



Programa Interdisciplinar de Pós-Graduação em
Computação Aplicada
Mestrado Acadêmico

Leandro Ferreira D'Avila

SW-Context: Um Modelo para *Software Analytics* Baseado em
Sensibilidade ao Contexto

São Leopoldo, 2017

Leandro Ferreira D'Avila

SW-CONTEXT: UM MODELO PARA *SOFTWARE ANALYTICS* BASEADO EM
SENSIBILIDADE AO CONTEXTO

Dissertação apresentada como requisito parcial
para a obtenção do título de Mestre pelo
Programa de Pós-Graduação em Computação
Aplicada da Universidade do Vale do Rio dos
Sinos — UNISINOS

Orientador:
Prof. Dr. Jorge Luis Victória Barbosa

Co-orientador:
Prof. Dr. Kleinner Silva Farias de Oliveira

São Leopoldo
2017

D259s D'Avila, Leandro Ferreira.
SW-Context : um modelo para software analytics baseado em
sensibilidade ao contexto / Leandro Ferreira D'Avila. – 2017.
101 f. : il. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos
Sinos, Programa de Pós-Graduação em Computação Aplicada,
2017.

“Orientador: Prof. Dr. Jorge Luis Victória Barbosa ; co-
orientador: Prof. Dr. Kleinner Silva Farias de Oliveira.”

1. Contexto para software. 2. Informações analíticas. 3. Consciência
situacional. I. Título.

CDU 004

Leandro Ferreira D'Avila

Título: SW-Context: Um Modelo para *Software Analytics* Baseado em Sensibilidade ao Contexto

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em 22 de fevereiro de 2017

BANCA EXAMINADORA

Prof. Dr. Kleinner Silva Farias de Oliveira - UNISINOS

Prof. Dr. Joao Carlos Gluz - UNISINOS

Prof. Dr. Leandro Krug Wives - UFRGS

Prof. Prof. Dr. Jorge Luis Victoria Barbosa (Orientador)

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Sandro José Rigo
Coordenador PPG em Computação Aplicada

AGRADECIMENTOS

O desenvolvimento deste trabalho se deu a partir do incentivo de várias pessoas em momentos e planos distintos. Algumas são lembradas a seguir, às demais fica meu agradecimento pela compreensão e gratidão pelo apoio em momentos difíceis.

Em primeiro lugar, agradeço a educação que meus pais me proporcionaram, mesmo que sem condições financeiras ideais para tal. Ao meu pai, Nelson D'Avila, que me ensinou a ser uma pessoa digna, positiva, e a enfrentar todos os desafios da vida com força, coragem e perseverança, sem perder a alegria. Sempre me incentivava ao final de alguma etapa perguntando qual seria a próxima. À minha mãe, Irene Ferreira D'Avila, que na falta do meu pai, sempre orgulhou-se da minha situação, incentivou a sequência dos meus estudos, mesmo tendo que conviver com a minha ausência e me motivou sempre a chegar mais longe.

À minha filha Betania, que me faz sorrir por pior que seja o meu dia, que me faz querer mais a cada momento e que me ensinou e ensina diariamente o verdadeiro sentido da palavra amor.

Aos meus irmãos, que igualmente me apoiaram de alguma forma neste projeto e que fazem parte deste momento e sempre estarão no meu coração.

Agradeço à família da minha esposa, por igual apoio e compreensão nas nossas ausências.

Ao orientador Jorge Barbosa, por acreditar no meu potencial desde a nossa primeira conversa, antes mesmo do início do mestrado. Estando presente ao longo de todo o curso propiciando, através de sua orientação, os caminhos mais assertivos para chegarmos ao objetivo final.

Ao co-orientador Kleinner Farias que, através de suas disciplinas e experiência, propiciou o entendimento de que seria possível unir as áreas de engenharia de software e computação ubíqua em um projeto de mestrado.

Aos meus colegas Fabio, Daniel e Ismael pelo companheirismo durante a nossa caminhada. Fica o agradecimento especial a CAPES-PROSUP pelo apoio financeiro, sem o qual, a realização deste curso de pós-graduação não seria possível.

À UNISINOS, por me acolher novamente como aluno e disponibilizar toda estrutura necessária para realização desta pesquisa. Aos meus colegas de trabalho, que me apoiaram na construção e avaliação do trabalho e foram impactados de alguma forma durante esse período.

Agradeço, em especial, a minha esposa Ana Carolina, a maior incentivadora deste projeto desafiador. Sempre me apoiou e conduziu todos os nossos assuntos pessoais nas minhas ausências. Nos momentos mais difíceis, sentou ao meu lado e fez com que as coisas se tornassem mais fáceis. Esteve comigo no dia-a-dia e sabe mais do que ninguém quais são os sentimentos e esforços necessários para se conquistar algo tão desafiador. Muito obrigado por existir e fazer parte da minha vida

RESUMO

Diariamente, desenvolvedores de *software* precisam se envolver com atividades de manutenção para adaptar aplicações existentes a novos cenários e necessidades, como por exemplo, novas funcionalidades, correções de defeitos e requerimentos legais. Entretanto, algumas questões organizacionais podem interferir nas atividades dos desenvolvedores impactando na qualidade e manutenibilidade do *software* produzido. Grande volume de documentação obsoleta, dificuldades na utilização desta documentação, dependências entre módulos de *software* e especialistas que deixam as empresas levando o conhecimento de determinados módulos e/ou sistemas são fatores determinantes para o sucesso dos projetos.

Uma das formas de mitigar o impacto destas questões seria a disponibilização de informações úteis referentes aos módulos ou artefatos de *software* de forma qualitativa. A disponibilização destas informações propicia um melhor entendimento do desenvolvedor em relação aos aspectos que cercam o *software* e o seu ambiente. De acordo com a natureza das informações disponibilizadas, os desenvolvedores podem adquirir informações relevantes sobre o *software* em questão.

Essa dissertação apresenta o *SW-Context*, um modelo que permite a combinação de diferentes informações relacionadas a artefatos de *software*, a fim de aprimorar a consciência situacional dos desenvolvedores nas atividades de desenvolvimento e manutenção. Desta forma, os principais desafios do modelo são: a definição de quais informações devem compor o contexto para *software*, o armazenamento estruturado destas informações em históricos de contextos e, finalmente, a análise e disponibilização destas informações de contexto, de forma que possam auxiliar a atividade de desenvolvimento e manutenção de *software*, utilizando o conceito *Software Analytics*.

Foi implementado um protótipo contendo os principais conceitos do modelo proposto. Este protótipo utilizou as informações contextuais de aplicações reais de uma empresa de desenvolvimento de *software* e foi avaliado através de um estudo de caso, onde 12 desenvolvedores o utilizaram pelo período de um mês em suas atividades diárias. Ao final deste período, os desenvolvedores responderam um questionário que abordou a utilidade da ferramenta e a facilidade de uso percebida.

A avaliação do modelo obteve respostas com percentuais satisfatórios tanto em relação à facilidade de uso percebida quanto à utilidade do sistema. Pode-se avaliar que a consolidação das informações contextuais em um local único e a disponibilização qualitativa das informações correlacionadas, através de *dashboard*, atingiu o objetivo de melhorar a consciência situacional dos desenvolvedores nas atividades de manutenção.

Palavras-chave: Contexto para *Software*. Informações Analíticas. Consciência Situacional.

ABSTRACT

Developers need to deal recurrently with the maintenance activities on existing applications in order to adapt them to new scenarios and needs, for example, new features, bug fixing and legal changes. Besides that, developers often deal with organization factors with a potential impact on the success or failure of software development projects. Some of these organization factors are: large amount of old poorly documented software, many interdependencies between software modules and expert developers who left the company.

A way to mitigate the impact of these factors on software correctness and maintainability can be providing useful information regarding the context of code or application under development using the analytics approach. The availability of this information provides a better understanding of the developer in relation to issues surrounding the software and its environment.

SW-Context aims to allow a combination of different information related to software artifacts in order to improve the situational awareness of developers on development and maintenance activities. On this way, the main challenges of the model are: a definition of what information must compose software context, structured storage of the contextual information and, finally, the analysis and availability of this context information in a way to help the development and maintenance activities, using the Software Analytics concept.

A prototype was implemented containing the main concepts of the proposed model. This prototype was prepared with the contextual information of actual applications under development by a software company and the prototype was evaluated through a case study, where 12 developers used it in their daily activities. By the end of this period, the developers responded a questionnaire, in which the usefulness and the ease of use were measured.

The evaluation of the model obtained answers with percentage well placed both in relation to the ease of use as to the usefulness of the system. It can be considered that the consolidation of the contextual information in a single location and the availability of this correlated information in a graphical way, through a dashboard, reached the objective of improving the situational awareness of software developers in maintenance activities.

Keywords: Software Context. Analytics Information. Situational Awareness.

