foram desenvolvidas exclusivamente para tratar do problema de conflitos semânticos, adicionando controle de tipo semântico aos tipos reconhecidos por compiladores, e são apresentadas neste trabalho.

Palavras-chave

Análise Estática; Ferramentas de Software; Notação; Qualidade; Semântica; Verificação.

Abstract

André, Rafael de Pinho; von Staa, Arndt (Advisor). **Evaluation of Static Analysis in Data Type Semantic Conflict Detection.** Rio de Janeiro, 2012. 95p. MSc. Dissertation– Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Within information system, faults can occur by the difference in understanding of the parties involved regarding the meaning of data. This is a well-known problem for software engineering and defects of this type have been responsible for catastrophic failures, such as the Mars Climate Orbiter in 1999. The current scenario of data processing and exchange, with high information traffic volume and heterogeneous participants, increases system's vulnerability to these defects. Besides that, techniques of software quality assurance are typically oriented to data structure and physical properties, failing to efficiently address semantics issues. This work has the objective to evaluate the use of static analysis to detect semantic conflicts in data types, investigating its efficacy through an qualitative study comparing different software quality assurance approaches. The static analysis tool VERITAS (VERIficador esTÁtico Semântico) and the SemTypes notation were exclusively developed to address the problem of semantic conflicts - adding a semantic control to the types recognized by compilers – and are presented in this work.

Keywords

Static Analysis; Software Tools; Notation; Quality; Semantics; Verification.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Computadores operam sobre dados, o nível mais baixo da abstração de onde os conceitos de informação e conhecimento são derivados. Dados são valores, qualitativos ou quantitativos, que descrevem um conjunto de entidades e são expressos na forma de símbolos. A informação, por outro lado, é um conjunto de dados ao qual é atribuído significado. Em outras palavras, um dado isolado usualmente não carrega significado.

O tipo computacional de um dado determina como este será processado por um computador, impondo restrições sobre atribuições e operações: não é possível armazenar uma grandeza de ponto flutuante em uma variável inteira, ou multiplicar uma sequência de caracteres por uma variável booleana. Estes tipos computacionais estão normalmente relacionados aos tipos de dados primitivos de uma linguagem. Estes tipos primitivos são os blocos básicos de construção oferecidos por uma linguagem, através dos quais tipos compostos – como estruturas de dados e classes – podem ser elaborados.

Não se restringindo somente a tipos primitivos oferecidos pelas linguagens de programação, tipos computacionais incluem os tipos abstratos definidos pelos usuários, como estruturas de dados e classes. Entretanto, embora dados possuam tipo computacional, seu significado é dependente de seu tipo semântico. Este tipo semântico é o significado atribuído pelo desenvolvedor e definido no momento de declaração de uma entidade, seja uma variável, função ou tipo abstrato. O nome de uma pessoa é semanticamente diferente do nome de um logradouro, e não se deve permitir que uma variável do primeiro tipo seja atribuída a uma variável do segundo tipo. Da mesma forma, uma função de cálculo da média ponderada de uma lista de notas de avaliações escolares não deveria aceitar que haja na lista um elemento representando a matrícula de um aluno, ainda que os tipos computacionais sejam os mesmos. Ou seja, assim como o tipo computacional, o tipo semântico também impõe restrições sobre atribuições e operações válidas para uma entidade.

Observar corretamente o tipo computacional e semântico de um dado, e tratá-lo adequadamente, é um requisito intrínseco de sistemas de informação. De outra forma, existe o risco de que o dado não seja utilizado corretamente e um defeito seja inserido no artefato. Estes defeitos, causados pela diferença de entendimento das partes envolvidas em relação ao significado de um dado, podem acarretar falhas de um sistema. Este é um problema bem conhecido pela engenharia de software, e defeitos deste tipo já foram responsáveis por falhas catastróficas, como a do Mars Climate Orbiter. A sonda espacial conhecida como Mars Climate Orbiter foi desenvolvida pela NASA para estudar o clima e a atmosfera de Marte, servindo ao mesmo tempo como estação de comunicações para o Mars Polar Lander. Seu lançamento ocorreu em dezembro de 1998, e em setembro de 1999 a NASA perdeu o contato com a sonda quando esta se desintegrou ao entrar na atmosfera de Marte. A causa do acidente foi o seguinte defeito no software de controle do Mars Climate Orbiter: diferentes módulos do sistema utilizavam diferentes sistemas métricos para representar grandezas físicas, acarretando inconsistências nas mensagens trocadas. Neste caso, como os diferentes sistemas métricos eram representados pelos mesmos tipos computacionais no sistema, o defeito não foi detectado.

Estes defeitos serão chamados de conflitos semânticos. Conflitos semânticos são violações que ocorrem quando operações e atribuições inadequadas para os tipos envolvidos são executadas ou propostas. Uma abordagem tradicional de desenvolvimento de software é conhecida como programação por contrato, ou `DbC` (`Design by contract`), e sugere que especificações precisas, formais e verificáveis de interface deveriam ser definidas para os artefatos (Meyer, 1994). O `DbC` faz forte uso dos conceitos de pré-condição, pós-condição e invariantes. Embora nem sempre os tipos semânticos de entidades de um artefato estejam claramente definidos na especificação deste artefato, sempre existe um contrato semântico implícito. Este contrato, um conjunto de obrigações e condições, é definido pelo desenvolvedor no momento em que a entidade é criada. Conflitos semânticos são as ocorrências de violação do contrato semântico.

O atual cenário de intercâmbio de dados, com grande volume de tráfego e heterogeneidade de participantes, cria um estado de suscetibilidade à quebra destes contratos. Alguns fatores como o crescimento da colaboração em larga

escala, a proliferação das redes sociais e a expansão da computação móvel sugerem que este cenário dificilmente se tornará menos desafiador. Entretanto, as técnicas de garantia de qualidade de software são tipicamente dirigidas à verificação e análise dos tipos computacionais dos dados e suas propriedades físicas, falhando ao verificar e analisar a concordância semântica dos sistemas.

1.1 Objetivo do Trabalho

Este trabalho tem o intuito de avaliar, através de um estudo qualitativo, o uso de análise estática na detecção de conflitos semânticos em tipos de dados. Deseja-se verificar se esta abordagem produz resultados satisfatórios, custo adequado e capacidade de se adaptar a diferentes processos e ambientes de desenvolvimento de software. Ao mesmo tempo, o trabalho também tem como objetivo o desenvolvimento de uma notação e uma ferramenta de análise estática voltadas exclusivamente para o problema de conflitos semânticos em tipos de dados. Notações são discutidas em detalhe no próximo capítulo.

A abordagem da análise estática consiste em analisar um artefato de software sem necessidade de sua execução, ao contrário da análise dinâmica que opera através da execução do artefato analisado. O uso do termo análise estática implica a utilização de uma ferramenta automatizada; quando este não é o caso, os termos inspeção ou verificação costumam ser utilizados. Ambos os tipos de análise operam tipicamente sobre código-fonte para realizar a análise, embora em alguns casos o código-objeto seja utilizado. A análise estática, ao contrário dos métodos formais, não prova a corretude de um software, sendo apenas capaz de examinar um artefato em busca de defeitos de uma natureza bem especificada. O uso de métodos formais e sua relação com este trabalho é discutida no próximo capítulo.

Os resultados deste trabalho de desenvolvimento foram a notação `SemTypes`, utilizada no estabelecimento de contratos semânticos, e a ferramenta de análise estática `VERITAS`, que simula a adição de controle semântico aos tipos reconhecidos por compiladores. Ambas são apresentadas e discutidas em detalhes em um capítulo dedicado a este propósito. Faz parte do escopo do objetivo deste

trabalho avaliar a notação e a ferramenta como exemplo de uso da abordagem de análise estática na detecção de conflitos semânticos.

O primeiro passo do trabalho é observar o desempenho de abordagens de garantia da qualidade, diferentes da análise estática, na detecção de conflitos semânticos. A razão deste passo é definir uma base de comparação para avaliar a eficiência e eficácia da análise estática, situando os resultados em um contexto real de processos e técnicas de garantia da qualidade de software. Para isto, foi selecionada uma abordagem de verificação e uma de validação: inspeção e testes unitários automatizados. Ferramentas de teste automatizado são consideradas ferramentas de análise dinâmica, o que torna sua comparação com a `VERITAS` particularmente interessante. A discussão sobre a escolha destas abordagens é apresentada no capítulo que trata exclusivamente do estudo realizado.

Embora a taxa de detecção seja uma das informações mais importantes do desempenho das abordagens selecionadas, este trabalho se propõe a realizar uma análise mais ampla das características destas abordagens. Medir o custo em tempo de desenvolvimento e recursos computacionais é muito importante para a avaliação da solução proposta, pois sua relação com o benefício percebido irá determinar o uso ou descarte da abordagem por parte de uma organização ou equipe. Esta medição deve ser cautelosa, pois em certos cenários os custos possuem uma tendência acentuada nas fases iniciais, sendo então diluído ao longo do tempo, algumas vezes ao ponto de se tornarem irrelevantes para o projeto ou empresa. Em outros cenários o custo inicial não é elevado, mas o custo operacional é proporcional ao resultado desejado. Ferramentas automatizadas se enquadram, ou deveriam se enquadrar, no primeiro cenário. Isto engloba os testes unitários automatizados e a ferramenta `VERITAS`. Já a inspeção humana, manual, possui características que a enquadra no segundo cenário, mesmo quando considerado o fator treinamento.

A medição dos custos será planejada para considerar estas particularidades e obter resultados mais consistentes e confiáveis. A avaliação das características de treinamento e manutenção das abordagens, assim como a satisfação dos desenvolvedores e percepção de benefício com seu uso, fazem parte do escopo do estudo e são necessárias para a comparação desejada. Alguns processos e métodos de engenharia de software, tais como desenvolvimento orientado a testes e até

mesmo inspeções de código, enfrentam grande resistência de equipes de desenvolvimento (Cohen, 2006). É comum que estes métodos e processos sejam progressivamente abandonados pelos desenvolvedores, caso não haja reforço contínuo e observação cuidadosa por parte das organizações.

Estas impressões, embora de natureza extremamente subjetiva dentro do escopo do estudo, são tão importantes quanto as métricas para a avaliação das abordagens. O investimento financeiro não é relevante neste estudo, pois todas as abordagens utilizadas não necessitam de software ou hardware comerciais, e os resultados e conclusões que poderiam ser obtidos deste tipo de análise fugiriam ao escopo do trabalho.

O segundo passo do trabalho é observar o desempenho da análise estática no tratamento dos mesmos problemas de detecção de conflitos semânticos. As mesmas métricas e impressões de custo, eficiência, treinamento e manutenção que serão coletadas para a inspeção e os testes unitários devem ser coletadas na avaliação da notação e da ferramenta. Isto é necessário para permitir a comparação e conclusões desejadas no trabalho. O conjunto de métricas de processos de garantia da qualidade, os procedimentos de uso e o roteiro de entrevista serão elaborados para que a avaliação do uso de análise estática na detecção de conflitos semânticos ocorra concomitantemente à avaliação da notação e ferramenta propostas neste trabalho, em função dos recursos limitados do estudo. Uma das metas é a obtenção de pelo menos duas equipes voluntárias, para que se possa observar diferenças e notar situações excepcionais. Estas equipes devem, preferencialmente, ser de empresas distintas, trabalhar em setores diferentes do mercado, e utilizar ferramentas, processos e métodos variados – aumentando o número de casos de uso exercitados da notação e da ferramenta.

O último passo deste trabalho é avaliar o impacto de uso da notação `SemTypes` em todas as fases de desenvolvimento de um artefato de software. É possível que, a partir da percepção da necessidade e dos benefícios de se estabelecer um contrato semântico, a forma como os artefatos são concebidos, elaborados e aprovados se altere. Caso isto ocorra, talvez haja impacto nos processos e métodos de levantamento e análise de requisitos, modelagem, especificação e testes, por exemplo. Em outras palavras, ao reconhecer defeitos derivados de conflitos semânticos e utilizar uma notação para mitigar o risco de

existirem defeitos desta classe, um grande número de benefícios secundários pode ser obtido. Até mesmo a comunicação interna entre os membros de uma equipe, ou entre estes membros e stakeholders do projeto, pode ser beneficiada por uma maior precisão semântica. O objetivo das entrevistas que serão realizadas no estudo qualitativo é justamente obter este tipo de impressão, usualmente complexas demais para serem obtidas através do preenchimento de formulários pelos participantes voluntários.

1.2 Estrutura dos Capítulos

Este trabalho é dividido em cinco capítulos e conta com três apêndices. O próximo capítulo descreve o estado da arte das principais áreas de conhecimento e ferramentas relacionadas a este trabalho. Diferentes abordagens, como o uso de métodos formais e linguagens específicas de domínio, são apresentadas. Para cada abordagem, uma pequena discussão é apresentada acerca de seus benefícios e limitações, e também é explorada sua relação com a proposta deste trabalho. No capítulo três são apresentados, em uma discussão detalhada, os três principais componentes da solução proposta neste trabalho:

- A ferramenta `ANTLR` (`ANother Tool for Language Recognition`): um gerador de parser `LL(*)` utilizado para montar a árvore sintática abstrata, ou `AST` (`Abstract Syntax Tree`), do código analisado.
- A notação `SemTypes`: conjunto de regras para a definição de contratos semânticos através de identificadores de variáveis, tipos e funções. Estas regras são estruturadas de forma hierárquica em um documento `XML`, nomeado de arquivo de definição semântica.
- A ferramenta `VERITAS`: um verificador de quebras de contrato semântico. Processa uma `AST` fornecida, utilizando as regras declaradas nos arquivos de definição semântica como base para detectar conflitos semânticos no artefato.

O capítulo quatro descreve (i) o estudo qualitativo realizado para avaliar a eficiência e eficácia do uso de análise estática para detectar conflitos semânticos e

(ii) a avaliação da notação `SemTypes` e da ferramenta `VERITAS`. Os objetivos e método do estudo são discutidos no início, seguidos da descrição dos participantes, da execução, da coleta dos dados e das entrevistas. Ao final, os resultados são apresentados e analisados.

O quinto e último capítulo contém conclusões e observações de todo o processo de elaboração deste trabalho, contando com uma seção dedicada às contribuições obtidas e outra com oportunidades de trabalhos futuros.

O apêndice A inclui o termo de consentimento utilizado no estudo qualitativo, preenchido apenas pelo líder de cada time de desenvolvedores. O termo descreve brevemente o trabalho proposto e a participação esperada do voluntário, e também estabelece as garantias de privacidade e anonimato. O apêndice B apresenta o formulário de resultados e observações, elaborado pelo time de desenvolvedores ao final de cada iteração sob a orientação do líder da equipe. Este formulário contém informações qualitativas e qualitativas sobre o uso da ferramenta e seu impacto nos processos de garantia de qualidade. Por último, o apêndice C contém o roteiro da entrevista realizada com o líder e um desenvolvedor de cada equipe, utilizada para complementar o formulário de resultados e observações.

2 Estado da Arte

Este trabalho avalia experimentalmente o uso de análise estática na detecção de quebras do contrato semântico na utilização de tipos de dados. Em outras palavras, avalia o uso de uma abordagem tradicional de garantia de qualidade de software na detecção de um tipo especifico, e bem conhecido, de defeito em sistemas de informação.

Se considerarmos o defeito estudado como um risco, podemos pensar em estratégias para (i) aceitação, (ii) mitigação ou (iii) prevenção. Aceitar este risco, adotando um tratamento similar ao `Ostrich Algorithm` descrito em Rensselaer Polytechnic Institute (2004), não é uma opção viável. Esta abordagem de aceitação é amplamente utilizada em programação concorrente e sistemas operacionais, sendo aplicada aos casos em que (i) se acredita que a probabilidade de ocorrência de um defeito é tão baixa que ele apenas ocorre raramente, ou que (ii) o custo de se prevenir ou evitar é maior que o possível impacto, não justificando qualquer tratamento.

No caso de quebras do contrato semântico no desenvolvimento de sistemas de informação, sabemos que a probabilidade de ocorrência de uma falha não é tão baixa. Também não é possível assegurar que o impacto de uma falha será menor que o custo do seu tratamento, a não ser no caso de sistemas em que o maior prejuízo por sua não conformidade com a especificação seja uma inconveniência para o usuário. Mitigar o risco – reduzir sua chance de ocorrência – é o objetivo central deste trabalho e de outras ferramentas e técnicas de garantia da qualidade de software.

Neste capítulo serão apresentados alguns dos mais relevantes trabalhos que contribuem para a redução do risco de quebra do contrato semântico, seja por construção, verificação ou validação. Existe também a abordagem de evitar o risco, que consiste em adotar um paradigma onde o risco não exista ou não seja relevante. Particularmente, as `Domain Specific Languages`, ou `DSL`, propõem um paradigma de desenvolvimento que evita quebras de contrato semântico, e também serão abordadas neste capítulo. Abaixo, na seção 3.1, são discutidos trabalhos relacionados à mitigação do risco e, na seção 3.2, são

apresentados alguns trabalhos ou áreas de conhecimento que podem auxiliar em sua prevenção.

2.1 Abordagens que Mitigam o Risco

A notação húngara de Simonyi (2006) é uma convenção de nomeação de identificadores, e como toda convenção de nomeação, impõe um conjunto de regras que deve ser utilizado para validar a sequência de caracteres de um identificador – variáveis, tipos ou funções – tanto em código-fonte quanto em outros artefatos do sistema como os de documentação. A notação[1] foi desenvolvida pelo desenvolvedor Charles Simonyi, que após trabalhar cerca de dez anos no XEROX PARC atuou como arquiteto chefe na Microsoft em grandes projetos como o processador de textos WORD.

A notação húngara foi desenvolvida para ser independente de linguagem, definindo que um identificador deve iniciar com um conjunto específico de caracteres em caixa-baixa, chamado de mnemonics, e ser seguido do nome desejado, ou given name. O given name não possui nenhum significado específico na notação, mas seu primeiro caractere é escrito em caixa-alta, tornando sua distinção mais fácil e aumentando a legibilidade do artefato. O mnemonics é a parte do identificador que deve expressar seu propósito ou origem, o que explica o nome da notação: na Hungria, país de origem de Charles Simonyi, o nome de família precede o nome dado à pessoa. Existem dois tipos de notação húngara: a Apps Hungarian Notation e a Systems Hungarian Notation. A versão Systems Hungarian Notation foi desenvolvida por uma equipe de desenvolvedores do grupo de sistemas da Microsoft, especificamente do Microsoft Windows, com base na Apps Hungarian. Esta versão da notação define que mnemonics devem ser utilizados para expressar o tipo físico ou tipo de dados, como por exemplo:

- iExpirationTime (a variável é um inteiro)
- zsGuestList (a variável é uma string terminada em zero)

1 Microsoft Hungarian Notation.
http://msdn.microsoft.com/en-us/library/aa260976%28VS.60%29

- bRead(iPort) (função que recebe um inteiro e retorna um valor booleano)

Esta forma da notação pouco contribui para o problema abordado neste trabalho. Embora seja considerada adequada em linguagens fracamente tipadas, onde fornece uma estrutura mínima para verificação de tipos, a `Systems Hungarian` se torna redundante nas linguagens em que compiladores executam esta verificação. Os ambientes integrados de desenvolvimento atuais, em sua maioria, auxiliam o desenvolvedor destacando usos incompatíveis do tipo físico de uma variável, tipo ou função, e permitem que o desenvolvedor conheça esse tipo em qualquer trecho do código, sem que seja necessário recorrer à definição. Um conceito um pouco mais específico que o definido pela `Systems Hungarian` é o `sigil`. Um `sigil` é um símbolo que deve ser utilizado como prefixo ou sufixo de um identificador, e indica o tipo de dado ou escopo de uma variável ou função. Entretanto, um `sigil` é um elemento natural da linguagem, sendo reconhecido pelo compilador e necessário do ponto de vista sintático. Do ponto de vista deste trabalho, `sigils` se comportam da mesma forma que a `Systems Hungarian`, pois são incapazes de adicionar controle semântico ao processo de desenvolvimento.