LISTA DE FIGURAS

Figura 1:	Relação entre os conceitos	19
Figura 2:	Facetas de contexto	24
Figura 3:	Questões da abordagem <i>analytics</i>	25
Figura 4:	Visão do <i>Fishtail</i> dentro do Eclipse	30
Figura 5:	Resultado do <i>Strathcona</i> dentro do Eclipse	32
Figura 6:	<i>Interface de codificação de duelos do Code Hunt</i>	33
Figura 7:	<i>Tela de busca avançada do CodeExchange</i>	35
Figura 8:	Resultado interativo do <i>CodeExchange</i>	36
Figura 9:	<i>Gráfico de bolhas do modelo TDCAM</i>	38
Figura 10:	Abordagem para captura de contexto proposta	38
Figura 11:	Modelo de contexto proposto	40
Figura 12:	<i>Plugin no Eclipse</i> demonstrando informações de contexto	41
Figura 13:	Arquitetura e organização modelo <i>SW-Context</i>	49
Figura 14:	Diagrama de componentes do modelo	51
Figura 15:	Diagrama de classes do componente <i>IDataInput</i>	52
Figura 16:	Dimensões consideradas no modelo	53
Figura 17:	Diagrama de classes do componente <i>ContextManager</i>	54
Figura 18:	Diagrama de domínio de contexto para <i>software</i>	55
Figura 19:	Diagrama de classes dos componentes de configuração	58
Figura 20:	Diagrama de classes do <i>IOContextAnalytics</i>	59
Figura 21:	Diagrama de sequência do <i>IOContextAnalytics</i>	60
Figura 22:	Diagrama de classes dos componentes de visualização	61
Figura 23:	Organização das informações no <i>dashboard</i> analítico	62
Figura 24:	Serviço de disponibilização da identificação do artefato	66
Figura 25:	Serviço de disponibilização das métricas do artefato	67
Figura 26:	Serviço de disponibilização das alterações do artefato	68
Figura 27:	Serviço de disponibilização dos incidentes relacionados ao artefato	68
Figura 28:	Serviço de disponibilização dos clientes que usam o artefato	69
Figura 29:	Serviço de disponibilização das parametrizações do modelo	69
Figura 30:	Tela principal do protótipo	71
Figura 31:	Limites das métricas/composições	72
Figura 32:	Composição proporcional	72
Figura 33:	Composição simples	73
Figura 34:	Composição ponderada	73
Figura 35:	Tela de e-mail de alerta enviado automaticamente	74
Figura 36:	<i>Dashboard</i> construído para o experimento	80
Figura 37:	Corretude: pontuação geral	82
Figura 38:	<i>Dashboard</i> utilizado	87
Figura 39:	Resultado da questão 3	90
Figura 40:	Resultado da questão 4	90
Figura 41:	Resultado da questão 5	91
Figura 42:	Resultado das questões 6 e 7	91
Figura 43:	Resultado da questão 8	92
Figura 44:	Resultado da questão 9	92

Figura 45: Resultado da questão 10	93
Figura 46: Resultado da questão 11	93
Figura 47: Resultado da questão 12	94
Figura 48: Resultado da questão 13	94
Figura 49: Resultado da questão 14	95
Figura 50: Resultado da questão 15	95
Figura 51: Resultado da questão 16	96
Figura 52: Utilidade do <i>SW-Context</i> e facilidade de uso percebida	97

LISTA DE TABELAS

Tabela 1:	Evento de interação	28
Tabela 2:	Paralelo de tipos de contexto	34
Tabela 3:	Tabela comparativa dos trabalhos relacionados	43
Tabela 4:	Exemplo de configuração	57
Tabela 5:	Hipóteses formuladas	79
Tabela 6:	Pontuação das atividades em relação a Manutenibilidade	81
Tabela 7:	Teste <i>Pearson Chi-square</i> para corretude	82
Tabela 8:	Estatística descritiva do esforço de manutenção	83
Tabela 9:	Teste de normalidade - Esforço	83
Tabela 10:	Estatística descritiva da taxa de manutenibilidade	84
Tabela 11:	Teste de normalidade - Manutenibilidade	84
Tabela 12:	Métricas primárias	85
Tabela 13:	Configuração da composição - Ponderada	85
Tabela 14:	Configuração da composição - Simples	86
Tabela 15:	Questionário - perguntas de caracterização	88
Tabela 16:	Questionário - perguntas sobre utilidade e facilidade de uso	89

LISTA DE SIGLAS

IDE	Integrated Development Environment
GQM	Goal Question Metric
API	Application Programming Interface
CASE	Computer-Aided Software Engineering
ISO	International Organization for Standardization
CMMI	Capability Maturity Model - Integration
PIM	Personal Information Manager

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Motivação	13
1.2	Definição do Problema e Questão de Pesquisa	15
1.3	Objetivo	16
1.4	Metodologia	16
1.5	Organização do Texto	17
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	Contexto e Histórico de Contextos	19
2.2	Contexto e <i>Software</i>	21
2.3	<i>Software Analytics</i>	24
2.4	Considerações sobre o capítulo	26
3	TRABALHOS RELACIONADOS	27
3.1	Metodologia para escolha dos trabalhos	27
3.2	<i>Mylyn a Plugin to deal with task context</i>	27
3.3	<i>Build usage contexts during program comprehension</i>	28
3.4	<i>Fishtail a Plugin to automate the discovery of relevant code examples from the web</i>	29
3.5	<i>Recommending Task Context: Automation Meets Crowd</i>	30
3.6	<i>Using structural context to recommend source code examples</i>	31
3.7	<i>Harnessign the Crowd: Descontextualizing Software Work</i>	32
3.8	<i>Code Hunt: Context-Driven Interactive Gaming for Learning Programming and Software Engineering</i>	33
3.9	<i>Context in Code Search</i>	34
3.10	<i>StackMine: a Microsoft Project</i>	36
3.11	<i>Test and Defect Coverage Analytics Model</i>	37
3.12	<i>Context-Based Retrieval in Software Development</i>	37
3.13	<i>Context Capture in Software Development</i>	39
3.14	Análise comparativa dos trabalhos relacionados	40
3.15	Considerações sobre o capítulo	45
4	MODELO SOFTWARE CONTEXT	47
4.1	Visão geral	47
4.2	Requisitos	49
4.3	Componentes do Modelo SW-Context	50
4.3.1	Componente <i>IDataService</i>	52
4.3.2	Componente <i>ContextManager</i>	52
4.3.3	Componentes <i>ConfigurationManager</i> e <i>ConfigurationUI</i>	56
4.3.4	Componente <i>IOContextAnalytics</i>	58
4.3.5	Componentes <i>ODataService</i> e <i>VisualizationUI</i>	61
4.4	Considerações sobre o capítulo	63

5 ASPECTOS DE IMPLEMENTAÇÃO	65
5.1 Tecnologias utilizadas	65
5.2 Serviços implementados	66
5.3 Telas implementadas	70
5.4 Considerações sobre o capítulo	74
6 ASPECTOS DE AVALIAÇÃO	75
6.1 Metodologia	75
6.2 Experimento controlado	75
6.3 Estudo de caso na indústria	84
6.3.1 Preparação do protótipo	84
6.3.2 Questionário	87
6.3.3 Resultados do questionário	89
6.4 Considerações sobre o capítulo	97
7 CONSIDERAÇÕES FINAIS	99
7.1 Contribuições	99
7.2 Trabalhos futuros	100
REFERÊNCIAS	102

1 INTRODUÇÃO

Em seu artigo *The computer for de 21st century*, publicado em 1991, WEISER (1991) utilizou pela primeira vez o termo computação ubíqua. A área da Computação Ubíqua estuda o desenvolvimento de técnicas que visam integrar a tecnologia da informação ao cotidiano das pessoas, ou seja, é o uso da tecnologia de uma forma que as pessoas não percebam que estão dando comandos a um computador, em suas tarefas mais rotineiras como conversando com alguém ou locomovendo-se para o trabalho. O autor vislumbrou há mais de vinte anos que, no futuro, computadores teriam sistemas inteligentes e estariam embarcados nos mais triviais objetos como etiquetas de roupas, interruptores de luz e canetas de forma invisível para o usuário.

Para que essa visão se torne realidade é necessário obter informações através das atividades cotidianas das pessoas, a partir de dispositivos simples e também o envio dessas informações a computadores de maior dimensão e capacidade. Desse fato, derivou da Computação Ubíqua a área de pesquisa Computação Sensível ao Contexto. De acordo com DEY; ABOWD; SALBER (2001), entende-se por contexto informações sobre a situação, identidade e localização de pessoas, grupos e objetos físicos ou computacionais. Através do conhecimento de dados contextuais, uma aplicação pode ajustar seu próprio funcionamento ou ainda agir de maneira proativa, alertando o usuário de alguma situação em específico ou ajudando-o a realizar suas atividades de forma mais eficiente.

A Computação Sensível ao Contexto é uma área de muitos desafios que oportuniza muitas possibilidades de pesquisas, considerando a capacidade de reconhecer a situação atual de um usuário e o estado dos objetos, sejam eles físicos ou computacionais. Uma questão relevante de pesquisa é a aplicação dos conceitos de sensibilidade ao contexto para sistemas de *software*, bem como a definição de quais informações podem compor o contexto para artefatos e módulos no que tange o desenvolvimento e manutenção dos mesmos (PETERSEN; WOHLIN, 2009) (ANTUNES; GOMES, 2009). Com esse objetivo, é proposto um modelo para *software Analytics* baseado em históricos de contextos denominado *SW-Context*. O modelo define quais informações referentes a artefatos de *software* necessitam ser armazenadas com o objetivo de compor contextos para *software*, provendo ainda informações qualitativas baseadas nesses contextos, para que possam auxiliar desenvolvedores e times de desenvolvimento a realizar suas atividades de forma mais eficiente e assertiva.

1.1 Motivação

Desenvolvedores precisam se envolver diariamente com atividades de manutenção de *software* para adaptar aplicações existentes a novos cenários e necessidades, como por exemplo, novas funcionalidades, correções de defeitos, requerimentos legais, mudanças de hardware entre outros. RAJLICH (2001) define manutenção de *software* como a modificação de um produto de *software*, após sua entrega, com o objetivo de corrigir falhas, melhorar desempenho e outros

atributos ou adaptar o produto a modificações no ambiente. Segundo LIENTZ; SWANSON (1980) manutenção de *software* pode ser caracterizada em quatro classes: adaptativo, aperfeiçoamento, corretivo e preventivo.

Para realizar modificações desta natureza, os desenvolvedores costumam alterar o código-fonte das aplicações considerando somente sua experiência sobre o *software* em questão. Essa estratégia pode não ser suficientemente eficaz na implementação de solicitações de mudança complexas. Em parte, porque geralmente os desenvolvedores precisam manter requisitos não-funcionais, por exemplo, *performance* e segurança, cujas implementações são conhecidas por impactar amplamente as aplicações (AMELLER et al., 2012). Ou seja, as implementações dos requisitos não-funcionais acabam estendendo-se sobre os módulos do sistema, ocasionado duplicidades de códigos e dependências significativas tendo como consequência a degradação da arquitetura do sistema.

De acordo com o estudo de LAVALLÉE; ROBILLARD (2015), algumas questões organizacionais interferem diretamente nas atividades dos desenvolvedores impactando na qualidade e manutenibilidade do *software* produzido. Grande volume de documentação obsoleta, dificuldades na utilização desta documentação, muitas dependências entre módulos de *software*, conflitos entre os cronogramas de projetos e processos de desenvolvimento não entendidos pelos desenvolvedores são algumas das questões organizacionais, descritas pelo autores como impac-tantes. Os autores citam ainda fatores como especialistas, que deixam as empresas levando o conhecimento de determinados módulos e/ou sistemas, que são determinantes para o sucesso dos projetos. Deve-se considerar ainda o próprio processo de desenvolvimento de *software*, que produz vários tipos de dados como códigos-fonte, relatórios de defeitos, casos de teste e outros que compõem informações de extrema relevância a respeito do *software* em questão.

Uma das formas de mitigar o impacto dessas questões organizacionais no processo de desenvolvimento de *software*, bem como a degradação da arquitetura das aplicações, pode ser a disponibilização de informações úteis referentes a módulos ou artefatos de *software* de forma qualitativa. Segundo BAYSAL; HOLMES; GODFREY (2013), a disponibilização dessas informações propicia um melhor entendimento do desenvolvedor em relação aos aspectos que cercam o *software* e o seu ambiente. Dependendo da natureza das informações disponibilizadas, os desenvolvedores podem organizar melhor suas atividades, realizar priorizações de forma assertiva e ainda encontrar especialistas, que possam auxiliar na resolução de problemas complexos.

Entretanto, um aspecto desafiador é elencar e definir quais informações, de fato, podem contribuir para a execução de tarefas rotineiras dos desenvolvedores de *software*, se tornando úteis no processo decisório e auxiliando na prevenção da degradação da arquitetura. Segundo DEY; ABOWD; SALBER (2001), contexto é qualquer informação que possa ser usada para caracterizar a situação de entidades que são consideradas relevantes para a interação entre um usuário e uma aplicação, incluindo a própria aplicação. Dessa forma, emerge a motivação da criação de um modelo de contexto que busque caracterizar os objetos computacionais existen-

tes no desenvolvimento de *software*. Os autores definem ainda que contexto é tipicamente: a localização, identidade e o estado de pessoas, grupos e objetos físicos e computacionais.

Ao longo das últimas décadas, pesquisadores vêm utilizando técnicas da abordagem *analytics* nesses dados e informações, para um melhor entendimento da qualidade de *software* (ZHANG et al., 2013). A abordagem *analytics* fornece informações úteis, em tempo real, que auxiliam a tomada de decisão de forma proativa. As informações disponibilizadas por essa abordagem podem ser de natureza quantitativa e qualitativa. As informações quantitativas podem destacar tendência de dados, já as informações qualitativas permitem a tomada de decisão diária em tempo real (BAYSAL; HOLMES; GODFREY, 2013). Baseado nesses conceitos surge o *Software Analytics*.

Software analytics visa a obtenção de informações detalhadas e úteis de artefatos de *software* que ajudem os desenvolvedores a realizar tarefas relacionadas ao desenvolvimento e manutenção de *software* melhorando a consciência situacional¹ dos desenvolvedores (ZHANG et al., 2013). A complementação de *dashboards* quantitativos com dados qualitativos pode aprimorar o conhecimento dos desenvolvedores em relação a situação atual do ambiente e de seu contexto de trabalho.

1.2 Definição do Problema e Questão de Pesquisa

Os fatores apontados na Seção 1.1 tornam a definição de contexto para *software* uma questão desafiadora tanto na área de Computação Ubíqua quanto na Engenharia de *Software*. Diversas questões importantes devem ser considerados por sistemas que visam tratar contexto para *software*, como por exemplo:

- Quais informações devem compor o modelo de contexto para *software* e seu histórico de contexto?
- Qual forma de representação que provê informações úteis, a fim de auxiliar a tomada de decisão?
- Como as informações de contexto podem antecipar problemas ou guiar os desenvolvedores para uma solução mais correta?

Essas questões específicas nortearam o surgimento da principal questão de pesquisa deste trabalho: Como seria um modelo para *software analytics* baseado na análise de históricos de contextos, a fim de auxiliar os desenvolvedores de *software* no seu dia-a-dia?

¹Consciência situacional é um termo da psicologia cognitiva que se refere a um estado da mente onde a pessoa esta completamente consciente dos elementos de seu ambiente, possui um entendimento quanto ao significado maior do meio ambiente, e pode antecipar (ou pretende mudar) esses elementos em um futuro próximo (ENDSLEY, 1995).

1.3 Objetivo

Este trabalho de pesquisa tem como objetivo geral a especificação de um modelo para *software analytics* baseado na análise de históricos de contextos, que permita a combinação de diferentes informações relacionadas a artefatos de *software*, a fim de aprimorar a consciência situacional dos desenvolvedores nas atividades de desenvolvimento e manutenção. Para atingir este objetivo geral, são definidos os seguintes objetivos específicos:

- Avaliar os fundamentos teóricos da área;
- Identificar e comparar os trabalhos relacionados;
- Especificar o modelo de *software analytics* baseado em sensibilidade ao contexto;
- Implementar um protótipo de ferramenta que disponibilize as informações de contexto em forma de *dashboard analytics*;
- Validar o modelo a partir do protótipo.

1.4 Metodologia

Como ponto fundamental para o início desta pesquisa, procurou-se identificar quais seriam as técnicas e tópicos que oferecessem embasamento teórico para concepção de um modelo para *software analytics* baseado em sensibilidade ao contexto que atendessem aos objetivos propostos.

Após a elaboração do embasamento teórico, um experimento foi conduzido com o objetivo de avaliar a utilização de algumas informações de contexto, disponibilizadas utilizando a abordagem *software analytics*, em atividades de manutenção de *software* realizadas em uma aplicação protótipo desenvolvida por DEITEL; DEITEL (2010) e estendida pelo autor para uso específico no experimento. Esse experimento avaliou os efeitos do uso de informações de contexto em relação à corretude e manutenibilidade do código fonte da aplicação protótipo referida, bem como os efeitos no esforço de implementação das atividades propostas.

Posteriormente, partiu-se para a escrita da especificação do modelo de contextos, bem como sua arquitetura. Partindo desse embasamento teórico, do modelo experimental de contexto e da proposta de modelo, buscou-se identificar trabalhos semelhantes, para estudar as abordagens adotadas e os resultados obtidos. Realizou-se então um comparativo entre esses trabalhos de forma a identificar os pontos fortes e fracos de cada abordagem e, com isso, aperfeiçoar a especificação do modelo.

Obtida essa visão mais profunda, concluiu-se a especificação do modelo de contextos a ser desenvolvido. Com o avanço dos trabalhos, foi implementado um protótipo de ferramenta que disponibilizou informações de contexto em forma de *dashboard*. Por fim, foi realizada uma avaliação do modelo através de um estudo de caso em uma empresa de desenvolvimento de

software, onde foi possível verificar a eficácia, eficiência do modelo, bem como identificar suas limitações.