A notação `Apps Hungarian`, por sua vez, possui grande relevância para este trabalho. Ela é a versão original desenvolvida por Charles Simonyi no `XEROX PARC`, e recebeu este nome ao ser utilizada na divisão `Applications` da `Microsoft`. Nesta versão da notação, existe um conceito conhecido como `mnemonics`. Um `mnemonic` representa o que foi descrito como tipo lógico de uma variável, tipo ou função, e não seu tipo físico. No documento original de Simonyi (1977) é utilizada a palavra inglesa `kind`, e não `type`, para descrever o significado de um `mnemonic`, para que sua intenção fique clara. Isto é o que ele chama de tipo lógico, o propósito da variável, tipo ou função. Alguns exemplos:

- sExpirationTime (a variável representa o número de segundos de validade de uma entidade)
- usFormData (a variável representa dados de um formulário que foram inseridos por um usuário e podem não ser seguros – *unsafe*)
- dLength (a variável representa uma diferença de comprimento)

Ao comparar estes exemplos, da `Apps Hungarian`, com os exemplos anteriores da `Systems Hungarian`, é fácil perceber a diferença de uso e objetivos entre as notações. A proposta da notação original de Charles Simonyi, a `Apps Hungarian`, é a base conceitual deste trabalho. Sua estrutura orientada a propósito inspirou o desenvolvimento de uma notação com poder de expressão suficiente para definir tipos semânticos. A notação `SemTypes`, desenvolvida neste trabalho, adiciona elementos semânticos aos identificadores de um código-fonte. Ela torna possível o processamento destes identificadores por computador e, consequentemente, viabiliza a utilização de análise estática na detecção de quebras de contrato semântico.

Um grande número de convenções e boas práticas de nomeação de identificadores são propostas na literatura, como em Martin (2008, 2011). O objetivo destas práticas e convenções é ajudar desenvolvedores de software a tornar o código-fonte mais legível e fazer com que erros se destaquem, melhorando a qualidade geral dos artefatos e permitindo que sejam compreendidos mais rápida e integralmente. Todas estas convenções e práticas não atuam diretamente na detecção de quebras de contratos semânticos, mas sim no aumento da chance de detecção – tema central deste trabalho. Entretanto, a necessidade de inspeção humana limita a eficiência e a eficácia destas propostas, e apenas a melhoria de legibilidade não é adequada para garantir assegurar a qualidade e a baixa presença de defeitos de origem semântica no produto final. Ainda assim, as convenções e práticas estudadas ao longo deste trabalho, tais como a *Code Conventions for the Java Programming Language* e a *CamelCase*, foram úteis por auxiliar na elaboração da sintaxe da notação `SemTypes` (Bloch, 2008). Os princípios e conceitos apresentados pelos trabalhos estudados permitiram o desenvolvimento de uma notação mais limpa, clara e objetiva.

O aplicativo Checkstyle[2] é uma ferramenta de desenvolvimento Java, disponibilizada sob a licença `LGPLv2` (GNU Lesser General Public License, version 2.1)[3], elaborada para auxiliar desenvolvedores na utilização de

2 Checkstyle 5.6.
http://checkstyle.sourceforge.net/
3 GNU LGPLv2.
http://www.gnu.org/licenses/lgpl-2.1.html

convenções e normas de codificação, chamadas pela ferramenta de estilo de código. O Checkstyle é uma ferramenta de análise estática, pois automatiza o processo de verificação das convenções e normas de codificação de um artefato, grupo de artefatos ou sistema. Sua operação é baseada em verificações padrão, chamadas de `standard checks`, que representam as estruturas sintáticas que a ferramenta é capaz de analisar. Embora sejam restritas em seu escopo de verificação, as propriedades consideradas na análise de cada estrutura sintática podem ser configuradas com grande liberdade pelo usuário, permitindo seu uso para apoiar grande parte dos padrões utilizados pela linguagem Java. Uma breve descrição das verificações que fazem parte do escopo da ferramenta:

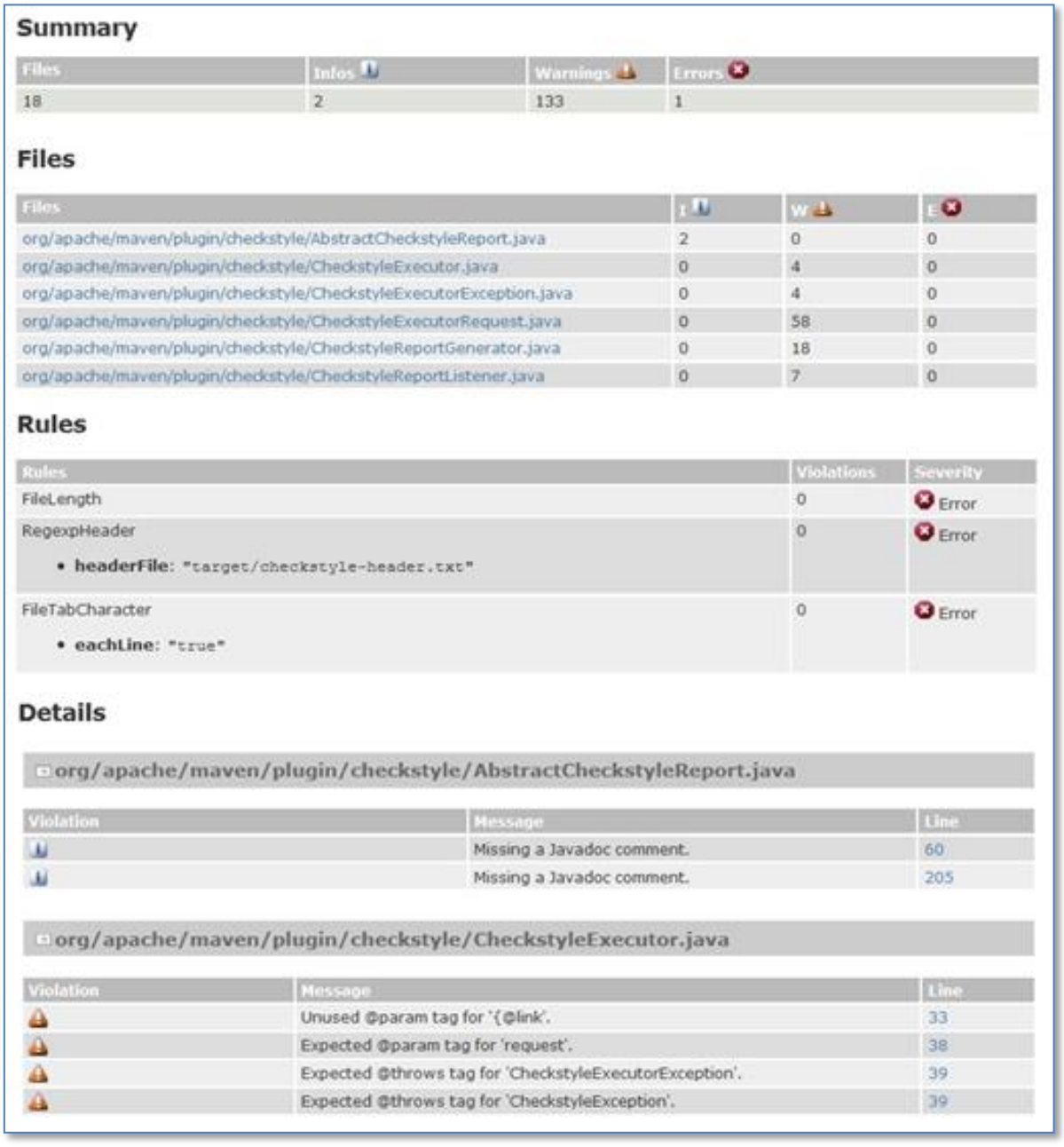
- `Annotations`: Verificação do estilo de código de anotações, funcionalidade disponível a partir de Java 5.
- `Block Checks`: Verificação de estilo de blocos. Analisa estruturas como blocos vazios, blocos aninhados ou recuo.
- `Class Design`: Verificação do estilo de definição de classes. Analisa estruturas como a visibilidade e escopo de membros de dados, características de interfaces estendidas, características dos construtores presentes e potencial de reúso.
- `Coding e Duplicate Code`: Uma das verificações mais amplas de estilo oferecidas pela ferramenta. Possui um grande número de verificações padrão para análise de diversas características de um código-fonte. Analisa estruturas como membros de dados ocultados, declarações vazias, exceções redundantes, números mágicos, trechos de código duplicados e modificações em variáveis de controle de iteração, por exemplo.
- `Headers e Imports`: Verificação do estilo de arquivos de cabeçalho e importados. Analisa estruturas como a formatação do texto cabeçalho e uso de importações estáticas, redundantes ou desnecessárias.

- `Javadoc Comments`: Verificação do estilo do `javadoc` de um artefato. Analisa a estrutura de formatação geral e de todas as `tags javadoc` presentes.
- `Metrics`: Verificação puramente estrutural, e não de estilo, do código. Calcula métricas como complexidade ciclomática, `fan out` e complexidade NPATH.
- `Modifiers`: Verificação de uso dos onze modificadores presentes na especificação da linguagem Java[4].
- `Naming Conventions`: Verificação de formatação dos identificadores de variáveis, tipos e funções. Este `standard check` é composto de um módulo para cada elemento verificado, como por exemplo os módulos `ConstantName`, `LocalVariableName` e `PackageName`. Cada módulo utiliza uma expressão regular para definir os identificadores válidos em seu escopo.
- `Miscellaneous`, `Regular Expressions`, `Size Violations` e `Whitespace`: Verificações gerais de estilo para cobertura de todo o código-fonte. Analisam a formatação e uso dos espaços brancos, a presença de estruturas compatíveis a expressões regulares, e consolidam dados referentes ao número de variáveis, caracteres em uma linha ou métodos em uma classe, por exemplo.

A abrangência dos `standard checks` e a possibilidade de definição e configuração de parâmetros de análise de estruturas são responsáveis pela capacidade de verificação da ferramenta. Sua operação pode ser realizada através do interpretador de comando do sistema ou da criação de uma tarefa na ferramenta `Ant`, muito comum em ambientes de desenvolvimento Java. Para apresentar ao usuário o resultado de uma verificação, o `Checkstyle` produz um relatório que contém quatro seções: (i) o resumo da verificação realizada, indicando o número de arquivos analisados e de advertências e erros encontrados, (ii) uma listagem de

4 Java Language and Virtual Machine specification.
http://docs.oracle.com/javase/specs/

todos os arquivos analisados, (iii) uma listagem com os `standard checks` e configurações utilizadas e (iv) uma listagem para cada arquivo com as advertências e erros encontrados. Abaixo, um exemplo de relatório de verificação produzido pelo `Checkstyle`. A figura 1 exibe um trecho de cada seção:



Summary

Files	Infos	Warnings	Errors
18	2	133	1

Files

Files	I	W	E
org/apache/maven/plugin/checkstyle/AbstractCheckstyleReport.java	2	0	0
org/apache/maven/plugin/checkstyle/CheckstyleExecutor.java	0	4	0
org/apache/maven/plugin/checkstyle/CheckstyleExecutorException.java	0	4	0
org/apache/maven/plugin/checkstyle/CheckstyleExecutorRequest.java	0	58	0
org/apache/maven/plugin/checkstyle/CheckstyleReportGenerator.java	0	18	0
org/apache/maven/plugin/checkstyle/CheckstyleReportListener.java	0	7	0

Rules

Rules	Violations	Severity
FileLength	0	Error
RegexpHeader • **headerFile**: "target/checkstyle-header.txt"	0	Error
FileTabCharacter • **eachLine**: "true"	0	Error

Details

org/apache/maven/plugin/checkstyle/AbstractCheckstyleReport.java

Violation	Message	Line
	Missing a Javadoc comment.	60
	Missing a Javadoc comment.	205

org/apache/maven/plugin/checkstyle/CheckstyleExecutor.java

Violation	Message	Line
	Unused @param tag for '{@link'.	33
	Expected @param tag for 'request'.	38
	Expected @throws tag for 'CheckstyleExecutorException'.	39
	Expected @throws tag for 'CheckstyleException'.	39

Figura 1 – Exemplo de relatório do Checkstyle.

Apesar de todas as funcionalidades presentes na ferramenta, ela não é capaz detectar conflitos semânticos em tipos de dados. Entretanto, existe um `standard check` particularmente interessante para o trabalho proposto neste documento, o `Naming Conventions`. Este `standard check` é capaz de verificar diversos tipos de identificadores, como descrito anteriormente, através de seus módulos. Com a configuração adequada destes módulos, através do fornecimento de expressões regulares que definam corretamente a sintaxe de uma notação, é possível utilizar o `Checkstyle` para realizar a verificação de uso da notação `SemTypes`. Uma hipótese é que, ao automatizarmos o processo de verificação da notação e ao mesmo tempo a detecção de conflitos semânticos, haja um impacto positivo nos resultados dos processos de garantia da qualidade. Ao menos, eliminando a participação humana da verificação de uso da notação, minimiza-se a chance de que um erro em um identificador impeça a ferramenta `VERITAS` de detectar um conflito semântico. Esta hipótese será avaliada no estudo qualitativo realizado, e discutida mais detalhadamente no mesmo capítulo.

Em von Staa (2012), uma abordagem semelhante à proposta do presente trabalho é apresentada. Nesta abordagem, os tipos semânticos utilizados no processo de desenvolvimento são previamente construídos e fornecidos aos desenvolvedores através de uma tabela de tipos. Assim como os arquivos de definição semântica utilizados no presente trabalho, as tabelas de tipos permitem que as definições sejam utilizadas em diversos projetos e organizações. A principal diferença é que, na abordagem apresentada em von Staa (2012), os tipos fornecidos são tipos computacionais, ou seja, são reconhecidos pelo compilador e usufruem da mesma infraestrutura que os tipos computacionais. Para que isto seja possível, a criação de um tipo semântico requer que este seja construído na linguagem de programação utilizada, e não apenas definido através de descritores estruturados, como proposto no presente trabalho. Desta forma, artefatos desenvolvidos com base nas tabelas de tipos não são reconhecidos por compiladores que não tenham acesso às tabelas, limitando a portabilidade do código. Nas duas abordagens existe a necessidade de adaptação dos desenvolvedores aos novos tipos ou identificadores de tipos, seja no desenvolvimento, teste ou inspeção. A maior vantagem desta abordagem sobre a solução proposta no presente trabalho é a ausência de uma notação para nomeação

de identificadores. Notações podem impactar negativamente na legibilidade e manutenibilidade de um artefato, e no caso particular da SemTypes tipos complexos ou compostos podem induzir à criação de identificadores igualmente complexos. Ainda, existe a necessidade de se manter a consistência e a coerência do conjunto de tipos semânticos existentes em um projeto ou organização, e a estrutura baseada em strings da SemTypes, ao contrário da estrutura baseada em tipos computacionais das tabelas de tipos, pode dificultar neste trabalho.

O FindBugs™ 2 é uma ferramenta de análise estática utilizada para detectar defeitos em artefatos de código-fonte escritos com a linguagem de programação Java. Assim como o Checkstyle, o FindBugs 2 é um aplicativo livre distribuído pela licença LGPLv2, amplamente utilizado pela comunidade Java e considerado eficiente e eficaz em sua operação (Grindstaff, 2006, 2008).

A ferramenta funciona inspecionando bytecodes em busca de padrões de defeitos. Bytecodes são instruções concebidas para serem executadas por um interpretador, e não são legíveis do ponto de vista do desenvolvedor (Aarniala, 2005). Compiladores Java transformam código-fonte Java, como classes e arquivos JAR, em bytecodes Java, que são interpretados pela máquina virtual Java em tempo de execução. Desta forma, o FindBugs 2 atua exclusivamente analisando software escrito na linguagem de programação Java.

Os padrões de defeitos que a ferramenta analisa são divididos em nove categorias de defeitos, são elas:

- Malicious code vulnerability: Indica trechos de código vulneráveis a serem maliciosamente alterados por outro trecho de código.
- Dodgy: Indica trechos que código que podem levar um desenvolvedor a cometer erros.
- Bad practice: Indica trechos de código que violam práticas recomendadas de desenvolvimento.
- Correctness – Indica trechos de código que podem obter resultados diferentes da provável intenção do desenvolvedor.

- `Internationalization`: Indica trechos de código que podem inibir ou impossibilitar uso de caracteres de internacionalização.
- `Performance`: Indica trechos de código que poderiam ser escritos de forma diferente para melhorar o desempenho.
- `Security`: Indica trechos de código que podem levar a falhas de segurança.
- `Multithreaded correctness`: Indica trechos de código que podem causar problemas em ambientes de programação concorrente.
- `Experimental`: Indica trechos de código que podem afetar negativamente rotinas de limpeza, tais como limpeza de memória ou cache.

Cada uma destas nove categorias possui filtros configuráveis que possibilitam ao desenvolvedor determinar o funcionamento de alguns aspectos da ferramenta. Estes aspectos incluem quais classes devem ser investigadas, quais padrões de defeito devem ser ignorados e, principalmente, a rigidez da análise, que impacta diretamente na quantidade de falsos positivos detectada em uma verificação. Relatos de uso e pesquisas publicadas, como em Rutar (2004), indicam que é comum a ocorrência de falsos positivos nos relatórios de bugs gerados pelo `FindBugs`. Ao final de uma execução da ferramenta, sua interface gráfica permite que o usuário navegue pelos prováveis defeitos indicados – a figura 2 exibe esta interface. O `FindBugs 2` também pode ser executado como uma tarefa `Ant`[5], assim como o `Checkstyle`.

5 Apache Ant
http://ant.apache.org

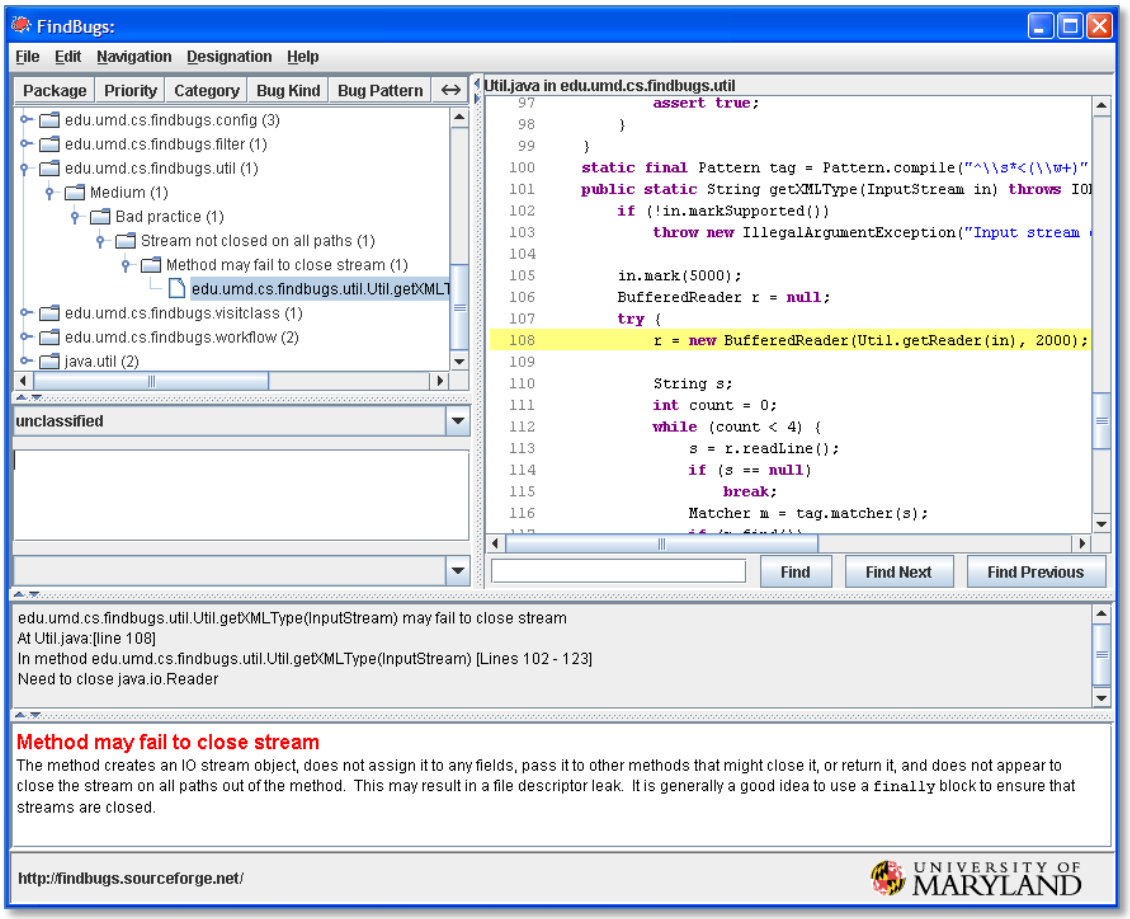


Figura 2 – Exemplo de relatório do FindBugs 2.

É importante destacar que, embora haja controle sobre os critérios utilizados na detecção de um defeito potencial e sua inclusão no relatório gerado pela ferramenta, não se pode alterar a heurística de detecção. Sendo assim, não é possível configurar filtros para que a ferramenta se torne capaz de detectar conflitos semânticos (Sandcastle Group, 2009) [6].

Apesar das limitações, o `FindBugs 2` teve grande influência no desenvolvimento da ferramenta `VERITAS`. A estrutura dos arquivos de definição semântica utilizados pela ferramenta `VERITAS` seguem a mesma estrutura dos filtros do `FindBugs 2`, assim como as regras de composição de definições. O formato `XML` dos filtros do `FindBugs 2` também foi escolhido por facilitar o intercâmbio das definições entre sistemas e projetos. A principal influência conceitual do `FindBugs 2` na ferramenta `VERITAS`, entretanto, é a existência

6 An Evaluation of FindBugs.
http://www.cs.cmu.edu/~aldrich/courses/654/tools/Sandcastle-FindBugs-2009.pdf

de categorias de defeitos. Ao tratar cada tipo de identificador – variáveis, tipos e funções – como uma categoria diferente, a forma de operação das duas ferramentas se torna similar. A `VERITAS`, assim como o `FindBugs 2`, processa cada categoria independentemente e com base em definições organizadas em arquivos `XML`. A diferença entre as ferramentas se resume ao tipo dos grupos analisados e a sintaxe dos arquivos `XML` de definição.

Uma das áreas de conhecimento da engenharia de software mais relevantes para este trabalho é a inspeção de software. A inspeção de software é o exame sistemático, realizado por humanos, de um artefato do desenvolvimento de software, tipicamente especificações de requisitos, arquiteturas, modelos e código-fonte. Alguns estudos, como em Brown (1996), apontam a inspeção de software como uma das melhores práticas da indústria. O objetivo das inspeções é a detecção prematura de defeitos, idealmente no momento em que o custo de correção e o risco da ocorrência de falhas com elevados danos é baixo. Inspeções são normalmente realizadas após a fase de construção do artefato, embora em alguns casos específicos possa ser realizada após a fase de concepção ou elaboração (Fagan, 1976).

Existem diversas metodologias de inspeção, e são divididas em dois grandes grupos: inspeções formais e inspeções informais. De forma geral, as inspeções formais demandam mais recursos e tempo dos participantes, mas possuem taxas melhores de detecção de defeito que as inspeções informais (Wiegers, 2002). Este método de garantia de qualidade de software é capaz de detectar defeitos como vazamento de memória, condições de corrida e, principalmente, conflitos semânticos. As inspeções estão entre os métodos mais adequados para detectar quebras do contrato semântico, como detalhado em Fagan (1976) e Wiegers (2002), mas sua natureza manual e dependente de atuação humana afeta o desempenho, pois requer muita atenção para que seja bem sucedida.

O presente trabalho se propõe a encontrar uma forma adequada de inspecionar artefatos de software em busca de conflitos semânticos e, então, automatizar esta tarefa. Em outras palavras, utilizar uma abordagem humana adequada para inspeção de conflitos semânticos em artefatos de código-fonte e transformá-la em uma ferramenta de análise estática.

2.2 Abordagens que Previnem o Risco

O termo métodos formais se refere a um amplo conjunto de técnicas, ferramentas e abstrações matemáticas – principalmente do campo da lógica – utilizadas na especificação, desenvolvimento e verificação de artefatos. O uso de métodos formais no desenvolvimento de software é uma prática antiga, contando com um grande número de trabalhos publicados na literatura. É possível utilizar níveis incrementais de rigor, com base nas demandas e recursos de um projeto.

Métodos formais são considerados métodos de análise estática, pois a realização da análise não requer a execução do programa. O pequeno número de profissionais com competência necessária para a utilização de métodos formais, aliado ao alto custo em recursos humanos, torna seu uso proibitivo em grande parte dos projetos, principalmente em sistemas onde o risco ou seu impacto são pequenos (Fleck, 2006)

Podemos dividir os métodos formais[7] em quatro grandes grupos: especificações formais, provas formais, verificação de modelos e abstrações (Lundqvist, 2002). Especificações formais são utilizadas para descrever o comportamento e as propriedades de um sistema através de uma linguagem de especificação formal. O objetivo é traduzir descrições não matemáticas, como tabelas, diagramas e texto, em uma linguagem que permita deduções formais sobre a especificação. Existe um grande número de linguagens de especificação formal disponíveis na literatura, com características adequadas para determinados usos.

Provas formais são utilizadas para determinar de forma inequívoca a validade de uma propriedade do sistema analisado, permitindo eliminar a ambiguidade e subjetividade acerca desta propriedade. Elas são construídas passo a passo, cada um justificado por um conjunto de regras.

A verificação de modelos consiste em representar o artefato ou sistema – através, por exemplo, de uma máquina de estados finitos, de um diagrama de classes, ou de uma linguagem formal – e determinar se a representação satisfaz

7 Formal Methods.
http://web.mit.edu/16.35/www/lecturenotes/FormalMethods .pdf

alguns requisitos, muitas vezes expressados como fórmulas de um sistema dedutivo. É uma abordagem fortemente operacional, e a verificação é realizada através da exploração de todos os estados possíveis alcançados a partir da representação do artefato ou sistema. Assim como na análise estática, não é necessária a execução do sistema durante a verificação, pois apenas suas propriedades são avaliadas.

Abstrações são usadas para simplificar o entendimento e a modelagem de propriedades de um artefato ou sistema analisado. Expressam apenas as questões centrais e mais relevantes, deixando para um segundo plano os detalhes que pouco contribuem para a análise. Permitem que outros grupos de técnicas e ferramentas sejam utilizados de forma mais eficiente.

A abordagem proposta neste trabalho não requer conhecimento especializado, ao contrário dos métodos formais, e todo desenvolvedor pode utilizar a notação e a ferramenta com pouco treinamento. Ainda, uma vez que se tenha definido as regras semânticas, o custo em recursos computacionais ou o tempo de desenvolvimento não é significativo, já que se dilui ao longo das verificações executadas. Como mencionado previamente, um dos objetivos do experimento é avaliar se a abordagem proposta é eficiente e eficaz. Se isto for comprovado pelos resultados obtidos, a solução se torna particularmente interessante quando comparada a utilização de métodos formais, considerando sua complexidade e custo.

Linguagens de domínio específico, ou DSL (Domain-Specific Language), são linguagens de programação ou linguagens executáveis de especificação que oferecem, através de abstrações e notações apropriadas, poder de expressão restrito a um domínio específico de problemas (van Deursen et al., 2000). Este é um conceito antigo, e linguagens de propósito específico, modelagem ou especificação existem desde o início da era digital.

Algumas linguagens de programação atualmente consideradas linguagens de propósito geral, ou GPL (General Purpose Languages), evoluíram gradativamente a partir de linguagens de domínio específico, notavelmente (i) COBOL, originalmente uma linguagem de processamento de negócios, (ii)

FORTRAN, concebida para computação numérica e (ii) LISP, desenvolvida para possibilitar processamento simbólico.

Linguagens de propósito geral oferecem diversas capacidades, como tratamento de diferentes tipos de dados, controles e estruturas de abstração. Estas capacidades são úteis, mas dificultam seu aprendizado e uso.

As linguagens específicas de domínio tipicamente são pequenas, oferecendo apenas um conjunto restrito de notações e abstrações, e por isso também são conhecidas como micro-languages ou little languages. Sua estrutura é usualmente declarativa, o que possibilita seu uso como linguagens de especificação, e as operações disponíveis empacotam conhecimento do domínio de forma a facilitar seu reuso. Isto ocorre em função da natureza limitada do escopo (Fowler, 2010). O uso de DSLs permite que soluções sejam expressas no mesmo idioma e nível de abstração do domínio, que por sua vez permite que especialistas deste domínio desenvolvam e validem aplicações. Alguns estudos como van Deursen e Klint (1998) e Herndon e Berzins (1988) apontam que a produtividade, confiabilidade, manutenibilidade portabilidade de aplicações escritas com DSLs é superior, e que a absorção de conhecimento sobre o domínio e o sistema é mais rápida, assim como sua retenção. Os mesmos estudos também concluem que a comunicação se torna mais eficiente e eficaz. Podemos dividir as DSLs em dois grupos:

- DSL Externa: A DSL é separada da GPL utilizada no desenvolvimento da aplicação. Um *parser* na linguagem da aplicação é utilizado para processar um script contendo o código escrito na linguagem específica de domínio.
- DSL Interna: A DSL é um subconjunto da GPL utilizada no desenvolvimento da aplicação. O subconjunto de características herdadas da GPL é utilizado de uma forma específica para tratar de um pequeno aspecto do domínio. Esta é uma abordagem comum em algumas linguagens de programação de propósito geral tais como LISP e Ruby.

Em uma aplicação desenvolvida com o auxilio de uma DSL, é possível criar e utilizar tipos específicos do domínio desta aplicação. Assim, ao substituirmos os

tipos físicos e nativos das GPL por estes tipos fortemente vinculados ao domínio e carregados de semântica, podemos prevenir quebras de contrato semântico. Entretanto, esta abordagem possui algumas limitações, tais como: (i) os recursos necessários para seu desenvolvimento e manutenção, (ii) o custo de treinamento dos especialistas de domínio, (iii) a necessidade de compiladores especiais para gerar as aplicações, chamados de application generators e (iv) o impacto sobre a equipe de desenvolvimento. Os desenvolvedores são obrigados a lidar com uma linguagem nova e, provavelmente, esta linguagem possui menos recursos e referências. Esta é a maior diferença entre a solução de conflitos semânticos utilizando DSLs e a solução proposta neste trabalho: através da notação SemTypes o desenvolvedor está livre para trabalhar com uma GPL com a qual esteja familiarizado e seja adequada ao ambiente de desenvolvimento, processos e ferramentas disponíveis.

3 A Ferramenta VERITAS e a Notação SemTypes

Este capítulo apresenta a solução desenvolvida no decorrer do presente trabalho. Esta solução é composta de uma notação para especificação de contratos semânticos, nomeada `SemTypes`, e uma ferramenta de análise estática, nomeada `VERITAS`, que investiga código-fonte em busca de violações de um contrato semântico estabelecido. Entretanto, a solução desenvolvida possui uma limitação: requer dados de entrada em um formato complexo para sua operação. Uma segunda ferramenta, `ANTLR`, prepara e fornece a entrada adequada para operação da `VERITAS`. A `ANTLR` é discutida detalhadamente neste capítulo, pois seu entendimento é necessário para compreensão do funcionamento da ferramenta `VERITAS`. A seção 3.1 discute a notação SemTypes, detalhando sua estrutura e sintaxe. A seção 3.2 consiste em uma descrição completa da ferramenta `ANTLR` e seu relacionamento com a solução desenvolvida. Por último, a seção 3.3 apresenta a ferramenta de análise estática `VERITAS`.

3.1 A Notação SemTypes

A notação SemTypes foi desenvolvida para viabilizar a definição explícita de contratos semânticos de tipos utilizados em um sistema de informação. Como discutido no capítulo 1, desenvolvedores de software implicitamente atribuem significado aos tipos de dados declarados e utilizados na construção de um artefato. Entretanto, este significado tende a se tornar um conhecimento tácito daqueles que participaram do processo de elaboração dos tipos, que nem sempre é compreendido por todos os desenvolvedores que trabalharão direta ou indiretamente com os tipos de dados ao longo da vida útil do sistema. Isto ocorre pois a especificação do significado dos tipos não é sistematicamente registrada em nenhum dos artefatos usuais do processo de desenvolvimento de software, tais como casos de uso, testes unitários e glossários de termos de negócio, por exemplo.

A linguagem de marcação e formato XML é utilizado pela notação SemTypes, sendo selecionado em função características importantes: (i) seu formato é compreensível por humanos e máquinas, (ii) permite que o usuário defina seu próprio vocabulário, (iii) sua representação textual de dados e

estruturas de dados é amplamente suportada por linguagens de programação através do Unicode[8] e, principalmente, (iv) provê uma estrutura hierárquica bem definida ao seu conteúdo.

A estrutura da notação `SemTypes` é baseada em elementos de marcação, ou `tags`. Cada elemento possui uma função única na definição de um contrato semântico. Estes elementos contêm atributos, que especializam sua função através de parâmetros configurados pelo usuário, e fazem parte de uma estrutura hierárquica bem definida. A estrutura deve ser respeitada para que o conjunto de elementos de marcação em um determinado documento seja considerado válido.

Para estabelecer contratos semânticos verificáveis, o desenvolvedor utiliza um novo artefato, proposto pela solução: os arquivos de definição semântica. Estes arquivos devem ser elaborados e mantidos pela equipe de desenvolvedores, que tem a responsabilidade de representar o conhecimento sobre os tipos semânticos na forma textual definida pela notação. Os arquivos de definição semântica podem pertencer a três distintos espaços de nomes:

- Extensão: arquivos que contêm definições específicas de um negócio ou modelo, e atuam estendendo as definições de um projeto. São elaborados para facilitar o reuso e normalmente representam tipos semânticos bem conhecidos, tais como o sistema métrico internacional ou um modelo de entidades de uma instituição de ensino, por exemplo.
- Global: arquivos que contêm as definições semânticas que devem ser utilizadas em todos os projetos
- Local: arquivos que contêm as definições semânticas específicas de um projeto.

A ferramenta `VERITAS` é capaz de compor diferentes arquivos de definição semântica para utilizar em uma verificação, com base em três regras simples:

8 Unicode Consortium.
http://www.unicode.org/

- Regra 1: um tipo semântico definido em um arquivo global e um arquivo local possui o significado especificado no arquivo global.
- Regra 2: um tipo semântico não pode ser definido em mais de um arquivo global.
- Regra 3: um tipo semântico definido em um arquivo local e um arquivo de extensão possui o significado especificado no arquivo local.

A capacidade de expressão da notação é o fator mais determinante na eficácia da verificação de um artefato. A ferramenta VERITAS apenas identifica sintaticamente quais estruturas presentes nos arquivos de definição semântica não foram respeitadas no artefato. O significado e o conjunto destas estruturas definem o que é considerado uma violação das propriedades semânticas de um tipo, sendo a forma direta do desenvolvedor guiar a ferramenta para detectar defeitos no artefato. Resta então definir quais são os elementos da notação SemTypes e sua capacidade de expressão:

- `semanticTypes`: Elemento que marca o início e o fim de um arquivo de definição semântica. Não possui atributos e não define o significado de nenhuma entidade.
- `definition`: Elemento que define o tipo de arquivo de definição semântica. Possui um único atributo, `type`, que assume um valor de acordo com o espaço de nomes em que o arquivo é utilizado: `extension`, `global` ou `local`.
- `entity`: Elemento básico da notação, utilizado como bloco de construção das definições semânticas. Representa um tipo semântico. Pode ser de três tipos, representados pelo atributo `type`: `simple`, `defined` ou `function`. Um tipo declarado com um elemento `simple` não possui qualquer regra específica de comportamento, mas apenas pode ser utilizado em operações onde os operandos sejam do seu tipo. Um tipo declarado com um elemento `defined` possui regras específicas de comportamento. Os tipos declarados como `function` representam métodos e

procedimentos. O atributo `name` permite que o desenvolvedor defina um nome para o tipo semântico em questão. O atributo `prefix` é utilizado para definir o radical que representa o tipo semântico no código-fonte.

- `behaviour`: Elemento, filho de um elemento `entity`, utilizado para agrupar os comportamentos do tipo semântico do elemento pai.
- `legacyPrefixes`: Elemento, filho de um elemento `entity`, que contém uma lista de elementos do tipo `legacyPrefix`.
- `legacyPrefix`: Elemento que define um radical que deve ser interpretado como o tipo do elemento pai do presente elemento. Utilizado para identificadores de código legado, emprestando o comportamento de um tipo semântico definido às variáveis já criadas.

O elemento `rule`, filho do elemento `behaviour`, é o mais complexo da notação SemTypes e será explicado separadamente. Seu primeiro atributo, `id`, define um identificador da regra para facilitar a depuração do artefato com base no relatório de verificação produzido pela ferramenta `VERITAS`. O atributo `key` define o significado de uma regra semântica, e é auxiliado pelo atributo `value` que define quantitativamente as características da regra semântica. O atributo `key` pode receber:

- `from`: indica que a regra define um limite inferior para tipos primitivos ou um tamanho mínimo no caso de sequências de caracteres. O Atributo `value` define este limite.
- `to`: indica que a regra define um limite superior para tipos numéricos ou um tamanho máximo no caso de sequências de caracteres. O Atributo `value` define este limite.
- `operation`: indica que a regra define uma operação que não pode ser executada sobre o tipo semântico. O atributo `value` define esta operação. As operações reconhecidas pela ferramenta são as operações aritméticas e as booleanas. Nas sequências de caracteres o símbolo `+` é interpretado como uma concatenação. É possível

especificar entidades com permissão para a realização da operação. O atributo `allowedEntity` é utilizado para este fim, e deve conter o nome do tipo permitido.

- `parameter`: indica que a regra define um tipo semântico que deve ser utilizado como parâmetro da função. O Atributo `value` define este tipo e deve conter seu nome. Múltiplos elementos `rule` podem ser definidos para os casos de diferentes tipos de parâmetros.
- `return`: indica que a regra define um tipo semântico que deve ser utilizado como tipo de retorno da função. O Atributo `value` define este tipo e deve conter seu nome.