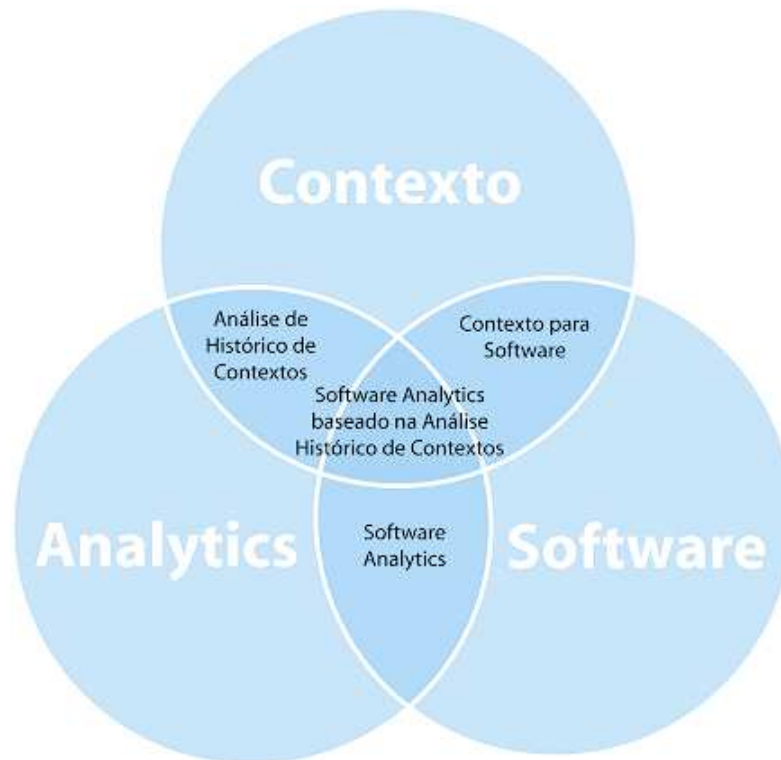
1.5 Organização do Texto

Esta dissertação está dividida em sete capítulos. No Capítulo 2 são discutidos os conceitos de pesquisa relevantes para o trabalho proposto. No Capítulo 3 são apresentados os trabalhos relacionados e um estudo comparativo de suas características, funcionalidades e limitações. O modelo proposto para *software analytics* baseado na análise de históricos de contextos, com sua arquitetura e funcionalidades é apresentado no Capítulo 4. Os aspectos de implementação do protótipo para avaliação do modelo serão descritos no Capítulo 5. No Capítulo 6, será descrita a metodologia para realização da avaliação do modelo bem como os resultados obtidos. Por fim, o Capítulo 7 contém as considerações finais e aponta os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado um resumo das referências teóricas consultadas para a elaboração desta pesquisa. A Figura 1 ilustra como os conceitos estudados se relacionam para compor o embasamento que levou a especificação do modelo proposto. A definição clássica de contexto cunhada por DEY; ABOWD; SALBER (2001) é de fundamental importância, visto que, esse conceito é aplicado nas derivações do que seria contexto para *software*. Os conceitos de *Software Analytics* norteiam esta proposta, no que diz respeito à representação visual e analítica dos dados de contexto para *software* (PETERSEN; WOHLIN, 2009; ANTUNES; GOMES, 2009).

Figura 1: Relação entre os conceitos



Fonte: Elaborado pelo autor

Este capítulo está organizado em 3 seções. A seção 2.1 descreve os conceitos e tecnologias referentes a contexto. As definições de contexto aplicadas para *software* são descritas na seção 2.2. Finalmente, a seção 2.3 elenca os principais conceitos e tecnologias relacionadas a *software analytics*.

2.1 Contexto e Histórico de Contextos

O termo contexto possui um significado intuitivo para humanos, mas por conta dessa conotação intuitiva este termo continua vago e generalista. Além disso, o interesse nas diferentes funções ou papéis de contexto vem de diferentes áreas como literatura, filosofia, linguística e

ciência da computação, onde cada área propõe sua própria visão de contexto (MOSTEFAOUI; PASQUIER-ROCHA; BRÉZILLON, 2004). Para DEY; ABOWD; SALBER (2001) contexto é qualquer informação que possa ser usada para caracterizar a situação de entidades que são consideradas relevantes para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. Contextos são, tipicamente: a localização, identidade e o estado de pessoas, grupos e objetos físicos e computacionais.

Com base nesses conceitos DEY; ABOWD; SALBER (2001) definiram uma categorização clássica para dados de contexto, que é utilizada até os dias de hoje. Eles argumentam que aplicações sensíveis ao contexto consideram **quem, onde, quando e o que** e usam essa informação para determinar qual situação está ocorrendo. Assim, os autores propuseram quatro categorias básicas para modelar contexto, que são: (1) identidade; (2) localização; (3) situação ou atividade; e (4) tempo.

Identidade refere-se a capacidade de atribuir um identificador único a cada entidade. **Localização** é mais do que simplesmente a posição atual da entidade. Pode abranger também a dados como a orientação e elevação, bem como informações que permitam deduzir as relações espaciais entre as entidades. Por exemplo, a possibilidade de determinar se um objeto está acima ou abaixo de outro. O conceito também se aplica a lugares e podem ter coordenadas relativas e absolutas. **Situação ou atividade** refere-se a características inerentes à entidade vinculada ao contexto. Para lugares, podem ser informações como a temperatura, umidade, luminosidade ou nível de ruído. Para pessoas ou grupos pode significar o estado emocional, seus sinais vitais ou ainda a atividade na qual o usuário está envolvido. Para coisas, podem ser dados de sensores ou no caso de *softwares*, podem ser atributos como o uso de CPU ou de memória. Por fim, o **tempo** identifica o momento em que os eventos ocorreram ou que os dados contextuais tenham sido capturados, permitindo a análise histórica desses eventos. Assim como a localização, o tempo possui um significado mais amplo do que aparenta inicialmente. Podendo representar um instante específico ou um intervalo, mas também ser simplesmente um valor sequencial indicando a ordem em que os eventos ou as coletas de dados do contexto ocorreram.

O conceito de histórico de contextos vem recebendo especial atenção da comunidade científica (ROSA et al., 2015; ROSA; BARBOSA; RIBEIRO, 2016). Em geral, os estudos têm em comum o fato de lidar com registros de sequências de eventos, ordenados cronologicamente e relacionados a uma entidade identificável. A diferença é o tipo de informação que é descrita nessas sequências. Um destes estudos trata sequências descrevendo a localização acompanhadas de informações associadas e uma ordem recomendada de visitação (DRIVER; CLARKE, 2004; LI; EICKHOFF; VRIES, 2012). Posteriormente, em DRIVER; CLARKE (2008) uma trilha é descrita como uma coleção de atividades a serem executadas pelo usuário, semelhante a uma *'to-do list'* mas cuja sequência de execução é ordenada conforme o contexto e as preferências do usuário ou atividades dos usuários (DRIVER; CLARKE, 2004, 2008; SMITH, 2008). Outros estudos, como SILVA et al. (2010) oferecem suporte a entidades genéricas. Essa abordagem genérica corresponde a categorização proposta por DEY; ABOWD; SALBER (2001) para

descrever o contexto de uma entidade.

A partir desses conceitos de histórico de contextos, começou a surgir a necessidade armazenar informações contextuais, gerando assim diversas bases de dados que armazenam as sequências de contextos capturados ao longo do tempo. As análises dessas bases de dados também se caracterizam como uma área de pesquisa relevante e pesquisadores vêm estudando essas oportunidades, que podem surgir a partir da análise dessas bases. ROSA; BARBOSA; RIBEIRO (2016) por exemplo, desenvolveram um modelo chamado ORACON que prevê os contextos futuros de uma entidade a partir de inferências sobre seu histórico de contextos. Já SILVA et al. (2010) chamam essas sequências de trilhas, e propõem um modelo chamado *UbiTrail*, acessado por aplicações clientes que necessitem obter e analisar dados sobre as trilhas. BARBOSA et al. (2016) estenderam o modelo *UbiTrail*, aplicando o gerenciamento e análise de tendência de históricos de contexto para a área de comércio eletrônico, através de um modelo denominado *TrailTrade*.

2.2 Contexto e Software

A interação do desenvolvedor com a IDE de desenvolvimento é considerada uma fonte de dados de contexto de *software* valiosa por KERSTEN; MURPHY (2006). Segundo os autores, cada interação do desenvolvedor com sua ferramenta de codificação provê um conjunto de informações relacionadas aos elementos e relacionamentos de artefatos de programa acessados. O armazenamento e posterior disponibilização dessas informações, em um contexto de tarefa, pode recuperar mais rapidamente o estado mental do desenvolvedor em relação às atividades que precisam ser finalizadas, melhorando assim a sua produtividade no desenvolvimento de *software*.

A estrutura dos artefatos de código também é considerada como contexto para *software*. O estudos de HOLMES; MURPHY (2005) MARTIE; HOEK (2014) definem a estrutura do código-fonte como contexto estrutural e utilizam tais informações na localização de códigos semelhantes ao que está sendo editado no momento pelo desenvolvedor. O contexto estrutural definido por ambos estudos considera elementos como assinaturas de métodos, tipos declarados, tipos de super classes e outros mais. Com base nessas informações, é possível localizar códigos semelhantes em repositórios ou *web*, auxiliando os desenvolvedores no uso de *frameworks* e *APIs* através de exemplos.