Abaixo, a figura 3 ilustra um arquivo de definição semântica. Este arquivo define cinco entidades e especifica suas regras semânticas. O primeiro elemento encontrado é o `semanticTypes`, que marca o início de um arquivo de definição semântica. Em seguida, temos o elemento `definition`, que marca com seu único atributo `type` que o arquivo de definição semântica em questão é do tipo `global` – sendo as outras opções disponíveis `extension` ou `local`. O arquivo segue com uma sequência de cinco declarações de entidade, através dos elementos `entity`, e se encerra com o fechamento dos elementos `definition` e `semanticTypes`, nesta ordem. Cada elemento `entity` define uma entidade semântica para a qual o desenvolvedor deseja especificar um contrato semântico, e temos no exemplo uma entidade do tipo `simple`, três do tipo `defined` e uma do tipo `function`. Quatro das entidades do exemplo utilizam o elemento `behaviour`, que agrupa todas as regras semânticas de uma entidade – representadas pelo elemento `rule`. A única entidade do tipo `simple`, nomeada *Currency*, define um tipo semântico identificado pelo prefixo *cur*. Esta entidade, como todas do tipo `simple`, não possui regras específicas e apenas pode participar de operações e atribuições que envolvam outras entidades do seu mesmo tipo. A única entidade do tipo `function` define uma função, nomeada *PayrollCalculator* e indicada pelo prefixo *prcalc*. Este tipo semântico possui duas regras: (i) a regra *BR09*, que define que apenas argumentos do tipo semântico *Integer* serão aceitos, e (ii) a regra *BR10* que define que o retorno da função é do tipo semântico *Payroll*. Restam então as três entidades do tipo `defined`,

nomeadas *Integer*, *Payroll* e *Address*. A entidade *Integer*, prefixada por *int*, define um tipo semântico que possui duas regras: (i) a regra *BR01*, que define um limite inferior para variáveis deste tipo, e (ii) a regra *BR02*, que define um limite superior. A entidade *Integer* também define dois prefixos legados, nomeados *inter* e *integ*, que especificam quais prefixos de código legados identificam variáveis que devem se comportar como um *Integer*. Cada prefixo legado é identificado pelo elemento `legacyPrefix` e todos são agrupados em um elemento `legacyPrefixes`. A entidade *Payroll*, prefixada por *prol*, define cinco regras semânticas. A primeira regra, a *BR03*, define um limite inferior da mesma forma que a regra *BR01* da entidade *Integer*. As demais quatro regras, de *BR05* a *BR08*, definem as operações válidas e quais entidades podem participar destas operações. Neste exemplo, as regras semânticas indicam que apenas as operações aritméticas básicas são permitidas, e somente com outras entidades do tipo *Integer*. Por último, temos a entidade *Address*, prefixada por *adr*. Esta entidade possui duas regras semânticas que definem o limite inferior e superior para variáveis deste tipo semântico, mas se comportam de forma diferente das regras *BR01*, *BR02* e *BR03*. No caso particular da entidade *Address*, quando aplicada a variáveis do tipo computacional `string`, estes limites inferior e superior determinam o tamanho mínimo e máximo da cadeia de caracteres.

```xml
<semanticTypes>
    <definition type="global">

        <entity type="simple" name="Currency" prefix="cur"/>

        <entity type="defined" name="Integer" prefix="int">
            <legacyPrefixes>
                <legacyPrefix prefix="inter" />
                <legacyPrefix prefix="integ" />
            </legacyPrefixes>
            <behaviour>
                <rule key="from" value="1" id="BR01" />
                <rule key="to" value="100" id="BR02" />
            </behaviour>
        </entity>

        <entity type="defined" name="Payroll" prefix="proll">
            <behaviour>
                <rule key="from" value="0.0" id="BR03" />
                <rule key="operation" value="-" id="BR05" allowedEntity="Integer" />
                <rule key="operation" value="/" id="BR06" allowedEntity="Integer" />
                <rule key="operation" value="*" id="BR07" allowedEntity="Integer" />
                <rule key="operation" value="%" id="BR08" allowedEntity="Integer" />
            </behaviour>
        </entity>

        <entity type="function" name="PayrollCalculator" prefix="prcalc">
            <behaviour>
                <rule key="parameter" value="Integer" id="BR09" />
                <rule key="return" value="Payroll" id="BR10" />
            </behaviour>
        </entity>

        <entity type="defined" name="Address" prefix="adr">
            <behaviour>
                <rule key="from" value="1" id="BR11"/>
                <rule key="to" value="200" id="BR12"/>
            </behaviour>
        </entity>

    </definition>
</semanticTypes>
```

Figura 3 – Exemplo de arquivo global de definição semântica.

3.2 Exemplo de Utilização da Notação SemTypes

Um cenário provável de uso da notação SemTypes e da ferramenta VERITAS é o de desenvolvimento `web`, como descrito previamente no primeiro capítulo. Aplicações executadas do lado servidor, para tratar requisições enviadas das formas mais diversas por clientes, podem se tornar complexas e demandar o processamento de um grande número de entidades. Ainda, em alguns tipos de aplicações `web` tais como as de `home banking` existe a necessidade de prover um alto grau de segurança ao usuário. Em um caso comum de preenchimento de um formulário pelo usuário para posterior utilização destes dados, independente do fato deste formulário apresentar muitos campos repletos de dados confidenciais ou apenas um campo de informação textual, é possível que um usuário malicioso introduza sequências de caracteres potencialmente perigosas. Exemplos de ameaças são ataques de injeção de `SQL` ou `Cross Site Scripting` (Cert, 2000). Um exemplo de pseudocódigo vulnerável seria:

```
// Primeiro obtemos o nome do usuário
userData = webRequestForm.getParameter(“userName”);
// Em seguida, utilizamos este nome no processamento de uma
// regra de negócio
businessRule.do(userData);
```

Este trecho de código permite que sequências de caracteres fornecidas por um usuário sejam utilizadas pela aplicação do lado servidor sem tratamento prévio, expondo assim aos riscos discutidos no parágrafo anterior. Para evitar tais riscos o desenvolvedor deve tratar os dados fornecidos:

```
// Primeiro obtemos o nome do usuário
userData = webRequestForm.getParameter("userName");
// Em seguida tratamos os dados obtidos
userData = Data.encode(userData);
// Por último, utilizamos este nome no processamento de uma
// regra de negócio
businessRule.do(userData);
```

Desta forma, comumente utilizada na indústria, protegemos o sistema de possíveis dados maliciosos através da conversão de todos os dados fornecidos antes de sua utilização (Levantamento, 2013). Entretanto, esta solução não é satisfatória, pois seu sucesso depende inteiramente de um processo de desenvolvimento sem falhas humanas. Como o tipo computacional dos dados em ambos os formatos – seguro ou vulnerável – é o mesmo, os métodos tradicionais mais comuns irão falhar na detecção deste tipo de defeitos. Apenas inspeções e testes unitários elaborados especificamente para ameaças de segurança serão capazes de detectar tais defeitos. Ainda, o custo de elaboração dos testes para cobertura de todas as possíveis vulnerabilidades e a eficiência de detecção de defeitos em inspeções continuariam sendo graves problemas desta abordagem. É possível solucionar este problema utilizando a solução proposta neste trabalho, com a criação de um arquivo de definição semântica com regras para evitar as operações e atribuições indesejadas. Abaixo, a figura 4 ilustra este arquivo:

```xml
<semanticTypes>
    <definition type="global">

        <entity type="simple" name="UnsecureString" prefix="unsec"/>

        <entity type="simple" name="SecureString" prefix="sec"/>

        <entity type="function" name="SecureEncoding" prefix="senc">
            <behaviour>
                <rule key="parameter" value="UnsecureString" id="BR01" />
                <rule key="return" value="SecureString" id="BR02" />
            </behaviour>
        </entity>

        <entity type="function" name="VulnerableBusinessProcess" prefix="vul">
            <behaviour>
                <rule key="parameter" value="SecureString" id="BR03" />
            </behaviour>
        </entity>

    </definition>
</semanticTypes>
```

Figura 4 – Exemplo de uso da notação SemTypes.

O arquivo de definição semântica proposto é do tipo `global` e declara quatro entidades: duas do tipo `simple` e duas do tipo `function`. As entidades do tipo `simple` representam os dados seguros e não-seguros tratados pela aplicação, respectivamente *SecureString* e *UnsecureString*. A escolha deste tipo para ambas as entidades garante que elas não sejam utilizadas em nenhum tipo de operação ou atribuição, com exceção daquelas que sejam explicitamente permitidas em elementos do tipo `function`. Temos então duas entidades do tipo `function`: (i) uma representando a função de conversão de dados não-seguros em dados seguros e (ii) outra representando a regra de negócio que utiliza os dados seguros. A primeira entidade do tipo `function`, a *SecureEncoding*, estabelece um contrato semântico através das regras *BR01* e *BR02* que define que apenas dados não-seguros serão recebidos como parâmetro e apenas dados

seguros serão retornados após sua execução. A segunda entidade do tipo `function`, a *VulnerableBusinessProcess*, estabelece um contrato semântico através da regra *BR03* que define que apenas dados seguros serão aceitos como parâmetro. O pseudocódigo compatível com este arquivo de definição semântica seria:

```
// Primeiro obtemos o nome do usuário
unsecUserData = webRequestForm.getParameter("userName");
// Em seguida tratamos os dados obtidos
secUserData = Data.sencEncode(unsecUserData);
// Por último, utilizamos este nome no processamento de uma
// regra de negócio
businessRule.vulDo(secUserData);
```

Agora, o pseudocódigo abaixo simula um erro em alguma fase do processo de desenvolvimento da equipe, com falta de conversão dos dados não-seguros em dados seguros:

```
// Primeiro obtemos o nome do usuário
userData = webRequestForm.getParameter("userName");
// Por último, utilizamos este nome no processamento de uma
// regra de negócio
businessRule.vulDo(userData);
```

Neste caso, existe uma clara violação do contrato semântico, onde a função vulnerável é invocada com parâmetros não-seguros. O relatório de verificação da ferramenta VERITAS indicaria esta violação para análise da equipe. Este relatório é discutido na seção 3.4, e um exemplo de resultado de verificação é fornecido.

3.3 A Ferramenta ANTLR

A ferramenta ANTLR é um gerador de analisadores sintáticos que pode ser utilizada para o desenvolvimento de interpretadores, compiladores e outros tipos de tradutores de linguagens de programação (Par, 2007). Seus usos mais frequentes estão relacionados às DSLs, especialmente na construção dos tradutores e interpretadores necessários ao seu funcionamento, descrito no capítulo anterior. Em linhas gerais, a ferramenta é capaz de processar sua entrada – código fonte na linguagem de programação utilizada – e criar como saída uma AST equivalente ao código fornecido. Em muitos casos a saída desejada não é a AST fornecida após a execução do ANTLR. Nestes casos, é necessário desenvolver um aplicativo para processar a AST gerada, e o ANTLR fornece bibliotecas e tabelas de símbolos que são capazes de tratar as estruturas de dados específicas utilizadas para a construção e classificação dos elementos desta árvore. Os aplicativos que processam ASTs são denominados na literatura do ANTLR de caminhadores de ASTs, ou `tree walkers`. Abaixo, uma imagem ilustrando a arquitetura de alto nível da ferramenta:

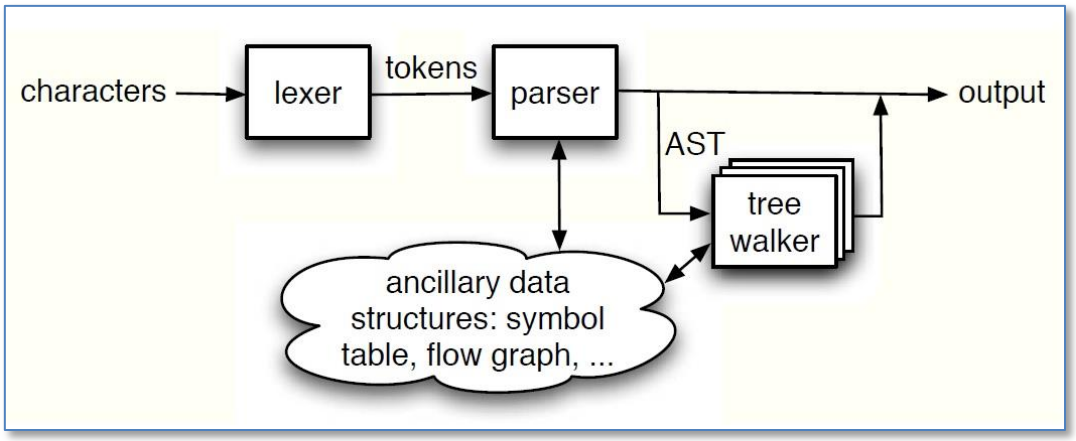


Figura 5 – Arquitetura de alto nível do ANTLR. Imagem extraída de Par, 2007.

Os dois principais módulos que compõem o ANTLR são o analisador léxico e o analisador sintático, e sua operação conjunta e coordenada define os estágios do fluxo de trabalho ferramenta. Este fluxo de trabalho é dividido em dois estágios sequenciais: a análise léxica e a análise sintática. Os estágios são

sequenciais porque o analisador sintático utiliza e depende da saída do analisador léxico como entrada. Para que as operações de ambos os analisadores sejam realizadas, a ANTLR precisa ter acesso a uma gramática formal de referência, para ser utilizada com guia nos dois estágios do fluxo de trabalho.

O analisador léxico é responsável por analisar os dados recebidos de entrada e agrupá-los em símbolos léxicos, chamados de lexical tokens, que são compreensíveis para o analisador sintático. Um símbolo léxico consiste em ao menos duas informações: o tipo de símbolo léxico e caracteres válidos que constituem uma unidade léxica. Os tipos de símbolo léxico existentes para classificação, tais como identificador, inteiro ou função, dependem da gramática utilizada. O processo de agrupar caracteres válidos em uma linguagem para formar os símbolos léxicos é chamado de análise léxica, e possui duas fases. Na primeira fase um leitor processa toda a entrada, identificando as estruturas léxicas e observando quais são as sequências de caracteres válidos e suas fronteiras. Assim, a entrada pode ser dividida em elementos válidos na gramática desejada. O leitor do ANTLR é uma máquina de estados finitos que reconhece todas as sequências válidas da gramática definida como referência. A segunda fase da análise léxica é a construção dos símbolos léxicos, onde ocorre a classificação com base nos tipos definidos pela gramática.

O analisador sintático desempenha o segundo e último estágio do fluxo de trabalho da ferramenta. Após receber os símbolos léxicos, o analisador sintático inicia o trabalho de processamento das estruturas da entrada e suas frases componentes. A estrutura de dados chamada de AST é gerada com base nesta análise, representando toda a informação reconhecida pelo analisador. Abaixo, a imagem ilustra o fluxo de trabalho da ferramenta:

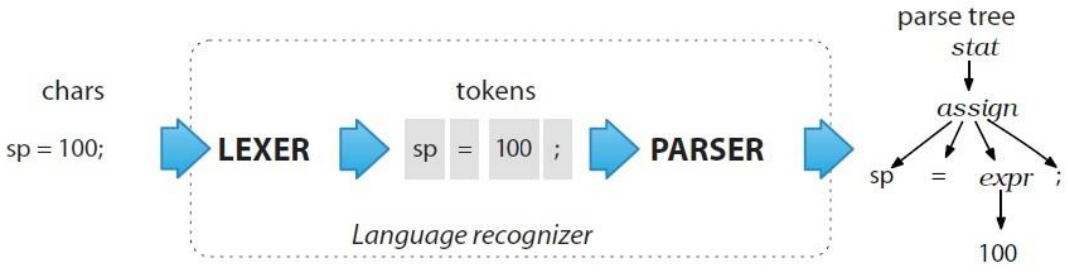


Figura 6 – Arquitetura de alto nível do ANTLR. Imagem extraída de Par, 2013.

A ferramenta ANTLR desempenha um importante papel da solução proposta neste trabalho fornecendo uma árvore sintática abstrata do código fonte verificado e uma biblioteca de funcionalidades de operação sobre esta árvore. Uma gramática formal de Java foi pesquisada para servir de referência a ferramenta em sua operação. A descoberta de uma gramática adequada foi uma tarefa longa, que contou com um processo meticuloso de avaliação. Este é um ponto importante para a expansão da ferramenta: para ser capaz de verificar um artefato de código fonte em outra linguagem de programação em busca de violações do contrato semântico, primeiro se torna necessário encontrar uma gramática formal adequada da nova linguagem. Na seção 5.2 esta questão e suas implicações são discutidas mais detalhadamente.

3.4 A Ferramenta VERITAS

A ferramenta VERITAS é o principal objeto de estudo do presente trabalho. De forma resumida, é um analisador estático de código-fonte Java com funcionamento baseado no conceito de contratos semânticos, discutido previamente no capítulo 1. A ferramenta trabalha verificando um artefato em busca de possíveis violações do contrato semântico estabelecido, e produz um relatório com o resultado desta verificação para ser analisado pela equipe de desenvolvimento. A solução desenvolvida é composta de dois módulos: o gerenciador de arquivos de definição semântica e o verificador de violações do contrato semântico.

O primeiro módulo, o gerenciador de arquivos de definição semântica, possui funcionalidades independentes e não é utilizado durante o processo de verificação de um artefato. Ele foi elaborado para permitir que uma equipe de desenvolvedores possa executar mais facilmente tarefas administrativas em relação aos arquivos de definição semântica. Este módulo oferece sete funcionalidades, que são executadas a partir do interpretador de comandos do sistema operacional. Estas funcionalidades operam sobre arquivos que se encontram em diretórios internos da ferramenta. Cada projeto é representado sob a forma de um destes diretórios internos, que devem ser criados pela equipe de desenvolvedores utilizando o módulo gerenciador de arquivos de definição semântica. Para cada projeto três espaços de nome são fornecidos, para os três

tipos de arquivo de definição semântica: extensão, global ou local. As funcionalidades são:

- `Add`: Adiciona um arquivo de definição semântica ao projeto e espaço de nomes desejado. Caso se deseje adicionar um arquivo de definição semântica global, não é necessário fornecer o nome do projeto.
- Exemplo de uso: “add e.nome_do_arquivo.xml nome_do_projeto”.
- `Create Project`: Cria um novo projeto dentro da estrutura de diretórios da ferramenta
- Exemplo de uso: “create_project nome_do_projeto”.
- `List`: Lista todos os arquivos de definição semântica em um projeto
- Exemplo de uso: “list”.
- `Move`: Move um arquivo de definição semântica para outro espaço de nomes do mesmo projeto.
- Exemplo de uso: “move l.nome_do_arquivo.xml e”.
- `Remove`: Remove um arquivo de definição semântica do projeto e do espaço de nomes desejado. O arquivo é apagado do sistema de arquivos.
- Exemplo de uso: “remove g.nome_do_arquivo.xml nome_do_projeto”.
- `Remove Project`: Remove um projeto da estrutura de diretórios da ferramenta.
- Exemplo de uso: “remove_project nome_do_projeto”.
- `Verify`: Verifica se o conteúdo dos arquivos de definição semântica respeita as regras de composição apresentadas na seção 3.1.
- Exemplo de uso: “verify”.

Todas as funcionalidades deste módulo possuem uma estrutura de execução adequada para o ambiente dos interpretadores de comando dos sistemas operacionais e, principalmente, para automação por ferramentas de desenvolvimento.

O segundo módulo da ferramenta `VERITAS` é o responsável pela verificação do artefato e produção do relatório que consolida os resultados, e atua

como um caminhador de árvores sintáticas sobre as entradas fornecidas. Na primeira fase de sua operação, o módulo de verificação executa a ferramenta ANTLR através de uma biblioteca fornecida pela mesma, aguardando até que termine a análise sintática. Em seguida, a AST gerada é recebida pela ferramenta. A segunda fase é a composição das definições semânticas. A VERITAS processa todas as definições semânticas em um de seus projetos, selecionado pelo desenvolvedor, e cria um arquivo temporário que contém as definições que serão utilizadas na verificação. As regras de composição são simples, e estão descritas na seção 3.1. A terceira e mais importante fase de operação é o processamento da AST. A ferramenta caminha na árvore, avaliando todas as expressões presentes em sua estrutura. Cada expressão é composta por símbolos léxicos, e cada um destes símbolos possui um tipo definido pela gramática, como discutido na seção 3.2. Para cada expressão avaliada, se ela contiver um símbolo léxico do tipo operando ou função, seu identificador é obtido. Deste, o radical é extraído e consultado no arquivo composto de definições semânticas. Caso seja encontrada alguma regra que envolva o radical pesquisado, a expressão deve ser investigada:

- Uma declaração de variável de um tipo: todas as regras de restrição deste tipo são verificadas.
- Um operando como parte de uma operação: todas as regras de restrição de operações deste operando são verificadas.
- Uma função: todas as regras de restrição de argumentos e retorno da função são verificadas.

Em cada regra aplicável, o verificador analisa os outros símbolos léxicos envolvidos para decidir se indicará ou não a revisão do trecho de código, ou seja, se há possibilidade de quebra de contrato semântico. Em caso afirmativo são armazenadas todas as informações referentes a possível violação, e a verificação continua para a próxima expressão.

Por último, a ferramenta elabora o relatório de verificação, que é a consolidação de toda a informação que é armazenada em memória no decorrer de sua operação. O relatório produzido pela ferramenta é um arquivo em formato texto, que contém indicações de revisão ordenadas cronologicamente em função da hora de detecção da visualização. Estas indicações têm o intuito de alertar a

equipe de trechos do código que podem conter defeitos derivados da violação do contrato semântico de algum tipo semântico previamente definido. Apenas uma indicação é apresentada por expressão. Isto evita que expressões, particularmente aquelas que contam com diversos tipos, gerem um grande número de indicações e poluam o relatório. O conteúdo de cada indicação é:

- Artefato: Indica o artefato em que existe a suspeita de uma violação do contrato semântico. Exibe o nome do artigo da forma que se encontra no sistema de arquivos da máquina.
- Linha: Indica a linha em que foi detectada a possível violação do contrato semântico.
- Tipo Semântico: Indica qual tipo semântico está envolvido na possível violação, exibindo o radical e o nome completo do tipo.
- Arquivo de Definição: Indica em qual arquivo de definição se encontra a regra utilizada na detecção. Uma letra indica o tipo de arquivo de definição: E (extensão), G (global) e L (local).
- Regra: Identifica qual regra foi utilizada para a detecção da possível violação, exibindo seu atributo `id`.

Abaixo, a figura 6 ilustra um relatório de verificação da ferramenta `VERITAS`:

```
Consttruir --- VERITAS Log

1.
Artefato = PersonContainer.JAVA
Linha = 21
Tipo Semantico = favorite_store_list
Arquivo de Definicao = logistics_model.xml
Regra = BR03

2.
Artefato = PersonContainer.JAVA
Linha = 22
Tipo Semantico = favorite_store_list
Arquivo de Definicao = logistics_model.xml
Regra = BR03

3.
Artefato = PersonContainer.JAVA
Linha = 41
Tipo Semantico = customer
Arquivo de Definicao = user_model.xml
Regra = BR12

4.
Artefato = Person.JAVA
Linha = 198
Tipo Semantico = visited_store
Arquivo de Definicao = user_model.xml
Regra = BR16

Consttruir --- VERITAS Log
```

Figura 7 – Exemplo de relatório de resultado de verificação.

4 Estudo Qualitativo

Para a elaboração do presente trabalho, foi percebida a necessidade de se realizar um estudo qualitativo com o objetivo de avaliar a eficiência e eficácia do uso de análise estática para detectar conflitos semânticos e a notação `SemTypes` e a ferramenta `VERITAS`. Este estudo desempenhou três papéis importantes: (i) fornecer resultados que serviram como base para as conclusões gerais a respeito da solução apresentada, (ii) indicar suas necessidades de correção, evolução e adaptação e (iii) permitir que a solução fosse refinada ao longo da elaboração deste trabalho para que resultados mais interessantes fossem obtidos nas fases finais do estudo realizado.

Entretanto, a partir da pesquisa realizada para apresentar e detalhar o estado da arte, foram levantadas questões interessantes relacionadas a aspectos não-funcionais da solução proposta no presente trabalho e ao uso de análise estática e de uma notação de especificação semântica em processos de desenvolvimento de `software`. Embora estas questões não estivessem dentro do escopo do objetivo original do estudo, foram inseridas como objetivos secundários e investigadas juntamente com os objetivos principais no decorrer da realização do trabalho.

4.1 Método

O método selecionado para o estudo qualitativo conduzido neste trabalho é a pesquisa qualitativa, descrita em Bogdan (1975) e Denzin (2000), que possui características adequadas a estudos informais, particularmente em contextos subjetivos onde são avaliadas a percepção e a opinião de pessoas (Creswell, 2002). Pesquisas qualitativas têm facilidade em destacar aspectos subjetivos e motivações não explícitas, e auxiliam na busca de percepções e entendimento sobre a natureza de uma questão. Seu caráter exploratório, que estimula o pensamento livre sobre temas e conceitos, está alinhado com o intuito de explorar pela primeira vez a notação `SemTypes` e a ferramenta `VERITAS`.

Tipicamente, pesquisas qualitativas exigem o apoio de técnicas auxiliares de coleta de dados, como a realização de entrevistas ou formulários. Para apoiar a pesquisa realizada neste estudo foram utilizadas três técnicas de coleta de dados:

questionário autoaplicável estruturado, observação de uso e entrevista presencial semiestruturada. Um questionário estruturado é composto por um conjunto de perguntas sequenciais que possuem alternativas de resposta fixas, ou seja, onde as respostas se limitam a sim, não, dados quantitativos ou múltipla escolha. Estes questionários não requerem a participação de um pesquisador ou entrevistador, podendo ser preenchidos pelos participantes apenas com seu conhecimento sobre o domínio. A observação de uso é realizada pelo acompanhamento do participante durante a realização do estudo qualitativo. Através desta atividade, o pesquisador pode observar interações e reações dos participantes que não são facilmente obtidas através de entrevistas, questionários, formulários. As entrevistas presenciais semiestruturadas são conjuntos de perguntas flexíveis, organizadas como uma lista de questões ou tópicos que devem ser cobertos, que são respondidas em uma conversa informal entre o pesquisador e o participante. Neste tipo de entrevista é possível seguir uma ordem diferente do roteiro preparado, e até mesmo propor novas questões no decorrer da atividade caso o pesquisador perceba uma oportunidade. Uma característica importante destas entrevistas – e útil para os objetivos do estudo qualitativo realizado no presente trabalho – é que apesar do roteiro elaborado pelo pesquisador o participante possui liberdade para desenvolver as respostas na direção que considera adequada, explorando os aspectos percebidos como mais relevantes.

4.2 Planejamento

O planejamento do estudo qualitativo conduzido foi dividido em duas partes: (i) definição das questões de investigação e (ii) seleção de voluntários.

Para a primeira etapa do planejamento, a definição das questões de investigação, foi utilizada a abordagem `Goal-Question-Metric`, ou `GQM` (BASILI et al., 1994). Esta abordagem, proposta por Victor Basili, define um processo de medição deve ser realizado de acordo com um objetivo específico de perguntas que represente a definição operacional do objetivo da pesquisa, aumentando sua eficiência e eficácia. O principal motivo de escolha desta abordagem é seu caráter orientado a objetivos, adequado para a medição de produtos e processos de software. O modelo de medição descrito por Basili é dividido em três níveis (Figura 8) direcionados para as áreas de interesse do

medidor: Nível conceitual, que representa o objetivo; Operacional, que representa o conjunto de questões usadas para definir modelos do objeto de estudo e focar no objeto que caracteriza o acesso e conclusão de um objetivo específico; Quantitativo, que representa um conjunto de métricas baseadas nos modelos e associadas a cada questão na tentativa de representá-las de forma mensurável.

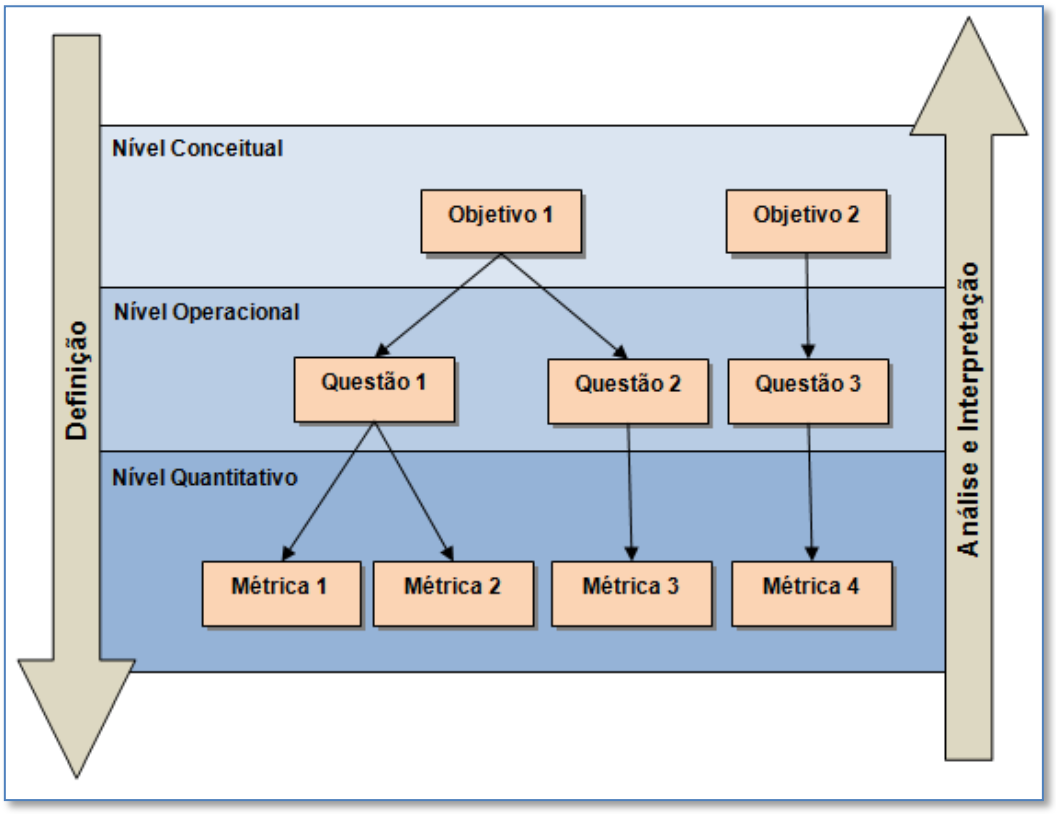


Figura 8 – GQM – Definição e interpretação do processo de medição.

A abordagem descreve um processo que envolve três passos. As métricas são definidas a partir de uma perspectiva `top-down`, onde primeiro são definidos os principais objetivos do processo de medição. Os objetivos da medição são definidos em termos da entidade, propósito, atributos de qualidade, ponto de vista e ambiente. Em seguida, são derivadas as perguntas que devem ser respondidas para determinar se estes objetivos foram atingidos. Por último, deve-se decidir o que será medido para que as perguntas sejam respondidas adequadamente. A análise e interpretação dos resultados são realizadas de forma oposta, em uma perspectiva `botton-up`, onde os dados das métricas são utilizados para verificar se os objetivos foram ou não alcançados. A Figura 8 ilustra o paradigma `GQM`,

contendo seus diferentes níveis, perspectiva `top-down` para definição das métricas e botton-up para análise e interpretação.

Abaixo, são listadas as principais questões de investigação derivadas dos objetivos do estudo realizado:

- A ferramenta é adequada (eficiente) para detecção de conflitos semânticos?
- A ferramenta é adequada (eficaz) para detecção de conflitos semânticos?
- A notação é adequada para definições semânticas?

O questionário de resultados e observações (Apêndice B) foi derivado destas perguntas e contém as métricas observadas do processo de medição. As questões secundárias de investigação foram exploradas através da entrevista semiestruturada, mas algumas métricas foram estabelecidas para auxiliar a análise e se encontram no questionário de resultados e observações. Abaixo, as questões contempladas pelo o roteiro de entrevista (Apêndice C):

- A curva de aprendizado da ferramenta é adequada?
- A curva de aprendizado da notação é adequada?
- Alguma fase do processo de desenvolvimento foi modificada após o início de utilização da notação ou da ferramenta?
- Algum processo de garantia da qualidade foi modificado após o início de utilização da notação ou da ferramenta?
- O processo de levantamento e análise de requisitos foi modificado após o início de utilização da notação ou da ferramenta?
- Os benefícios obtidos com a utilização da solução justificam seu custo operacional?
- Os usuários apresentaram dificuldades de operação da ferramenta?
- Quais cenários foram mais adequados para a utilização da ferramenta?
- Quais são as limitações da abordagem na detecção de conflitos semânticos?

Para a segunda etapa do planejamento do estudo, a seleção de participantes, foram definidos critérios para compor o grupo de investigação. Esta seleção é de

extrema importância em uma pesquisa qualitativa, pois interfere diretamente na qualidade das informações que servirão de base para a análise e compreensão do problema investigado, e exige cuidado na elaboração dos critérios. Durante esta fase do estudo foi construído um perfil com base nos critérios definidos, elaborado para servir como orientação na seleção dos participantes. Todas as características deste perfil estão alinhadas com as questões de investigação apresentadas nesta seção, tanto as principais quanto as secundárias. O perfil desejado é de (i) um desenvolvedor de software com mais de dois anos de experiência – para ser considerado desenvolvedor pleno – na linguagem de programação Java e suas ferramentas mais comuns, (ii) com experiência em revisão de software, (iii) experiência no uso de testes unitários automatizados, (iv) que participe dos processos de garantia de qualidade de software da organização e (v) em um processo de desenvolvimento de software que documente requisitos (Meszaros, 2007). Em função do caráter voluntário do estudo, do tempo requerido de cada um dos participantes e da formação heterogênea das equipes disponíveis, a seleção baseada em perfil não pôde ser muito rigorosa: três programadores com menos de dois anos de experiência, um deles com vínculo de estágio, foram incluídos no grupo de participantes. Uma das equipes participantes não possuía experiência no uso de testes unitários automatizados, e as outras equipes não trabalhavam com desenvolvimento orientado a testes (`Test-Driven Development`) (Beck, 2002).

4.3 Execução

O estudo qualitativo foi realizado com a participação de quatro equipes de desenvolvimento, nomeadas E1, E2, E3 e E4 no presente trabalho. O estudo teve início em agosto de 2012 e foi finalizado em janeiro de 2013. As equipes E1, E2 e E3, ingressaram concomitantemente no estudo em setembro. Em função da demanda de um novo projeto na organização, em meados de setembro a equipe E3 foi reduzida a metade do tamanho que possuía no início do estudo, e um dos membros da configuração final da equipe foi absorvido de um projeto cancelado e não havia participado das reuniões iniciais do estudo. Este acontecimento levou ao afastamento da equipe E3 do estudo, e nenhum dado coletado – seja por questionário, entrevista ou feedback espontâneo – foi utilizado na consolidação

dos resultados. Isto se deve ao fato da equipe E3 ter tido um tempo inadequado de exposição à solução e de reflexão sobre as mudanças no processo de desenvolvimento, e ou poucos dados coletados podem ser questionados em relação a sua qualidade. A equipe E4 ingressou no estudo no início do mês de outubro, brevemente após o afastamento da equipe E3, e permaneceu até o final do estudo.

A equipe E1 é a única equipe de desenvolvimento de uma empresa `startup` de tecnologia da informação formada em junho de 2012, atualmente conta com cinco desenvolvedores. Nestes, estão dois dos quatro sócios fundadores e um desenvolvedor pouco experiente – menos de dois anos de experiência nas tecnologias utilizadas no processo de desenvolvimento. Seu único produto, um sistema de comércio eletrônico de uso geral, ainda está em fase de construção. Este sistema tem o propósito de servir com base para empresas que desejam expandir seu nicho de negócios para o ambiente digital, sejam estas empresas estabelecidas ou em fase inicial. Seus principais módulos são de gerenciamento de produtos, estoque, finanças e comunicação. Toda a estrutura é modular e houve um grande esforço para torná-la tão geral quanto possível, permitindo que diferentes negócios utilizem a aplicação apenas configurando seu conteúdo e forma. Todo o processo de desenvolvimento de software da equipe ainda está em fase de amadurecimento, mas as práticas e metodologias selecionadas – como a metodologia de gerenciamento de projetos e a de documentação de requisitos – já foram implantadas. Embora a equipe tenha consciência de que ainda precisa de tempo para este amadurecimento, seus integrantes têm um perfil interessado e inovador, avaliando métodos e práticas modernas e fazendo escolhas bem informadas. A equipe demonstra grande preocupação com a garantia de qualidade do software sendo desenvolvido.

A equipe E2 é parte de uma empresa de médio porte com mais de dez anos de participação no mercado e mais de cinquenta desenvolvedores em seu quadro operacional. Os principais produtos desta empresa são sistemas `web`, tipicamente ferramentas de gerenciamento de informação e de conteúdo para uso interno de organizações. A equipe de participantes do estudo trabalha em um sistema de gestão do conhecimento feito sob medida para uma empresa de grande porte, conta com oito integrantes e possui a maior média de experiência por desenvolvedor das equipes voluntárias – mesmo considerando um integrante ainda

inexperiente com vínculo de estágio. O sistema de gestão do conhecimento está em uso há quase sete anos pela empresa contratante, que continua demandando manutenções corretivas e evolutivas. A aplicação é construída com base em um pequeno número de entidades simples, mas possui uma elevada quantidade de relacionamentos e complexas regras de negócio. O processo de desenvolvimento atualmente utilizado pela organização foi implantado há mais de quatro anos, tendo atingido um elevado nível de maturidade e operação. Todos os desenvolvedores contratados são treinados por um mês nas práticas e metodologias antes de ingressar em uma equipe.

A equipe E4 é a única equipe de desenvolvimento de uma empresa independente de desenvolvimento de jogos eletrônicos. Ambos os sócios fundadores da empresa – um engenheiro de computação e um designer – fazem parte da equipe de seis integrantes. A equipe possui o mesmo perfil misto dos sócios, com três integrantes formados em cursos de artes e design e os outros três da área de informática. Apenas um dos integrantes de informática têm menos de dois anos de experiência como desenvolvedor de jogos, e todos os integrantes de artes e design possuem experiência nas ferramentas e linguagens de programação utilizados pela equipe no projeto. Embora a empresa tenha sido criada em torno de um único projeto, periodicamente realiza trabalhos de desenvolvimento de jogos para plataformas móveis comerciais com o intuito de arrecadar capital necessário para sua operação. O processo de desenvolvimento da empresa é baseado inteiramente em metodologias ágeis, como o framework de gerenciamento de projetos SCRUM descrito em Schwabe (2004) e a prática de levantamento e documentação de requisitos conhecida como User Stories (Beck, 2002), com os quais os integrantes já estavam familiarizados em função de suas antigas experiências profissionais.

A execução do estudo foi dividida em quatro etapas sequenciais: (i) reunião de kickoff, (ii) treinamento e operacionalização, (iii) acompanhamento e (iv) encerramento do estudo. Todas as equipes participantes passaram individualmente pelas mesmas etapas, na mesma ordem. Entretanto, a duração das etapas variou bastante entres as equipes, em função da disponibilidade e experiência de cada uma.