Em uma abordagem mais ampla, PETERSEN; WOHLIN (2009) propuseram um *checklist* para a descrição de contexto em *software*. Esse *checklist* considera algumas facetas ou aspectos de contexto, como: produto, processos, práticas e técnicas, pessoas, organização, e *marketing*. No centro da Figura 2, tem-se o objeto de estudo que interage com o contexto. Por exemplo, quando se tem um processo ágil como o objeto de estudo, esse processo interage com o produto determinando a forma metodologia de desenvolvimento. Adicionalmente, esse processo ágil é executado por pessoas, interage com outros processos, e é suportado por práticas, ferramentas

e técnicas. Além disso, o objeto de estudo é incorporado em uma organização. A organização está operando dentro de um mercado. Ou seja, o objeto de estudo se relaciona com as demais faces de contexto descritas.

Cada faceta de contexto compreende um conjunto de elementos de contexto e serão detalhadas a seguir:

- **Produto:** o produto é o *software* desenvolvido relacionado ao objeto de estudo. Elementos de contexto:
 - *Maturidade:* a maturidade do produto precisa ser descrita com o tempo que este está no mercado, quantas versões já foram disponibilizadas e outros;
 - *Qualidade:* esforço de desenvolvimento empreendido para melhorar a manutenibilidade do produto;
 - *Tamanho:* o tamanho do produto pode ser medido, por exemplo, considerando linhas de código, número de funcionalidades ou pontos de função. O tamanho é um indicador importante para determinar a complexidade do produto;
 - *Tipo de Sistema:* o sistema pode ser de diferentes tipos como, sistema de informação, sistema embarcado ou sistema distribuído;
 - *Customização:* isto significa se o sistema é suficientemente customizável para adaptar-se a diferentes mercados;
 - *Linguagem de programação:* descreve a linguagem de programação que o sistema foi desenvolvido.

- **Processos:** o processo descreve a ordem de execução do desenvolvimento. Elementos de contexto:
 - *Atividades:* as atividades são as diferentes etapas do desenvolvimento, por exemplo, especificação de requisitos;
 - *Workflow:* descreve a ordem em que as atividades são executadas;
 - *Artefatos:* representam o resultado das atividades, por exemplo, código-fonte.

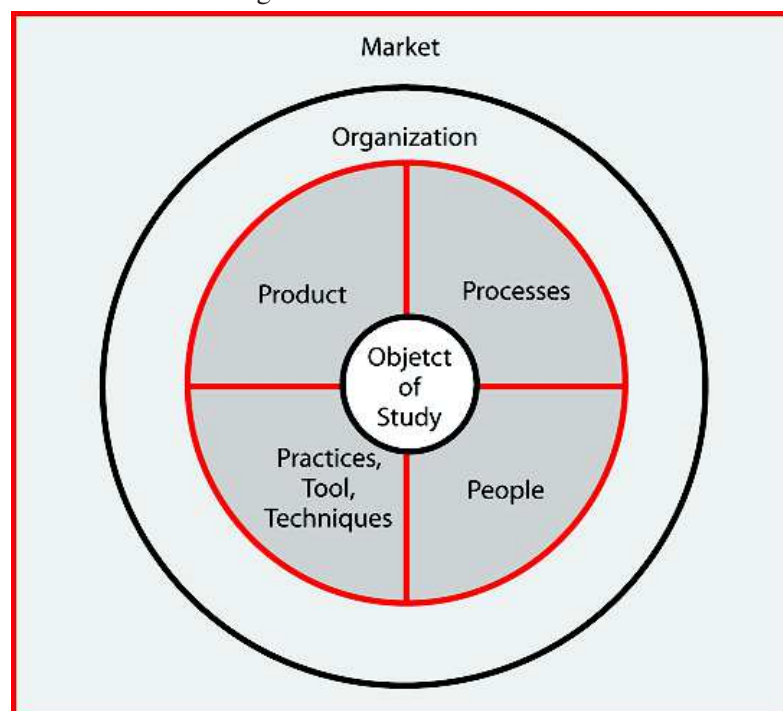
- **Práticas e técnicas:** descreve a abordagem sistemática utilizada na organização. Elementos de contexto:
 - *Ferramentas CASE:* descreve as ferramentas utilizadas para suportar ou automatizar o desenvolvimento de *software*;
 - *Práticas e Técnicas:* descreve as práticas utilizadas no desenvolvimento de *software*, como, por exemplo, reuniões de revisão de código.

- **Pessoas:** o fator humano é importante quando se estuda desenvolvimento de *software*. Elementos de contexto:
 - *Papéis:* descreve que tipo de papéis estão envolvidos como o objeto de estudo, bem como suas responsabilidades e autoridades;
 - *Experiência:* descreve a experiência das pessoas em determinadas áreas que afetam o objeto de estudo.

- **Organização:** descreve a estrutura da organização. Elementos de contexto:
 - *Modelo geral da organização:* define como a empresa está organizada, por exemplo, de forma hierárquica ou matricial;
 - *Unidade organizacional:* descreve a unidade organização mais próxima a interagir com o objeto de estudo;
 - *Certificação:* descreve se a organização é certificada (por exemplo, ISO/CMMI);
 - *Distribuição:* as unidades organizacionais estão juntas ou distribuídas (nacional ou internacionalmente).

- **Mercado:** representa os clientes e competidores. Elementos de contexto que descrevem o mercado são:
 - *Número de clientes:* descreve os clientes que já possuem contrato com a organização e potenciais clientes;
 - *Segmentos de mercado:* descreve grupos de clientes que possuem a mesma necessidade;
 - *Estratégia:* a estratégia descreve como abordar o mercado no longo prazo;
 - *Restrições:* o mercado pode colocar restrições sobre o desenvolvimento de *software*, como um tempo muito para entrega, ou necessidade de certificações.

Figura 2: Facetas de contexto



Fonte: PETERSEN; WOHLIN (2009)

ANTUNES; GOMES (2009) propõem uma abordagem que segue algumas das facetas de contexto descritas anteriormente. Porém, o objeto de estudo nesse caso seria o desenvolvedor e a informação contextual que deve ser capturada, considerando todo ambiente de trabalho que suporta seu trabalho.

Isto significa que a informação contextual deve ser recuperada de todos os aplicativos que o desenvolvedor geralmente utiliza, como uma IDE, ferramentas gerenciamento de projetos, ferramentas de gerenciamento de capital humano e até mesmo o próprio sistema operacional. Muitas dessas ferramentas já fornecem interfaces de extração de dados via *plug-in* ou APIs. A informação obtida através das várias aplicações, então, pode ser centralizada e integrada de forma coerente em um modelo de contexto, para que instantaneamente o modelo de contexto esteja disponível em qualquer lugar em tempo real.

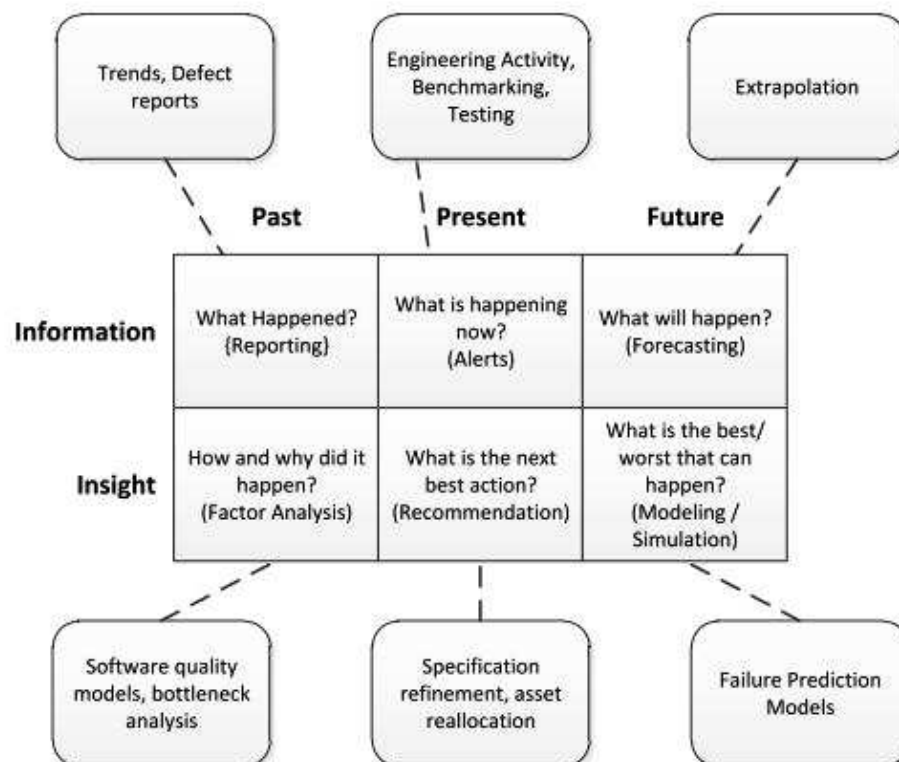
2.3 *Software Analytics*

A Engenharia de *Software* é uma atividade rica de dados. Muitos aspectos do desenvolvimento, a partir de repositórios de código podem ser medidos com um alto grau de automação, eficiência e granularidade (MENZIES; ZIMMERMANN, 2013a). Deve-se considerar ainda o processo de desenvolvimento de *software*, que produz vários tipos de dados como códigos-fonte, relatórios de defeitos, casos de teste e outros que podem compor informações de contexto. Considerando essa abundância de dados produzidos na Engenharia de *Software*, pesquisadores vêm utilizando técnicas da abordagem *analytics* nestes dados para um melhor entendimento da

qualidade de *software* (ZHANG et al., 2013).