Na primeira etapa, a reunião de kickoff, todos os participantes de uma determinada equipe eram reunidos para serem instruídos sobre a ferramenta, a notação e o objetivo do estudo. Todos os presentes tinham um conhecimento básico sobre os temas abordados na reunião, pois já haviam passado pela abordagem inicial e aceito participar do estudo. A abertura das reuniões foi uma apresentação sobre o problema de conflitos semânticos na qualidade de software, seguida pela descrição detalhada – inclusive código-fonte e decisões de implementação – da ferramenta VERITAS e uma pausa para perguntas. Depois, a notação SemTypes era apresentada brevemente, pois sua cobertura detalhada seria realizada na próxima etapa. A última fase da reunião era utilizada para explicar detalhadamente os objetivos, perguntas e métricas do estudo, contextualizando todo o trabalho que seria feito. As reuniões de kickoff tiveram duração inferior a quarenta minutos com todas as equipes.

A segunda etapa do estudo foi o treinamento e operacionalização da solução. O primeiro passo do treinamento das equipes foi a descrição detalhada da notação SemTypes, com exemplos de uso desenvolvidos durante este treinamento e baseados na realidade da equipe em questão. Em seguida, era iniciado um pequeno workshop sobre a operação da ferramenta e suas funcionalidades, utilizando os arquivos de definição semântica criados ao longo da explicação da notação. A parte final do treinamento consistia na apresentação do questionário de resultados e observações (Apêndice B) e instrução de como este questionário deveria ser periodicamente preenchido e enviado. O tempo de execução desta foi de aproximadamente duas horas nas equipes E1 e E2 e de três horas na equipe E4.

A terceira etapa do estudo realizado foi o acompanhamento. Nesta etapa, cada uma das equipes foi auxiliada através da remoção dos impedimentos reportados e da solução das dificuldades encontradas, com o intuito de proteger e manter em funcionamento o processo de coleta de dados. Foram considerados como impedimento, por exemplo, defeitos na ferramenta, falta de documentação do usuário ou limitações de funcionalidade. A remoção destes impedimentos demandou um grande tempo de envolvimento com a equipe, pois nenhum processo de comunicação e solução de impedimentos foi estabelecido no início do estudo. Exemplos de problemas que foram considerados como dificuldades

enfrentadas pelas equipes são: uso inadequado da ferramenta, preenchimento equivocado do questionário de resultados e observações e dificuldades de utilizar a notação `SemTypes` para descrever tipos semânticos. O tempo utilizado com cada equipe nesta etapa oscilou muito. A equipe E1 demandou praticamente vinte horas em reuniões e discussões sobre os impedimentos e dificuldades. A equipe E2 demandou aproximadamente duas horas para as mesmas atividades, e manteve a comunicação predominantemente por mensagens eletrônicas. Por último, a equipe E4 precisou de aproximadamente de quatro horas. O processo utilizado pelas equipes participantes nesta etapa era simples, baseado em duas diretrizes. A primeira diretriz determinava que os desenvolvedores deveriam alternar entre a inspeção e a ferramenta `VERITAS` na primeira verificação de um artefato, ou seja, metade das verificações iniciais utilizaria inspeção e a outra metade utilizaria a ferramenta `VERITAS`. Sendo assim, a liberdade da equipe de utilizar ambas as abordagens foi respeitada, mas era necessário selecionar a primeira apenas para questões de preenchimento do questionário de resultados e observações. Desta forma, considerando que não haveria um grande desvio na média de complexidade dos artefatos verificados pelas duas abordagens nas equipes participantes, os resultados obtidos seriam válidos. O cenário mais interessante seria o de divisão de uma equipe em dois grupos, simulando para cada artefato uma verificação inicial com ambas abordagens e permitindo a comparação entre os resultados. Como não era um cenário razoável dentro das limitações do estudo realizado, e a utilização da ferramenta `VERITAS` na primeira verificação invalidaria os dados da inspeção, este esquema de alternância de abordagens foi proposto.

A segunda diretriz apenas determina que as equipes são livres para a realização de testes unitários e funcionais, mas deveriam executar os testes desejados antes da inspeção ou ferramenta, sempre que possível. Desta forma, nos casos em que houvesse um caso de teste para detecção de conflitos semânticos, seria possível observar o desempenho dos testes frente ao uso da ferramenta ou inspeção

A quarta e última etapa realizada foi o encerramento do estudo, no início de janeiro de 2013. Na primeira parte do encerramento foram realizadas as entrevistas presenciais semiestruturadas. De cada equipe dois integrantes foram

selecionados: o líder da equipe, em todos os casos o principal interessado no estudo, e um desenvolvedor. Cada um destes integrantes foi entrevistado individualmente, com base no roteiro de entrevista previamente elaborado (Apêndice C). Os participantes tiveram liberdade de guiar a entrevista em direções diferentes da planejada pelo roteiro, assim como o entrevistar possuir liberdade de elaborar novas perguntas com base na conversa e oportunidades observadas. Estas entrevistas foram realizadas em ambiente informal sem nenhum tipo de registro eletrônico, pois a maioria dos participantes se mostrou desconfortável com a opção de gravação de voz. Um documento chamado de relatório de entrevista foi criado para cada entrevista com o intuito de registrar as respostas dos participantes. Ao final das entrevistas foi conduzida uma reunião final com cada equipe para discutir os resultados obtidos, com foco no impacto na rotina dos desenvolvedores, e agradecer pela participação. Embora tenha sido uma reunião informal e sem um guia, algumas observações e comentários interessantes foram registrados. A tabela abaixo apresenta o tempo utilizado com cada uma das etapas, com cada equipe:

Tabela 1 – Tempo utilizado por equipe por etapa.

	E1	E2	E4
Reunião de Kickoff	35 minutos	25 minutos	35 minutos
Treinamento e Operacionalização	2 horas	2 horas e 15 minutos	3 horas
Acompanhamento	20 horas	2 horas	4 horas
Encerramento Entrevistas	1 hora e 30 minutos	2 horas	1 hora e 30 minutos
Encerramento Reunião Final	20 minutos	25 minutos	25 minutos

4.4 Resultado das Medições

Após o término das quatro etapas planejadas do estudo, todos os dados coletados foram compilados para futuras referências e análise. No total, foram preenchidos setenta e nove questionários de resultados e observações e seis relatórios de entrevista. Nas primeiras semanas do estudo alguns questionários foram respondidos de forma equivocada, mesmo após o treinamento realizado na etapa anterior. Infelizmente, nem todas as dúvidas puderam ser detectadas e tratadas, acarretando em uma perda de oito questionários, um pouco mais de 10% da amostra. Optou-se por descartar todos os resultados destes questionários defeituosos e nenhuma atitude foi tomada no sentido de corrigi-los. Em todos os momentos em que foi detectado um defeito nos questionários o esforço era dirigido no sentido da compreensão de sua causa e na educação de todas as equipes, responsáveis ou não. O objetivo era proteger as próximas amostras evitando que o erro fosse repetido. Nas tabelas desta seção são apresentados os resultados dos questionários fornecidos, separados por equipe. Para cada equipe participante existem duas tabelas: (i) uma com o resultado parcial e (ii) outra com o resultado consolidado. As tabelas de resultados parciais apresentam três resultados distintos, um para cada fase de contabilização e análise dos relatórios enviados pelas equipes participantes. Ao final, a última tabela apresenta a consolidação de todos os resultados de todas as equipes participantes. Abaixo, segue o modelo de tabela de resultados e a descrição das métricas utilizadas:

Tabela 2 – Modelo de tabela de resultados

Planilha de Resultados Consolidados da Equipe XPTO [X Questionários Recebidos, Y Questionários utilizados]	
Especificação do artefato define contrato semântico	*Indica o percentual de artefatos que possui alguma definição semântica em sua documentação de requisitos.*
Métodos Utilizados	
Inspeção	*Indica o percentual de artefatos inspecionados.*
Teste	*Indica o percentual de artefatos para o qual foram criados testes unitários.*
Análise Estática	*Indica o percentual de artefatos analisados pela ferramenta VERITAS.*
Inspeção	
Tempo de duração da inspeção	*Indica o tempo médio utilizado nas inspeções.*
Defeitos detectados por artefato	*Indica a média, por artefato, de defeitos encontrados durante a inspeção.*
Conflitos semânticos detectados por artefato	*Indica a média, por artefato, de conflitos semânticos encontrados durante a inspeção.*
Identificadores inválidos	*Indica a média, por artefato, de identificadores inválidos encontrados durante a inspeção.*
Teste	
Tempo de elaboração dos testes	*Indica o tempo médio utilizado na elaboração dos testes unitários.*
Existe caso de teste para contrato semântico	*Indica o percentual de artefatos que possuem casos de teste que contenham cláusulas de verificação do contrato semântico.*
Casos de teste executados por artefato	*Indica a média, por artefato, de casos de teste executados durante a fase de teste.*
Defeitos detectados por artefato	*Indica a média, por artefato, de defeitos encontrados durante a fase de teste.*
Conflitos semânticos detectados por artefato	*Indica a média, por artefato, de conflitos semânticos encontrados durante a fase de teste*

Análise Estática	
Tempo de análise	*Indica o tempo médio utilizado para a análise da ferramenta VERITAS.*
Número de arquivos de definição semântica utilizados por artefato	*Indica a média, por artefato, do número de arquivos de definição utilizados pela análise da ferramenta VERITAS.*
Conflitos semânticos indicados por artefato	*Indica a média, por artefato, de prováveis conflitos semânticos encontrados durante a análise da ferramenta VERITAS.*
Conflitos semânticos detectados por artefato	*Indica a média, por artefato, de conflitos semânticos encontrados durante a análise da ferramenta VERITAS.*
Defeitos detectados nos arquivos de definição semântica	*Indica a média, por arquivo de definição semântica, de defeitos em sua sintaxe ou semântica.*

Os resultados obtidos através do estudo realizado são suficientes para responder as questões principais de investigação propostas no início deste capítulo. No decorrer do estudo, trinta e seis quebras de contrato semântico foram corretamente detectadas, de um total de trinta e nova violações indicadas – comprovando que a ferramenta foi eficiente nas indicações em função do baixo número de falsos positivos. Outro importante indicador da eficiência da ferramenta é o tempo de análise. O consumo médio por artefato de verificação utilizando a solução proposta é de apenas um minuto, enquanto a média das inspeções é de aproximadamente vinte e três minutos e dos testes unitários em torno de uma hora e vinte um minutos. Este era exatamente o resultado esperado de uma ferramenta de análise estática, servindo para constatar que a solução se comportava adequadamente.

Como descrito no parágrafo anterior e exposto das tabelas desta seção, trinta e seis violações de contrato semântico foram detectadas pela ferramenta. Destas, dezoito foram detectadas apenas pela ferramenta, sendo ignoradas pelas outras abordagens. A média de detecção de conflitos semânticos por artefato foi em torno de 1,24 para a ferramenta, aproximadamente 0,89 no caso dos testes e apenas 0,7 nas inspeções. A detecção exclusiva de grande parte dos defeitos e a

média de detecção por artefato são indícios de que a ferramenta é eficaz, mesmo havendo um grande espaço para melhoria destes resultados. Estas melhorias são discutidas em detalhes na seção 5.2. Outro ponto é que, embora em termos absolutos o número de defeitos detectados seja pequeno, o custo e o impacto estimados de alguns destes defeitos em produção é grande – mesmo considerando que todos os sistemas em desenvolvimento das organizações que participaram do estudo não são críticos.

A última questão de investigação se refere à adequação da notação para definições semânticas: o resultado não é conclusivo. A notação foi suficiente para que resultados satisfatórios fossem obtidos, mas houve grande resistência e dificuldades em seu uso. O número de identificadores inválidos foi alto, assim como a média de defeitos nos arquivos de definição semântica. A eficácia da ferramenta é dependente da capacidade da notação em especificar contratos semânticos, e os resultados obtidos foram bons, mas não excepcionais. Isto pode ser mais um indício de que a notação possui limitações. Por fim, percebeu-se que os desenvolvedores enfrentaram dificuldades para estabelecer explicitamente contratos semânticos, da mesma forma que enfrentam dificuldades para definir responsabilidades e escopo de classes, por exemplo. Sendo assim, a capacidade de definição semântica da notação pode ser interpretada como (i) a causa dos resultados medianos ou (ii) como um sintoma das habilidades de modelagem das equipes participantes do estudo. Abaixo, as tabelas de resultados:

Estudo Qualitativo

Tabela 3 – Planilha de resultados da equipe E1

Planilha de Resultados da Equipe E1 [36 Questionários Recebidos, 36 Questionários utilizados]	
Especificação do artefato define contrato semântico	1º Fase: 63,63% [7/11] 2º Fase: 94,11% [16/17] 3º Fase: 100% [8/8]
Métodos Utilizados	
Inspeção	1º Fase: 90,90% [10/11] 2º Fase: 100% [17/17] 3º Fase: 100 % [8/8]
Teste	1º Fase: 90,90% [10/11] 2º Fase: 100% [17/17] 3º Fase: 100 % [8/8]
Análise Estática	1º Fase: 72,72% [8/11] 2º Fase: 100% [17/17] 3º Fase: 87,50% [7/8]
Inspeção	
Tempo de duração da inspeção	1º Fase: 12 minutos 2º Fase: 22 minutos 3º Fase: 20 minutos
Defeitos detectados por artefato	1º Fase: 4,75 [19/4] 2º Fase: 4,00 [24/6] 3º Fase: 2,50 [20/8]
Conflitos semânticos detectados por artefato	1º Fase: 1,00 [4/4] 2º Fase: 0,66 [4/6] 3º Fase: 0,75 [6/8]
Identificadores inválidos	1º Fase: 2,25 [9/4] 2º Fase: 1,50 [9/6] 3º Fase: 0,25 [2/8]
Teste	
Tempo de elaboração dos testes	1º Fase: 55 minutos 2º Fase: 1 hora e 25 minutos 3º Fase: 55 minutos
Existe caso de teste para contrato semântico	1º Fase: 0,00% [0/10] 2º Fase: 29,41% [5/17] 3º Fase: 25,00% [2/8]
Casos de teste executados por artefato	1º Fase: 3,5 [35/10] 2º Fase: 4,94 [84/17] 3º Fase: 4,38 [35/8]
Defeitos detectados por artefato	1º Fase: 0,30 [3/10] 2º Fase: 0,64 [11/17] 3º Fase: 0,38 [3/8]

Conflitos semânticos detectados por artefato	1º Fase: – [0/0] 2º Fase: 0,20 [1/5] 3º Fase: 0,00 [0/2]
Análise Estática	
Tempo de análise	1º Fase: 1 minuto 2º Fase: 1 minuto 3º Fase: 1 minuto
Número de arquivos de definição semântica utilizados por artefato	1º Fase: 7,20 [36/5] 2º Fase: 4,00 [32/8] 3º Fase: 2,75 [11/4]
Conflitos semânticos indicados por artefato	1º Fase: 0,80 [4/5] 2º Fase: 1,75 [14/8] 3º Fase: 0,75 [3/4]
Conflitos semânticos detectados por artefato	1º Fase: 0,80 [4/5] 2º Fase: 1,75 [13/8] 3º Fase: 0,75 [3/4]
Defeitos detectados nos arquivos de definição semântica	1º Fase: 0,33 [1/3] 2º Fase: 0,16 [1/6] 3º Fase: 0,00 [0/3]

Estudo Qualitativo

Tabela 4 – Planilha de resultados consolidados da equipe E1

Planilha de Resultados Consolidados da Equipe E1 [36 Questionários Recebidos, 36 Questionários utilizados]	
Especificação do artefato define contrato semântico	86,11% [31/36]
Métodos Utilizados	
Inspeção	97,22% [35/36]
Teste	97,22% [35/36]
Análise Estática	86,11% [32/36]
Inspeção	
Tempo de duração da inspeção	18 minutos
Defeitos detectados por artefato	3,50 [63/18]
Conflitos semânticos detectados por artefato	0,78 [14/18]
Identificadores inválidos	1,11 [20/18]
Teste	
Tempo de elaboração dos testes	1 hora e 5 minutos
Existe caso de teste para contrato semântico	20,00% [7/35]
Casos de teste executados por artefato	4,4 [154/35]
Defeitos detectados por artefato	0,49 [17/35]
Conflitos semânticos detectados por artefato	0,14 [1/7]
Análise Estática	
Tempo de análise	1 minuto
Número de arquivos de definição semântica utilizados por artefato	4,64 [79/17]
Conflitos semânticos indicados por artefato	1,24 [21/17]
Conflitos semânticos detectados por artefato	1,18 [20/17]
Defeitos detectados nos arquivos de definição semântica	0,17 [2/12]

Tabela 5 – Planilha de resultados da equipe E2.

Planilha de Resultados da Equipe E2 [17 Questionários Recebidos, 14 Questionários utilizados]	
Especificação do artefato define contrato semântico	1º Fase: 0,00% [0/4] 2º Fase: 25,00% [1/4] 3º Fase: 16,67% [1/6]
Métodos Utilizados	
Inspeção	1º Fase: 75,00% [3/4] 2º Fase: 75,00% [3/4] 3º Fase: 100% [6/6]
Teste	1º Fase: 75,00% [3/4] 2º Fase: 75,00% [3/4] 3º Fase: 100% [6/6]
Análise Estática	1º Fase: 25,00% [1/4] 2º Fase: 50,00% [2/4] 3º Fase: 50,00% [3/6]
Inspeção	
Tempo de duração da inspeção	1º Fase: 31 minutos 2º Fase: 30 minutos 3º Fase: 20 minutos
Defeitos detectados por artefato	1º Fase: 1,50 [3/2] 2º Fase: 3,00 [6/2] 3º Fase: 2,00 [6/3]
Conflitos semânticos detectados por artefato	1º Fase: 0,00 [0/2] 2º Fase: 0,50 [1/2] 3º Fase: 0,33 [1/3]
Identificadores inválidos	1º Fase: 0,00 [0/2] 2º Fase: 2,50 [5/2] 3º Fase: 2,00 [6/3]
Teste	
Tempo de elaboração dos testes	1º Fase: 1 hora e 30 minutos 2º Fase: 2 horas e 10 minutos 3º Fase: 2 horas e 5 minutos
Existe caso de teste para contrato semântico	1º Fase: 0,00% [0/3] 2º Fase: 33,33% [1/3] 3º Fase: 16,67% [1/6]
Casos de teste executados por artefato	1º Fase: 3,00 [12/4] 2º Fase: 3,00 [12/4] 3º Fase: 2,50 [15/6]
Defeitos detectados por artefato	1º Fase: 1,50 [6/4] 2º Fase: 1,25 [5/4] 3º Fase: 1,83 [11/6]
Conflitos semânticos detectados por artefato	1º Fase: 0,00 [0/4] 2º Fase: 0,00 [0/4] 3º Fase: 0,00 [0/6]
Análise Estática	
Tempo de análise	1º Fase: 1 minuto 2º Fase: 1 minuto

Estudo Qualitativo

	3º Fase: 1 minuto
Número de arquivos de definição semântica utilizados por artefato	1º Fase: 1,00 [1/1] 2º Fase: 2,00 [4/2] 3º Fase: 2,33 [7/3]
Conflitos semânticos indicados por artefato	1º Fase: 2,00 [2/1] 2º Fase: 1,00 [2/2] 3º Fase: 1,00 [3/3]
Conflitos semânticos detectados por artefato	1º Fase: 1,00 [1/1] 2º Fase: 2,00 [2/1] 3º Fase: 3,00 [3/2]
Defeitos detectados nos arquivos de definição semântica	1º Fase: 1,00 [1/1] 2º Fase: 2,00 [2/1] 3º Fase: 3,00 [3/2]

Estudo Qualitativo

Tabela 6 – Planilha de resultados consolidados da equipe E2

Planilha de Resultados Consolidados da Equipe E2 [17 Questionários Recebidos, 14 Questionários utilizados]	
Especificação do artefato define contrato semântico	13,33% [2/15]
Métodos Utilizados	
Inspeção	85,71% [12/14]
Teste	85,71% [12/14]
Análise Estática	42,85% [6/14]
Inspeção	
Tempo de duração da inspeção	27 minutos
Defeitos detectados por artefato	2,14 [15/7]
Conflitos semânticos detectados por artefato	0,29 [2/7]
Identificadores inválidos	1,57 [11/7]
Teste	
Tempo de elaboração dos testes	1 hora e 55 minutos
Existe caso de teste para contrato semântico	16,67% [2/12]
Casos de teste executados por artefato	2,79 [39/14]
Defeitos detectados por artefato	1,49 [20/14]
Conflitos semânticos detectados por artefato	0 [0/2]
Análise Estática	
Tempo de análise	1 minuto
Número de arquivos de definição semântica utilizados por artefato	2 [12/6]
Conflitos semânticos indicados por artefato	1,16 [7/6]
Conflitos semânticos detectados por artefato	1,16 [7/6]
Defeitos detectados nos arquivos de definição semântica	1,5 [6/4]

Estudo Qualitativo

Tabela 7 – Planilha de resultados da equipe E4.