Segundo BUSE; ZIMMERMANN (2010) *Software Analytics* pode ajudar a responder perguntas importantes dos gestores sobre os seus projetos. O objetivo é auxiliar os tomadores de decisão em extrair informações importantes e percepções a partir de conjuntos de dados que poderiam estar escondidos. Em seu estudo, foram definidas seis áreas de questionamentos analíticos organizadas em uma linha de tempo que consideram **informação** (*information*) e **percepção** (*insight*) representadas na Figura 3. A ideia é distinguir questões de **informação**, que algumas ferramentas já fornecem (por exemplo, quantos erros estão no banco de dados?), de questões de **percepção**, que fornecem aos gerentes uma compreensão da dinâmica de um projeto e uma base para tomar decisões (por exemplo, o projeto será entregue com atraso?).

Figura 3: Questões da abordagem *analytics*



Fonte: BUSE; ZIMMERMANN (2010)

Em um outro estudo, BUSE; ZIMMERMANN (2012) definem que *Software Analytics* é a análise sobre os dados de *software* para os gestores e engenheiros de *software* com o objetivo de capacitar desenvolvedores e equipes a obter e compartilhar uma visão de seus dados para tomar melhores decisões.

Segundo BAYSAL; HOLMES; GODFREY (2013), abordagens analíticas (*analytics*) ambicionam fornecer informações em tempo real, úteis e que podem ser tanto de natureza quantitativa quanto natureza qualitativa. Enquanto informações de natureza quantitativa podem destacar tendências de dados de alto nível, informações qualitativas permitem a tomada de decisões em

tempo real para tarefas do dia-a-dia.

LOU et al. (2013) definem *Software Analytics* como a utilização de abordagens *data-driven* para permitir que profissionais de *software* realizem a exploração e análise detalhada dos dados para obter informações detalhadas e úteis para completar várias tarefas em torno de sistemas de *software*, usuários de *software* e do processo de desenvolvimento de *software*.

MENZIES; ZIMMERMANN (2013b) definem *Software Analytics* como o uso de análise e raciocínio sistemático sobre os dados de *software* para gerentes e desenvolvedores obterem *insights* de seus dados, e assim, tomar melhores decisões.

2.4 Considerações sobre o capítulo

O presente capítulo apresentou os principais conceitos de contexto, bem como sua aplicação para a área de desenvolvimento de *software*. Por fim, foram abordados os conceitos de *Software Analytics* e suas tecnologias. O capítulo teve como intuito apresentar os conceitos e tecnologias utilizados como base para propor o modelo de contexto para software denominado *SW-Context*. O próximo capítulo apresenta os trabalhos relacionados ao modelo, traçando um comparativo entre suas principais características.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados ao modelo proposto de contexto para *software*, assim como um comparativo dos trabalhos. São descritos os critérios para seleção e comparação dos trabalhos e apontadas as contribuições desta dissertação, em relação aos trabalhos analisados. A Seção 3.1 descreve a metodologia para escolha dos trabalhos relacionados. Posteriormente, são apresentados os principais aspectos presentes em cada um dos trabalhos, seguido de um comparativo entre os mesmos.

3.1 Metodologia para escolha dos trabalhos

O foco da pesquisa dos trabalhos relacionados levou em consideração a definição de DEY; ABOWD; SALBER (2001) sobre contexto que é "qualquer informação que possa ser usada para caracterizar a situação de entidades relevantes para a interação entre um usuário e uma aplicação, e que em geral, é composto por dados que descrevem a situação, identidade e localização espaço-temporal de pessoas, grupos e objetos físicos ou computacionais". A partir dessa definição foram pesquisados estudos que abordassem esse conceito na área de desenvolvimento de *software*. Ou seja, foram considerados trabalhos que abordam informações relacionadas ao *software* a fim de caracterizá-lo em alguma determinada situação ou estado. Outro critério de pesquisa utilizado, na busca de artigos, foi a forma de utilização e representação dessas informações.

Buscou-se, ainda, trabalhos relacionados que representassem algum modelo implementável, a fim de permitir a comparação como modelo proposto, sendo esse modelo implementável um *plugin*, protótipo, *framework*, ferramenta ou algoritmo. Esta seção apresenta doze trabalhos, os quais possuem como foco a utilização de informações relacionadas ao *software* e/ou seu processo de desenvolvimento para caracterização de alguma situação ou estado do mesmo.

O *International Workshop on Context for Software Developers*¹ foi a primeira fonte de trabalhos utilizada na pesquisa. Nesse *workshop* foram encontradas referências relevantes que permitiram pesquisas mais focadas com o intuito de obter trabalhos com objetivos semelhantes aos desta dissertação. Adicionalmente foram pesquisados trabalhos nas bases de periódicos da CAPES, *Scholar* – Google acadêmico, *Elsevier* – *Science Direct*, *IEEE* – *Xplore* e *ACM* – *Digital Library Springer* – *International Publishing AG*. Tais pesquisas foram baseadas nos seguintes termos: “*Software Context*”, “*Software Development*” e “*Software Analytics*”.

3.2 *Mylyn a Plugin to deal with task context*

KERSTEN; MURPHY (2006) propõem um *plugin* denominado *Mylyn*² que considera a

¹<http://csd-ws.github.io/2014.html>

²<http://www.eclipse.org/mylyn/>

interação do desenvolvedor com a IDE Eclipse a origem dos dados que compõem o contexto de tarefa. Um grafo de elementos e relacionamentos de artefatos de programa compõem o contexto de tarefa, segundo os autores, é a informação necessária que o desenvolvedor precisa conhecer para finalizar uma determinada tarefa. Para cada elemento e relacionamento do modelo é atribuído um coeficiente de relevância (*degree-of-interest*) em relação à tarefa corrente. Baseado nessa informação, o *plugin* cria visões do projeto exibindo somente os artefatos necessários (calculados a partir do coeficiente de relevância) para a conclusão de determinada tarefa. A alternância de tarefas é o principal foco de atuação do *plugin*, uma vez que somente os artefatos considerados mais relevantes são exibidos. O estado mental do desenvolvedor é, de certa forma, recuperado mais rapidamente melhorando assim a produtividade. Entretanto, o estudo considera contexto para *software* somente a interação do desenvolvedor com a IDE. As demais informações que fazem parte do ambiente de trabalho, como, por exemplo informações de projetos ou incidentes que demandam alterações em código, não são abordadas. O armazenamento das informações de cada interação do usuário é realizado em uma estrutura denominada *interaction event data* (Tabela 1). Conjuntos das interações compõem o *interaction history* que é utilizado para o cálculo do coeficiente de relevância.

Tabela 1: Evento de interação

Elemento	Descrição
<i>Time</i>	Identificação de tempo que o evento ocorreu
<i>Kind</i>	Classificação do evento: seleção, edição, execução de comando, propagação e predição
<i>Origin</i>	Identificação do componente de tela ou ferramenta que gerou o evento
<i>Content Type</i>	Identificador do tipo de elemento que está sendo manipulado
<i>Handle</i>	Identificação do elemento destino
<i>Delta</i>	Modificação que ocorreu com o evento

Fonte: Adaptado de KERSTEN; MURPHY (2006)

3.3 *Build usage contexts during program comprehension*

Na mesma linha de gerenciamento de atividades, PARNIN; GÖRG (2006) propõem igualmente uma abordagem de captura de contexto de atividades a partir da interação dos desenvolvedores com uma IDE. O foco do trabalho foi capturar o contexto de interação do desenvolvedor em classes e métodos, ou seja, especificamente para o paradigma de programação orientada a objetos. As categorias de interação com a IDE são diferentes das definidas por KERSTEN; MURPHY (2006) no trabalho descrito anteriormente:

- **Navegação:** comandos utilizados para ir a um determinado ponto ou posição dentro de um arquivo;

- **Clique:** seleção de um método via mouse em um editor de código;
- **Edição:** alguma modificação em uma linha de código;
- **Consulta:** busca pela localização de um determinado método.

A partir do armazenamento dos eventos de interação, cria-se um histórico de contexto que é utilizado posteriormente para recomendação do desenvolvedor sobre quais métodos e classes foram acessados previamente. A ideia é que a recomendação recupere o contexto relacionado à atividade que estava sendo executada de forma mais rápida.