Planilha de Resultados da Equipe E4 [26 Questionários Recebidos, 21 Questionários utilizados]	
Especificação do artefato define contrato semântico	1º Fase: 85,71% [6/7] 2º Fase: 100% [7/7] 3º Fase: 100% [7/7]
Métodos Utilizados	
Inspeção	1º Fase: 85,71% [6/7] 2º Fase: 71,42% [5/7] 3º Fase: 85,71% [6/7]
Teste	1º Fase: 85,71% [6/7] 2º Fase: 71,42% [5/7] 3º Fase: 85,71% [6/7]
Análise Estática	1º Fase: 42,85% [3/7] 2º Fase: 71,42% [5/7] 3º Fase: 42,85% [3/7]
Inspeção	
Tempo de duração da inspeção	1º Fase: 33 minutos 2º Fase: 27 minutos 3º Fase: 27 minutos
Defeitos detectados por artefato	1º Fase: 2,33 [7/3] 2º Fase: 3,66 [11/3] 3º Fase: 2,25 [9/4]
Conflitos semânticos detectados por artefato	1º Fase: 0,33 [1/3] 2º Fase: 1,33 [4/3] 3º Fase: 0,50 [2/4]
Identificadores inválidos	1º Fase: 0,00 [0/3] 2º Fase: 0,66 [2/3] 3º Fase: 0,25 [1/4]
Teste	
Tempo de elaboração dos testes	1º Fase: 1 hora e 15 minutos 2º Fase: 2 horas e 10 minutos 3º Fase: 1 hora e 5 minutos
Existe caso de teste para contrato semântico	1º Fase: 0,00 [0/6] 2º Fase: 0,00 [0/5] 3º Fase: 0,16 [1/6]
Casos de teste executados por artefato	1º Fase: 1,83 [11/6] 2º Fase: 4,40 [22/5] 3º Fase: 1,67 [10/6]
Defeitos detectados por artefato	1º Fase: 0,83 [5/6] 2º Fase: 2,20 [11/5] 3º Fase: 1,00 [6/6]
Conflitos semânticos detectados por	1º Fase: – [0/0]

artefato	2º Fase: – [0/0] 3º Fase: 2,00 [2/1]
Análise Estática	
Tempo de análise	1º Fase: 1 minuto 2º Fase: 1 minuto 3º Fase: 1 minuto
Número de arquivos de definição semântica utilizados por artefato	1º Fase: 0,50 [1/2] 2º Fase: 1,67 [5/3] 3º Fase: 1,00 [2/2]
Conflitos semânticos indicados por artefato	1º Fase: 1,00 [2/2] 2º Fase: 2,00 [6/3] 3º Fase: 1,50 [3/2]
Conflitos semânticos detectados por artefato	1º Fase: 0,50 [1/2] 2º Fase: 1,67 [5/3] 3º Fase: 1,50 [3/2]
Defeitos detectados nos arquivos de definição semântica	1º Fase: 0,00 [0/0] 2º Fase: 1,00 [1/1] 3º Fase: 0,00 [0/2]

Estudo Qualitativo

Tabela 8 – Planilha de resultados consolidados da equipe E4

Planilha de Resultados Consolidados da Equipe E4 [26 Questionários Recebidos, 21 Questionários utilizados]	
Especificação do artefato define contrato semântico	95,23% [20/21]
Métodos Utilizados	
Inspeção	80,95% [17/21]
Teste	80,95% [17/21]
Análise Estática	52,38% [11/21]
Inspeção	
Tempo de duração da inspeção	29 minutos
Defeitos detectados por artefato	2,7 [27/10]
Conflitos semânticos detectados por artefato	0,7 [7/10]
Identificadores inválidos	0,3 [3/10]
Teste	
Tempo de elaboração dos testes	1 hora e 30 minutos
Existe caso de teste para contrato semântico	5,88% [1/17]
Casos de teste executados por artefato	2,53 [43/17]
Defeitos detectados por artefato	1,29 [22/17]
Conflitos semânticos detectados por artefato	2 [2/1]
Análise Estática	
Tempo de análise	1 minuto
Número de arquivos de definição semântica utilizados por artefato	1,14 [8/7]
Conflitos semânticos indicados por artefato	1,57 [11/7]
Conflitos semânticos detectados por artefato	1,29 [9/7]
Defeitos detectados nos arquivos de definição semântica	0,33 [1/3]

Estudo Qualitativo

Tabela 9 – Planilha de resultados consolidados das equipes participantes.

Planilha de Resultados das Equipes [79 Questionários Recebidos, 71 Questionários utilizados]	
Especificação do artefato define contrato semântico	74,64% [53/71]
Métodos Utilizados	
Inspeção	90,14% [64/71]
Teste	90,14% [64/71]
Análise Estática	60,56% [43/71]
Inspeção	
Tempo de duração da inspeção	23 minutos
Defeitos detectados por artefato	2,7 [27/35]
Conflitos semânticos detectados por artefato	0,7 [7/35]
Identificadores inválidos	0,3 [3/35]
Teste	
Tempo de elaboração dos testes	1 hora e 23 minutos
Existe caso de teste para contrato semântico	15,15% [10/66]
Casos de teste executados por artefato	3,58 [236/66]
Defeitos detectados por artefato	0,89 [59/66]
Conflitos semânticos detectados por artefato	0,3 [3/10]
Análise Estática	
Tempo de análise	1 minuto
Número de arquivos de definição semântica utilizados por artefato	3,41 [99/29]
Conflitos semânticos indicados por artefato	1,3 [39/30]
Conflitos semânticos detectados por artefato	1,2 [36/30]
Defeitos detectados nos arquivos de definição semântica	0,47 [9/19]

4.5 Resultado das Entrevistas

Nesta seção são apresentadas as respostas obtidas com as perguntas do roteiro de entrevista e perguntas livres, nos casos em que os relatórios de entrevista indicam observações interessantes dos participantes. Os dados qualitativos coletados nas entrevistas realizadas, em função de sua natureza, não puderam ser consolidados da mesma forma que os dados quantitativos obtidos com o questionário de resultados e observações. Para isto, cada pergunta é discutida separadamente em uma subseção, que inclui as respostas de todos os participantes. Nestas subseções os resultados obtidos são discutidos de forma geral, considerando todas as equipes participantes e as perguntas livres realizadas no momento da pergunta em questão.

4.5.1 Você considera que a ferramenta VERITAS e a notação SemTypes agregaram valor ao seu processo de desenvolvimento? Você as utilizaria outra vez?

As equipes E1 e E2 foram enfáticas ao responder que a solução agregou valor ao processo de desenvolvimento. A equipe E1 foi detalhista ao descrever o número de defeitos encontrados e a satisfação com os resultados. Embora os entrevistados da equipe E1 não soubessem estimar quantitativamente o prejuízo evitado, percebiam que seria grande o dano à imagem de um produto, que estava sendo planejado para competir em um mercado monopolista já estabelecido. Esta equipe também destacou prematuramente na entrevista os principais artefatos que foram modificados pela experiência de uso da ferramenta. São eles: `user stories`, testes de aceitação, casos de teste e `javadoc`.

A equipe E2 respondeu que a ideia da ferramenta era muito interessante, e conceitualmente seria muito grande o valor agregado. Entretanto, em função do estado inicial de maturidade da solução, os entrevistados expuseram que todo o potencial da abordagem não havia sido explorado. Ainda assim, consideraram o valor agregado na prática um diferencial importante do processo de qualidade e se mostraram interessados em acompanhar o desenvolvimento da ferramenta.

A equipe E4 respondeu que viu valor na ferramenta, mas não a considerou interessante na forma e contexto utilizada. Os modelos de dados do jogo em desenvolvimento são muito simples e possuem um número muito alto de

entidades, e o custo de desenvolvimento imposto pelo uso da notação não agradou. Além disto, a equipe declarou (i) que os defeitos detectados pela ferramenta seriam facilmente detectados pelos testes funcionais e (ii) que o impacto de um defeito em produção é pequeno, dada a natureza não-crítica da aplicação e a cultura de atualização periódica e emergencial típica de jogos eletrônicos.

4.5.2 Alguma outra equipe da organização se interessou pela ferramenta?

Não houve nenhum relato de outros grupos interessados em utilizar a solução proposta dentro das organizações das equipes participantes. Em função da duração do estudo e das características das organizações participantes, esta era uma resposta esperada, pois nenhum trabalho específico de divulgação foi executado. Por exemplo, apenas a equipe E2 fazia parte de uma organização com mais de uma equipe de desenvolvimento. Ainda assim, geograficamente estão dispostos em um local onde se encontram múltiplos departamentos e não estão próximos do departamento de tecnologia da informação ou de outras equipes de desenvolvimento da empresa.

4.5.3 Quais dificuldades foram detectadas pela equipe no aprendizado de uso da ferramenta? E em sua instalação? O que poderia ser melhorado nestes aspectos?

Todas as equipes responderam que a utilização da ferramenta, do ponto de vista das funcionalidades disponíveis, é simples e intuitiva. O tempo utilizado na segunda etapa no estudo e o desempenho das equipes comprovam estas respostas. Nenhuma equipe necessitou de ajuda para execução da ferramenta ao longo do estudo. A equipe E2, entretanto, citou que estava insatisfeita com o primeiro processo manual de instalação da solução, mas que após a revisão e o novo empacotamento considerava o processo de instalação adequado.

4.5.4 Quais dificuldades foram encontradas na elaboração e manutenção dos arquivos de definição semântica? O que poderia ser melhorado nestes aspectos?

Todas as respostas envolvendo a notação `SemTypes` indicaram algum grau de desconforto. As equipes relataram que o tempo necessário para elaborar os contratos semânticos é alto, mas nenhuma sugestão de melhoria do processo foi citada. A interface baseada no interpretador de comandos do sistema operacional, utilizada para manutenção dos arquivos de definição semântica, foi citada pelas equipes E1 e E2 como um fator adicional de complexidade e custo em tempo de desenvolvimento.

As principais dificuldades relatadas para elaboração dos contratos semânticos foram (i) a nomeação de identificadores, (ii) a definição do comportamento exato de cada entidade definida e (iii) evitar duplicidade de definições. A criação de nomes adequados, descritivos e que facilitem a memorização e rápida identificação foi um problema. A equipe E4 apontou que embora esta seja uma dificuldade comum nos processos de desenvolvimento, esta foi potencializada pela necessidade particular dos radicais de serem curtos. As equipes E1 e E2 encontraram dificuldades na definição dos contratos semânticos de entidades de negócio, particularmente nas entidades com muitos relacionamentos. A percepção da equipe E2 era de que entidades com muitos relacionamentos possuíam muitos interesses e dependências transversais, o que dificultava a definição da entidade de forma isolada, como requerido pela notação. A equipe E2 também encontrou dificuldade em gerenciar os arquivos de definição semântica de forma a evitar múltiplas definições de entidades ou definições muito similares. Pontuaram que todo este gerenciamento foi baseado no conhecimento tácito dos desenvolvedores do projeto.

As dificuldades para manutenção dos contratos semânticos se concentraram em torno da interface da ferramenta. As tarefas de visualização e edição do conteúdo destes arquivos não são apoiadas pela ferramenta, e sua corretude deve ser verificada manualmente. Ao longo do experimento, uma a uma, as equipes passaram a utilizar seus ambientes de desenvolvimento para o tratamento dos arquivos de definição semântica.

4.5.5 Houve resistência da equipe na adoção da notação SemTypes?

Todas os líderes de equipe encontraram grande resistência na adoção da notação, percebida como um grande inconveniente para o uso da ferramenta. Alegações sobre a falta de necessidade de uso da notação em determinada tarefa foram comuns, particularmente na equipe E4, mesmo nos casos claramente indicados. As equipes E2 e E4 afirmaram que o reforço constante foi fundamental para assegurar assiduidade de uso da ferramenta. Esta resposta era esperada em função da opinião expressada nas perguntas anteriores.

4.5.6 Ocorreu reuso de arquivos de definição semântica?

Todas as equipes responderam que os arquivos de definição semântica foram reutilizados ao longo do tempo de experiência, mas que o custo foi alto em função da complexidade da elaboração dos arquivos de definição semântica. Entretanto, o treinamento fornecido na segunda etapa do estudo teve grande foco nesta questão e pode ter influenciado as equipes. A equipe E1 citou na resposta que estava seguindo as instruções do treinamento ao elaborar arquivos mais gerais para o projeto. A equipe E4 encontrou dificuldades na reutilização das definições semânticas e acredita ter elaborado arquivos muito extensos e com conteúdo demasiadamente heterogêneo.

4.5.7 A ferramenta se adaptou adequadamente ao seu processo de desenvolvimento e controle de qualidade de software?

Todas as equipes responderam que a execução da ferramenta é rápida e não tem nenhum impacto no custo em tempo de desenvolvimento dos processos de qualidade de software. Ainda, os defeitos indicados pela ferramenta eram avaliados da mesma forma os defeitos indicados pelo processo de qualidade tradicional, se somando ao grupo de defeitos que deveria ser corrigido. A equipe E4 destacou que as tarefas extras, tais como criação e edição dos arquivos de definição semânticas, foram transformadas em tarefas de desenvolvimento e adicionadas ao gerenciador de tarefas como tarefas comuns da iteração.

4.5.8 Quais dificuldades foram detectadas pela equipe no uso da ferramenta? E na visualização dos resultados? O que poderia ser melhorado nestes aspectos?

Todas as equipes foram enfáticas ao citar que a ausência de uma interface de operação ou da integração com as ferramentas existentes no ambiente de desenvolvimento empobreceu o uso da ferramenta. Mais uma vez, a ferramenta foi citada como um aplicativo simples e de fácil utilização, do ponto de vista das funcionalidades oferecidas. A visualização dos resultados em arquivos texto também foi citada como um fator negativo, embora a estrutura final do relatório de defeitos tenha recebido elogios da equipe E2. A integração com os ambientes de desenvolvimento utilizados foi apontada como a solução mais direta e interessante pra tratar destas dificuldades.

4.5.9 A ferramenta foi utilizada consistentemente para aprovação dos artefatos?

Apenas a equipe E1 utilizou a ferramenta em todos os artefatos desenvolvidos ao longo do estudo. As equipes E2 e E4 selecionaram um grupo de artefatos para o estudo e restringiram a utilização da ferramenta. Ambas as equipes justificaram esta decisão citando a necessidade de conciliar o estudo com as demandas da organização. De qualquer forma, as equipes relataram que nos artefatos em que a ferramenta foi utilizada o resultado da verificação serviu como entrada para o processo de aceitação do artefato. Todas as equipes demonstraram estar conscientes de que defeitos derivados de quebras de contrato semântico podem ser sérios eram cuidadosas quando a ferramenta indicava uma violação. A equipe E2 recebeu um incentivo positivo do `Product Owner`[9] do projeto, que nas reuniões de revisão foi comunicado da tentativa da equipe de melhorar a qualidade do aplicativo avaliando uma ferramenta acadêmica para verificação do código fonte. Este colaborador do projeto manifestou um desejo de continuar a utilizar a abordagem no projeto.

9 Pichler, R. Being an Effective Product Owner.
http://www.scrumalliance.org/articles/44-being-an-effective-product-owner

4.5.10 Os resultados de verificação apresentados pela ferramenta eram utilizados consistentemente pela equipe?

Assim como na resposta anterior, as equipes indicaram que trataram com seriedade os relatórios de verificação fornecidos pela ferramenta, da mesma forma que trataram as informações fornecidas pelas ferramentas habituais dos seus processos de desenvolvimento. Todos os defeitos detectados pela ferramenta receberam o mesmo tratamento e prioridade dos defeitos detectados pelo uso de testes unitários e inspeção, e no gerenciados de tarefas de nenhuma equipe era possível diferenciar visualmente as tarefas de correção de defeitos originadas da ferramenta `VERITAS`.

4.5.11 Alguma percepção de mudança na elicitação ou documentação de requisitos após o estudo realizado?

A equipe E1 foi a única equipe a responder que o conceito de definição semântica foi incorporado ao processo de desenvolvimento. Na primeira pergunta esta equipe já havia indicado que alguns artefatos – `user stories`, testes de aceitação, casos de teste e `javadoc` – foram modificados pelo estudo. Nas `user stories`, esta modificação consistiu na inclusão de informação sobre o significado das entidades de negócio e sobre as operações válidas, em um glossário semântico. Os testes de aceitação e unitários, por sua vez, passaram a incluir a verificação das operações realizadas envolvendo estas entidades. Por último, a documentação do código passou a incluir uma referência aos arquivos de definição semântica e descrever mais detalhadamente os tipos utilizados em cada declaração de tipo. A mesma equipe citou que ocorreu um processo de conscientização sobre os riscos de quebra de um contrato semântico e que este processo foi a principal força motivadora de mudança do processo de desenvolvimento e seus artefatos.

4.5.12 Alguma percepção de mudança na elaboração de casos de teste após o estudo realizado?

Apenas a equipe E1 passou por uma reformulação do processo de desenvolvimento e seus artefatos. Esta equipe, inclusive, respondeu a presente pergunta na resposta anterior. Entretanto, o desenvolvedor da equipe E2 citou que

em alguns poucos artefatos, especificamente os que descreviam regras de negócio, os casos de teste incluíram verificações semânticas que deveriam ser utilizadas pela ferramenta. O mesmo desenvolvedor, ao ser perguntado, respondeu que acredita que isto possa ser visto como o início de um processo de conscientização e evolução do processo de desenvolvimento.

4.5.13 Qual a sua visão geral da solução? Acredita existir espaço no mercado para ela?

A equipe E1 respondeu que a acredita na solução pois acredita no problema. Responderam também que a solução precisa ser refinada, mas que pode ser tornar um produto suficientemente interessante para que algumas organizações de software livre a adotem como projeto oficial, ou para que seja comercializado como ferramenta de qualidade. A equipe E2 respondeu que a solução possui funcionalidades interessantes mas ainda não está pronta para uso, principalmente em função das questões de interface e instalação. Se a solução evoluísse do estágio de protótipo para o de produto, estariam dispostos a apresentar para a gerência e recomendar a adoção institucional, pois já têm o aval do `Product Owner` do processo, um dos gerentes de produto da organização. Acreditam que no presente momento a solução encontraria mais resistência que motivação de adoção. A equipe E4 respondeu que considera a solução como uma ferramenta de qualidade, que cumpre seu propósito, mas que possui um mercado restrito – o líder da equipe E4 acredita que o custo da solução só se justifica para algumas organizações. Ainda assim, a equipe também citou que acredita que a ferramenta seria bem recebida pela comunidade de desenvolvedores caso fosse disponibilizada gratuitamente, e que a extensão da verificação para a linguagem de programação C++ deveria ser a prioridade para assegurar uma base expressiva de usuários.

5 Considerações Finais

Este capítulo apresenta as contribuições do presente trabalho ao estado da arte, suas sugestões de melhorias e trabalhos futuros. A principal fonte das sugestões aqui apresentadas é o estudo qualitativo realizado, embora exista grande influência dos trabalhos relacionados e literatura pesquisada.

5.1 Contribuições

As principais contribuições deste trabalho são a ferramenta `VERITAS` e a notação `SemTypes`. A `VERITAS` permite que desenvolvedores verifiquem código fonte em busca de violações do contrato semântico, potencialmente detectando defeitos muito sutis e prevenindo que estes cheguem à produção e causem prejuízos. Da mesma forma, a `SemTypes` é utilizada para reforçar explicitamente o contrato semântico de um artefato e viabilizar a detecção – manual ou automática – deste tipo de violação.

O estudo realizado auxiliou nesta percepção de adequação da ferramenta e, ainda, das mudanças ocorridas no processo de desenvolvimento decorrentes do uso de uma notação de especificação semântica. A conscientização de que o processo de desenvolver software também é uma tarefa de definição de contratos semânticos fez com que alguns participantes do estudo se preocupassem, explicita e ativamente, com o significado das entidades. Artefatos como casos de teste, documentos de requisitos, testes de aceitação e casos de uso passaram a ser compreendidos de forma diferente, e sua elaboração acompanhou a nova compreensão.

O estudo também serviu para comprovar a bem conhecida necessidade de reforçar ativamente a manutenção de processos de qualidade de software. Desenvolvedores, mesmo quando motivados e conscientes dos benefícios de uma nova prática, podem progressivamente abandoná-la e retornar aos hábitos antigos. Diversos trabalhos na literatura citam a dificuldade de se criar de forma sustentável um novo hábito no processo de desenvolvimento, como por exemplo o de `TDD` (Beck, 2002). Embora a ferramenta não tenha encontrado resistência neste sentido, sua operação é simples e exige pouco tempo e atenção do

desenvolvedor, muitos problemas ocorreram na adoção da notação `SemTypes`. Mesmo no papel de voluntários neste estudo os participantes consistentemente evitavam o uso da notação, e as tarefas de elaboração e manutenção dos arquivos de definição semântica eram gradualmente esquecidas. Apenas o reforço contínuo por parte da organização, representada através da figura dos líderes de equipe, levou a realização destas tarefas - que ao final do estudo se transformaram em práticas na equipe E1. Até a conclusão deste trabalho a equipe E2 continuou utilizando a ferramenta na manutenção e evolução de alguns artefatos críticos, mesmo após a conclusão do estudo qualitativo.

5.2 Trabalhos Futuros

Ao longo da elaboração deste trabalho foi coletado um grande volume de informações sobre a solução proposta. O estudo qualitativo permitiu que desenvolvedores, trabalhando em diferentes organizações, culturas e processos, interagissem com a ferramenta no intuito de melhorar a qualidade de seu trabalho. Consequentemente, suas opiniões acerca de todos os aspectos relacionados a solução refletem necessidades de equipes e projetos do mundo real. Após a análise realizada sobre os questionários de resultados e observações e os relatórios de entrevista, algumas sugestões de melhoria e trabalhos futuros são apresentadas nesta seção. Estas sugestões envolvem tanto a evolução da ferramenta quanto a evolução da abordagem .

A solução apresentada neste trabalho é um protótipo, e não uma ferramenta de produção. Ela apresenta o refinamento e características de operação esperados de um protótipo. Para que se atinja um estado de maturidade adequado a produção, alguns passos devem ser dados: (i) disponibilização de um aplicativo de instalação para diferentes plataformas e sistemas operacionais, (ii) elaboração de um manual de usuário completo, com informações detalhadas da ferramenta, (iii) elaboração de um tutorial de utilização da notação `SemTypes` e (iv) transformação da ferramenta em um módulo de extensão, ou `plug-in`, de pelo menos um ambiente de desenvolvimento Java. A partir deste amadurecimento da ferramenta seria interessante disponibilizar o trabalho para a comunidade de desenvolvedores Java sob licença `open source`. Ao fazer isto, a solução seria analisada e avaliada por um número de desenvolvedores – de todo tipo de

organização e cultura – maior do que seria possível em um estudo, possibilitando a evolução em direções que não puderam ser previstas ao longo deste trabalho.

Outra necessidade da ferramenta se tornou clara após o estudo qualitativo: a ampliação do escopo das linguagens de programação verificáveis. Embora a linguagem Java seja suficientemente expressiva em participação do mercado para justificar sua escolha para a realização deste trabalho, existe um grande número de linguagens igualmente expressivas e que poderiam se beneficiar da solução apresentada. Existem duas dificuldades para ampliação do escopo da ferramenta: (i) encontrar gramáticas formais adequadas e (ii) generalizar o processador de ASTs para compreender estruturas de diferentes linguagens de programação. Uma das atividades que mais consumiu tempo na construção da solução foi a busca por uma gramática formal Java adequada. Encontrar e testar gramáticas para ampliar o escopo da ferramenta, especialmente nos casos de linguagens menos estruturadas, é uma tarefa longa e complexa. Generalizar o processador de ASTs seria o segundo passo na ampliação do escopo da ferramenta. As linguagens de programação possuem diferentes estruturas sintáticas, e a consequência direta é que ASTs geradas pela ferramenta ANTLR seriam muito heterogêneas. Para trabalhar com estas ASTs heterogêneas, os artefatos da ferramenta VERITAS devem ser atualizados para tratar as estruturas sintáticas de cada linguagem de programação que se deseja verificar. Em um cenário onde a VERITAS seja disponibilizada como ferramenta open source, os próprios usuários poderiam realizar esta evolução com base em suas necessidades e progressivamente tornar a ferramenta mais interessante para os desenvolvedores de software em geral.

Muitos sistemas de informação são construídos em torno de bases de dados, possuindo grande dependência da arquitetura e estrutura dos bancos de dados utilizados. Uma passo natural de evolução da ferramenta é a extensão da verificação semântica a operações em bancos de dados. É comum o uso de scripts para automatizar as atualizações da base de dados, seja para operações de criação, atualização, recuperação ou remoção de entidades. Estes scripts, tipicamente construídos utilizando-se a linguagem de programação SQL, podem obter o mesmo benefício da solução proposta neste trabalho que os artefatos estudados. Até mesmo operações diretas sobre o banco, que são escritas em SQL

ou na linguagem de desenvolvimento da aplicação nos casos em que frameworks são utilizados, poderiam ser verificadas pela ferramenta.

Por último, existem duas questões importantes relacionadas à notação `SemTypes`: (i) padronização de identificadores e tratamento automático de relacionamentos. Foi observado no estudo realizado, e em muitos trabalhos na literatura, que o uso de notações pode se tornar muito complexo, muito rapidamente. Sem uma orientação objetiva, como por exemplo através de guias, treinamento ou padrões, desenvolvedores podem apresentar as mesmas dificuldades que encontram na nomeação de identificadores ao nomear radicais. Isto tem efeito negativo no processo de desenvolvimento, e compromete a eficiência e eficácia da ferramenta, pois todo o seu funcionamento – o contrato semântico – é baseado nos identificadores. Ainda, entidades podem ser compostas e possuir relacionamentos. Um exemplo: a entidade aluno, uma lista encadeada e uma lista de alunos. Cada uma destas entidades possui um contrato semântico bem definido, que pode ser especificado através da notação `SemTypes` e o uso correto dos identificadores criados. Neste caso, a solução trata cada entidade de forma independente, e não reconhece automaticamente as relações semânticas implícitas de uma lista de alunos com uma lista encadeada ou um aluno. Isto aumenta o custo de desenvolvimento, e torna a definição de todo tipo de relacionamentos – associação, agregação, composição, dependência, generalização ou realização – uma tarefa manual e sujeita a erros. A padronização dos radicais e o tratamento de relacionamentos é um importante passo na evolução da solução, solucionando ambas as questões discutidas. Entretanto, ao longo da elaboração deste trabalho, nenhuma solução interessante foi concebida e estas questões permanecem abertas para discussão.

Referências

An Evaluation of FindBugs. Sandcastle Group. 2009. Disponível em <http://www.cs.cmu.edu/~aldrich/courses/654/tools/Sandcastle-FindBugs-2009.pdf >

ARIE VAN DEURSEN, PAUL KLINT. **Little languages: Little maintenance?**. Journal of Software Maintenance. 10:75–92. 1998.

ARIE VAN DEURSEN ET AL. **Domain-Specific Languages: An Annotated Bibliography**. ACM SIGPLAN, Vol. 35, Issue 6. pp. 26-36. 2000.

ARNDT VON STAA. Notas de aula da disciplina **INF 1413 - Teste de Software**. PUC-Rio. 2012.

ARTHUR C. FLECK. **Formal Methods in Software Engineering**. 2006.

BERTRAND MEYER. **Applying "Design by Contract"**. OOSA. 1994.

BOGDAN, R.; TAYLOR, S. J. **Introduction to qualitative research methods: a phenomenological approach to the Social Sciences**. New York: J. Wiley and Sons. 1975.

CERT. **Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests**. SEI. 2000.

CHARLES SIMONYI. **Hungarian Notation**. DEXA. 2006.

CHARLES SIMONYI. **Meta-Programming: A Software Production Method**. XEROX Palo Alto Research Center. 1977.

CHRIS GRINDSTAFF. **FindBugs, Part 1: Improve the quality of your code**. IBM Developer Works. 2006.

CHRIS GRINDSTAFF. **FindBugs, Part 2: Writing custom detectors**. IBM Developer Works. 2008.

Code Conventions for the Java Programming Language. Oracle. 1999. Disponível em <http://www.oracle.com/technetwork/java/codeconv-138413.htm>

GERARD MESZAROS. **xUnit Test Patterns: Refactoring Test Code**. Addison-Wesley. 2007.

JARI AARNIALA. **Instrumenting Java bytecode**. University of Helsinki. 2005.

JASON COHEN. **Best Kept Secrets of Peer Code Review**. Smartbear Software. 2006.

Referências

Java Language and Virtual Machine Specification. Oracle. 1999. Disponível em <http://docs.oracle.com/javase/specs/>

JOHN W. CRESWELL. **Research Design: Qualitative, Quantitative, and Mixed Methods Approaches**. SAGE Publications, Inc. 2002

JOSHUA BLOCH. **Effective Java**. Addison-Wesley. 2008.

KARL E. WIEGERS. **Peer Reviews in Software: A Practical Guide**. Addison-Wesley. 2002.

KEN SCHWABE. **Agile Project Management with Scrum**. Microsoft Press. 2004.

KENT BECK. **Test Driven Development: By Example.** Addison-Wesley Professional. 2002.

KRISTINA LUNDQVIST. **Formal Methods**. MIT. 2002. Disponível em <http://web.mit.edu/16.35/www/lecturenotes/FormalMethods.pdf>

MARTIN FOWLER. **Domain Specific Languages**. Addison-Wesley Professional. 2010.

MICHAEL FAGAN. **Design and Code Inspections to Reduce Errors in Program Development**. IBM Systems Journal, Vol. 15, No. 3. pp. 182-211. 1976.

Microsoft Hungarian Notation. Microsoft. 1999. Disponível em <http://msdn.microsoft.com/en-us/library/aa260976%28VS.60%29>

MIKE COHN. **User Stories Applied: For Agile Software Development**. 304 pages. Addison-Wesley Professional. 2004.

NICK RUTAR ET AL. **A Comparison of Bug Finding Tools for Java**. University of Maryland. 2004.

NORMAN BROWN. **Industrial-Strength Management Strategies**. IEEE Software, Vol. 13, No. 4. pp. 94-103. 1996.

Rensselaer Polytechnic Institute. **Operating Systems**. Disponível em <http://www.cs.rpi.edu/academics/courses/fall04/os/c10/index.html>

ROBERT C. MARTIN. **Clean Code: A Handbook of Agile Software Craftsmanship**. Prentice Hall. 2008.

ROBERT C. MARTIN. **The Clean Coder: A Code of Conduct for Professional Programmers**. Prentice Hall. 2011.

Referências

ROBERT M. HERNDON AND VALDIS A. BERZINS. **The realizable benefits of a language prototyping language**. IEEE Transactions on Software Engineering. pp. 803–809. 1988.

ROMAN PICHLER. **Being an Effective Product Owner**. Scrum Alliance. 2007.

TERENCE PARR. **The Definitive ANTLR Reference: Building Domain-Specific Languages**. Pragmatic Bookshelf. 2007.

TERENCE PARR. **The Definitive ANTLR4 Reference**. Pragmatic Bookshelf. 2013.

VICTOR BASILI et al. **The Goal Question Metric Approach**. Encyclopedia of Software Engineering. John Wiley & Sons, 1994.

Apêndice A – Termo de Consentimento

Termo de Consentimento de Participação no Estudo Qualitativo da Notação SemTypes e da ferramenta de Análise Estática VERITAS.

Como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Ciência da Computação da PUC-Rio, um estudo será realizado para avaliar a utilização de análise estática na detecção de conflitos semânticos em tipos de dados. A notação SemTypes e a ferramenta de análise estática VERITAS foram desenvolvidas especialmente para este trabalho e serão o alvo deste estudo.

O objetivo deste estudo é (i) verificar se a abordagem da análise estática é adequada para detecção de conflitos semânticos e (ii) até onde a notação SemTypes e a ferramenta VERITAS são capazes de apoiar nesta tarefa. O estudo realizado compreende uma fase de coleta de dados brutos de processos de garantia de qualidade, o preenchimento de um formulário e a participação em uma entrevista de curta duração – aproximadamente vinte minutos. O desempenho do voluntário não será avaliado ou medido de nenhuma forma, e nenhuma das métricas utilizada é individual. Da mesma forma, o processo de desenvolvimento de software do empregador do voluntário não será medido ou avaliado de nenhuma forma.

Observações importantes:

a) As informações desta pesquisa são confidencias, e serão divulgadas apenas em eventos ou publicações científicas, não havendo identificação dos voluntários ou de seus empregadores. A divulgação destes resultados pauta-se no **respeito à sua privacidade** e o **anonimato** dos mesmos é preservado em quaisquer documentos a serem elaborados. **Nenhuma** informação coletada do processo de desenvolvimento será publicada sem autorização dos voluntários.

b) É importante expressar sua opinião de forma clara e honesta. As impressões dos voluntários do estudo serão utilizadas para refinar a notação e a ferramenta, assim como servir de base para as conclusões da análise qualitativa.

c) O voluntário não é obrigado a finalizar o experimento e possui liberdade para interromper a entrevista ou reter os dados coletados se e quando desejar.

d) É de extrema importância fornecer os dados requisitados em sua forma bruta, sem edição ou tratamento. Estes dados serão utilizados para fundamentar a análise da abordagem e a avaliação da notação e ferramenta desenvolvidas.

Apêndice A – Termo de Consentimento

Com base nas informações presentes neste documento e na reunião de apresentação do trabalho desenvolvido, solicito sua participação no estudo, coletando os dados necessários e utilizando a notação SemTypes e a ferramenta de análise estática VERITAS em seu processo de desenvolvimento.

Eu, _____, estou ciente dos objetivos da pesquisa e de minha participação e responsabilidade. Compreendo as garantias de privacidade e anonimato oferecidas e expresso minha concordância de espontânea vontade em participar deste estudo.

Rio de Janeiro, _____

Assinatura do Voluntário : _____

Apêndice B – Questionário de Resultados e Observações

O questionário abaixo foi utilizado no estudo qualitativo realizado neste trabalho. A elaboração e aplicação do questionário são discutidos detalhadamente no capítulo 4. Para preservar o anonimato e privacidade dos voluntários participantes e das organizações envolvidas, campos de identificação não foram incluídos nos questionário. A versão aqui apresentada, em formato de texto, é uma réplica do questionário utilizado no estudo, originalmente uma planilha eletrônica.

Planilha de Resultados

Identificador do projeto: _____

Identificador do artefato: _____

Especificação do artefato define contrato semântico? Sim ( ) Não ( )

Métodos Utilizados

Inspeção ( ) Teste ( ) Análise Estática ( ) Outros: _____ ( )

Inspeção

Tempo total de inspeção: _____

Número de linhas de código verificadas: _____

Número de defeitos detectados: _____

Número de conflitos semânticos detectados: _____

Número de identificadores não-válidos: _____

Teste

Tempo total de elaboração: _____

Existe caso de teste para contrato semântico Sim ( ) Não ( )

Número de casos de teste executados: _____

Número de defeitos detectados: _____

Número de conflitos semânticos detectados: _____

Apêndice B – Questionário de Resultados e Observações

Análise Estática

Tempo total de Análise: _____

Número de linhas de código verificadas: _____

Arquivos de definição semântica utilizados: _____

Número de conflitos semânticos indicados: _____

Número de conflitos semânticos detectados: _____

Número de defeitos nos arquivos de definição semântica utilizados: _____

Existe caso de teste para contrato semântico Sim ( ) Não ( )

Apêndice C – Roteiro de Entrevista

O roteiro abaixo foi utilizado no estudo qualitativo realizado neste trabalho. A elaboração e aplicação do roteiro são discutidos detalhadamente no capítulo quatro. Para preservar o anonimato e privacidade dos voluntários participantes e das organizações envolvidas, campos de identificação não foram incluídos nos questionário.

Roteiro de Entrevista

1. Você considera que a ferramenta VERITAS e a notação SemTypes agregaram valor ao seu processo de desenvolvimento? Você as utilizaria outra vez?

   [Se sim] Em todo tipo de projeto ou apenas em algumas circunstâncias?

   [Se não] Qual o principal motivo?

2. Alguma outra equipe da organização teve interesse em utilizar a ferramenta?

3. Quais dificuldades foram detectadas pela equipe no aprendizado de uso da ferramenta? E em sua instalação? O que poderia ser melhorado nestes aspectos?

4. Quais dificuldades foram encontradas na elaboração e manutenção dos arquivos de definição semântica? O que poderia ser melhorado nestes aspectos?

5. Houve resistência da equipe na adoção da notação SemTypes?

   [Se sim] A notação foi dispensada para alguns artefatos? O uso de identificadores nulos era comum em artefatos que deveriam ser verificados pela ferramenta?

6. Ocorreu reuso de arquivos de definição semântica?

   [Se sim] Qual escopo do reuso? Entre projetos? Entre equipes?

7. A ferramenta se adaptou adequadamente ao seu processo de desenvolvimento e controle de qualidade de software?

Apêndice C – Roteiro de Entrevista

[Se não] Onde ocorreu a maior incompatibilidade?

8. Quais dificuldades foram detectadas pela equipe no uso da ferramenta? E na visualização dos resultados? O que poderia ser melhorado nestes aspectos?

9. A ferramenta foi utilizada consistentemente para aprovação dos artefatos?

   [Se não] Qual o motivo?

10. Os resultados de verificação apresentados pela ferramenta eram utilizados consistentemente pela equipe?

    [Se não] Qual o motivo? Falsos positivos? Falsos negativos?

11. Alguma percepção de mudança na elicitação ou documentação de requisitos após o estudo realizado?

12. Alguma percepção de mudança na elaboração de casos de teste ou roteiros de inspeção após o estudo realizado?

13. Qual a sua visão geral da solução? Acredita existir espaço no mercado para ela?