O conceito foi avaliado através de um estudo de caso realizado com dez empregados de uma empresa que se voluntariaram a participar do estudo. Os projetos participantes do estudo de caso foram escritos em C++ e C# e possuem um tamanho que varia de 50 a 200 mil linhas de código. A IDE utilizada pelos participantes foi o *Visual Studio*, através de um *plugin* denominado *InteractionDB*.

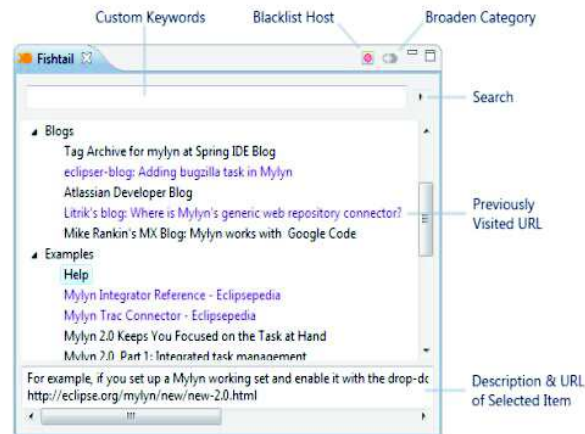
3.4 *Fishtail* a Plugin to automate the discovery of relevant code examples from the web

O histórico da interação entre o desenvolvedor, programas e artefatos também foi utilizado por SAWADSKY; MURPHY (2011) para descobrir exemplos de códigos relevantes da *web* pelo *plugin Fishtail*. O objetivo dessa implementação é sugerir de forma automática exemplos de código-fonte, a partir da *web*, que são relevantes para a funcionalidade que o programador está tentando implementar. Esse *plugin* foi desenvolvido como uma extensão da arquitetura do contexto de tarefa construído por KERSTEN; MURPHY (2006) e disponibilizado na IDE Eclipse.

Baseado no histórico de contexto das interações do desenvolvedor são derivadas palavras chaves para busca de exemplos de código.

Os principais componentes da arquitetura do *Fishtail* são:

- **um gerenciador de contextos de tarefa** para rastrear o coeficiente de relevância dos elementos de programa;
- **um componente gerador de consultas** que cria palavras chaves para os elementos de programa em uma determinada atividade;
- **um componente de execução de consultas** que é responsável por enviar consultas a um motor de buscas e priorizar os resultados; e
- **um componente de exibição** exibe resumos dos resultados das buscas dentro da IDE Eclipse (Figura 4).

Figura 4: Visão do *Fishtail* dentro do Eclipse

Fonte: SAWADSKY; MURPHY (2011)

3.5 *Recommending Task Context: Automation Meets Crowd*

LEANO; KASI; SARMA (2014) definem contexto de tarefa utilizando uma abordagem híbrida onde são consideradas a identificação automatizada de contextos bem como a contribuição de desenvolvedores experientes. Além disso, os autores partem da premissa que os desenvolvedores gastam muito tempo identificando que arquivos precisam ser alterados para que uma tarefa de desenvolvimento seja iniciada. Ou seja, os arquivos que serão editados também devem ser considerados como parte do contexto de tarefa.

A ideia da abordagem híbrida é uma combinação de captura de informações, mineração de dados e técnicas de análise textual para aprimorar a acurácia na recomendação de contextos de tarefa. Para determinar o conjunto completo de arquivos que precisam ser alterados, algumas abordagens podem ser utilizadas segundo os autores:

- ***Commit Graphs***: análises empíricas demonstram que arquivos que possuem dependências são, normalmente, envidados conjuntamente para repositórios de projetos. Os grafos que representam os envios de código são uma fonte de dados excelente para capturar as dependências de artefatos;
- ***Crowdsourcing***: o aproveitamento da sabedoria das multidões tem se tornado uma realidade em *sites* como o *Wikipedia*, onde todo o conteúdo é produzido por usuários que podem adicionar, editar e moderar o conteúdo. O *Crowdsourcing* também tem sido bem sucedido em casos onde os não-especialistas foram capazes de resolver rapidamente um problema que os especialistas não conseguiram;
- ***Informações de especialistas***: desenvolvedores experientes ou especialistas são a melhor fonte de identificação de contextos de tarefas, entretanto seu tempo é muito valioso e o

tempo necessário para a triagem de um erro pode ser muito alto. A ideia é criar uma interface para que os especialistas possam avaliar resultados automáticos validando as informações geradas por técnicas automáticas de maneira simples.

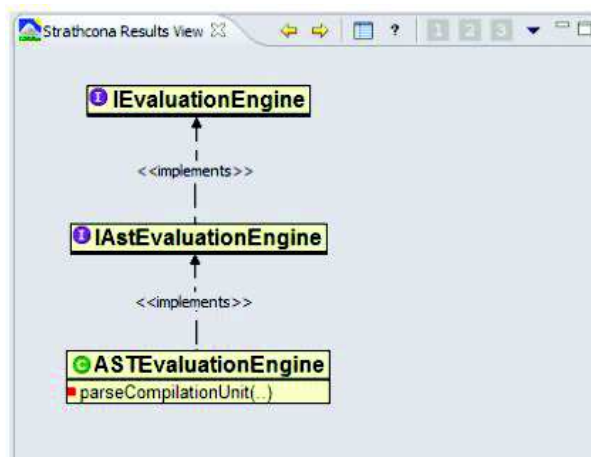
A motivação para utilização da abordagem híbrida é que técnicas unicamente automatizadas de identificação de contexto não possuem uma boa precisão (de 11% a 21%). A abordagem híbrida, que combina técnicas automatizadas e a contribuição humana, pode aumentar os benefícios de cada uma das técnicas.

3.6 Using structural context to recommend source code examples

HOLMES; MURPHY (2005) definiram uma abordagem de localização de códigos-fonte relevantes em um repositório de exemplo utilizando o contexto estrutural do código que está sendo editado na IDE. A ideia principal desse estudo é a de facilitar a utilização de *frameworks* e APIs facilitando a busca de exemplos de implementação. O contexto estrutural definido pelos autores considera os seguintes elementos: assinaturas de métodos, tipos de declarados, tipos de super classes, métodos chamados, nomes de atributos acessados e tipos instanciados referenciados por cada método.

Com o objetivo de avaliar o conceito, os autores implementaram um *plugin* para a IDE Eclipse denominado *Strathcona*, que trabalha extraíndo o contexto estrutural dos artefatos de código. O contexto estrutural extraído pode ser utilizado de duas formas: como uma consulta automática que descreve um fragmento de código que o desenvolvedor precisa de suporte e também para montagem de um banco de dados que contém o contexto estrutural das classes.

O funcionamento do *plugin* pode ser descrito através de uma sequência de atividades que se inicia quando o desenvolvedor solicita exemplos de código e o contexto estrutural do código que está sendo editado é enviado para o servidor. O servidor, por sua vez, ao receber o contexto estrutural, aplica algumas consultas no banco de dados do repositório com o objetivo de encontrar o mesmo contexto estrutural enviado na consulta. Se encontrado o contexto estrutural no repositório, esse é retornado para o desenvolvedor (Figura 5).

Figura 5: Resultado do *Strathcona* dentro do Eclipse

Fonte: HOLMES; MURPHY (2005)

A avaliação da implementação foi realizada através de um estudo de caso em que desenvolvedores realizaram quatro atividades de implementação, onde três dessas atividades utilizavam partes do *framework* Eclipse. Para tanto, foi construído um repositório de exemplo com os contextos estruturais deste *framework* com 1.316.204 entradas no banco de dados que foi utilizado como base para o experimento.

3.7 *Harnessign the Crowd: Descontextualizing Software Work*

Tradicionalmente o desenvolvimento de *software* requer que os desenvolvedores aprendam o contexto do seu trabalho antes de efetivamente contribuir para os seus projetos. Ao receberem uma determinada funcionalidade para implementar os desenvolvedores precisam saber o local correto em que a alteração precisa ser feita, precisam conhecer o contexto das funcionalidades já implementadas bem como a arquitetura envolvida. Ao escrever uma determinada função, os desenvolvedores precisam entender o contexto que esta será utilizada. Quando os desenvolvedores usam uma função já existente, precisam entender qual deve ser o estado do sistema e os efeitos que esta chamada pode causar. Essa informação é necessária para que o desenvolvedor consiga implementar suas alterações de forma consistente com a arquitetura e desenho do projeto.

LATOZA; TOWNE; HOEK (2014) citam observações realizadas que sugerem que desenvolvedores não necessitam ter um entendimento global da arquitetura do projeto para lidar com decisões de implementação mais complexas. Segundo as hipóteses dos autores, grande quantidade dessa informação de contexto necessária pode ser capturada das interfaces de funções.

Dessa forma, a implementação da funcionalidade requerida pode ser realizada de forma modularizada sobre essas funções através de tarefas menores ou micro-tarefas. Essa abordagem permite que desenvolvedores consigam contribuir para o projeto de forma mais rápida, não sendo necessário recrutar especialistas para tarefas complexas, uma vez que estas estarão

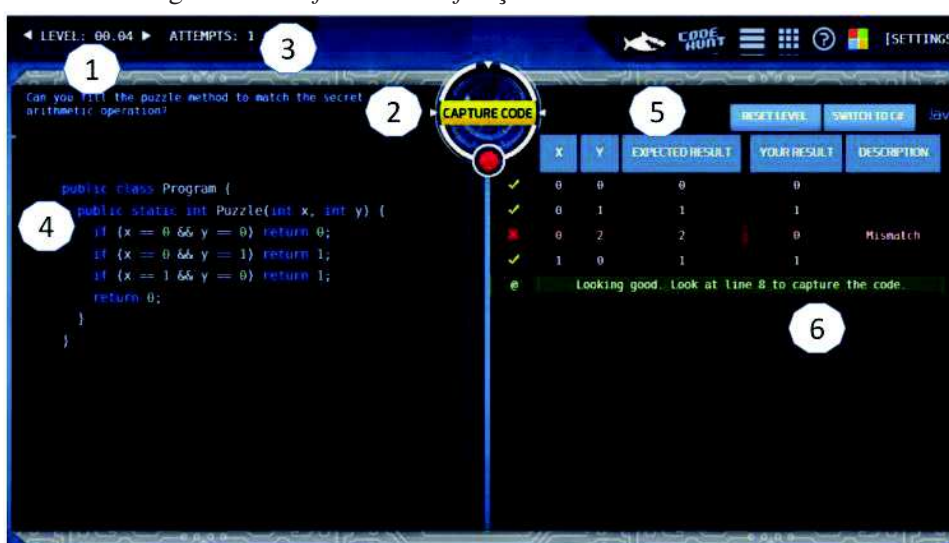
descontextualizadas em micro-tarefas.

3.8 Code Hunt: Context-Driven Interactive Gaming for Learning Programming and Software Engineering

*Code Hunt*³ é uma plataforma web de jogos onde os jogadores podem resolver desafios de programação avançando em níveis. TILLMANN et al. (2014) realizaram um estudo para identificar que tipos de contexto de *software* existem na ferramenta e como essas informações auxiliam os desenvolvedores a resolver os duelos de programação.

Os tipos de contexto identificados pelos autores foram classificados como visíveis e invisíveis. Além dessa classificação, as informações foram categorizadas conforme segue: duelos previamente resolvidos, o segmento de código secreto do duelo, o histórico duelos do jogador, o código que está sendo alterado no duelo atual, os pares de entrada e saída do jogador e as dicas reportadas para jogador. A Figura 6 mostra os tipos de contextos identificados por números.

Figura 6: Interface de codificação de duelos do Code Hunt



Fonte: TILLMANN et al. (2014)

A seguir são detalhadas as categorias de contexto, bem como a indicação da disposição na ferramenta proposta pelos autores:

- **Duelos previamente resolvidos** (identificado como 1 na Figura 6): um duelo de código pode ser designado para o jogador baseado em um duelo resolvido anteriormente;
- **O segmento de código secreto do duelo** (identificado como 2 na Figura 6): este segmento de código é escondido intencionalmente do desenvolvedor, que precisa descobrir a funcionalidade do código escondido baseado em um *feedback* de forma iterativa;

³<https://www.codehunt.com/>

- **O histórico duelos do jogador** (identificado como 3 na Figura 6): tipicamente o jogador realiza uma série de tentativas (formando um histórico de duelos) antes de resolver o duelo de forma correta;
- **O código que está sendo alterado no duelo atual** (identificado como 4 na Figura 6): é o código do duelo que o desenvolvedor está tentando resolver no momento;
- **Os pares de entrada e saída do jogador** (identificado como 5 na Figura 6): identifica o *status* de sucesso e/ou falha dos casos de teste;
- **As dicas reportadas para jogador** (identificado como 6 na Figura 6): além dos pares de entrada e de saída mostrados ao jogador, a ferramenta, por vezes, também exibe uma dica para indicar qual linha de código o jogador pode querer modificar a fim de fazer progressos no sentido de resolver o duelo de codificação.

TILLMANN et al. (2014) traçam um paralelo dos tipos de contextos encontrados na ferramenta com as informações mais tradicionais do desenvolvimento de *software*. A Tabela 2 mostra essas equivalências entre as informações.

Tabela 2: Paralelo de tipos de contexto

Tipo de Contexto da Ferramenta	Contexto no Desenvolvimento de Software
O histórico duelos do jogador	Histórico de alterações de artefatos de código
O segmento de código secreto do duelo e os pares de entrada e saída	Requisitos
Duelos previamente resolvidos	Tarefas dependentes
As dicas reportadas para jogador	Discussões e compartilhamento de conhecimento sobre atividades e artefatos

Fonte: Elaborado pelo autor

3.9 Context in Code Search

MARTIE; HOEK (2014) usam as informações de contexto na tarefa de busca de código *online*. Algoritmos, exemplos de linguagem, e exemplos de uso de APIs são muito buscados na *web*. A busca normalmente se inicia colocando palavras chaves em motores de busca. Na maioria das vezes, a partir dos resultados obtidos, os programadores modificam as palavras chaves e submetem a consulta novamente. Esse processo se repete até que o programador encontre o código desejado.

Os autores desenvolveram um novo *engine* de busca de código chamado *CodeExchange* com o objetivo de suportar justamente essa reformulação/refinamento de consultas e interação com os resultados. Os resultados das consultas são exibidos em um contexto interativo com riqueza de informações onde o programador pode realizar buscas dentro dos contextos resultantes de

consultas anteriores. Esse contexto inclui meta-dados e propriedades semânticas do resultado de código, recomendações de consulta e os histórico das consultas.

O desenvolvedor pode iniciar sua consulta digitando palavras-chave ou utilizando a busca avançada (Figura 7), onde é possível detalhar os critérios com informações de contexto relevantes como, propriedades da classe, localização, chamadas de métodos e declarações de método.

Figura 7: Tela de busca avançada do CodeExchange

The screenshot shows a web form titled "Find classes with:". It is organized into four main sections, each with a header and several input fields:

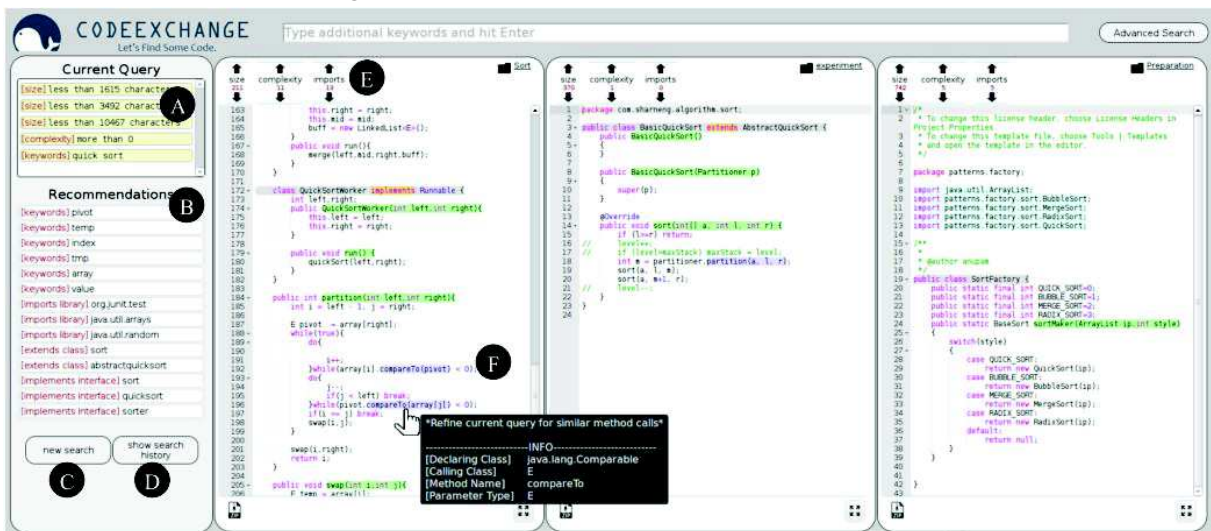
- Properties:** Includes fields for "Imports" (example: `java.io.File`), "Extends" (example: `java.util.observable`), and "Implements" (example: `java.util.observer`). It also has checkboxes for "generic" and "has wildcards".
- Location:** Includes fields for "Package" (example: `com.selimober.marsrovers`) and "Project" (example: `calico`).
- Method Call:** Includes fields for "Class" (example: `java.util.hashmap`), "Method" (example: `put`), and "Parameters" (example: `java.lang.String, int[], ...`).
- Method Declaration:** Includes fields for "Class" (example: `Bootstrap`), "Method" (example: `sort`), "Parameters" (example: `java.lang.String, int[], ...`), and another "Method" field (example: `java.lang.String`). It also has checkboxes for "generic" and "variable args".

At the bottom of the form are two buttons: "Cancel" and "Submit".

Fonte: MARTIE; HOEK (2014)

Após submeter a consulta, a ferramenta exibe os resultados obtidos de repositórios *GitHub* de forma interativa. Os parâmetros da consulta atual são exibidos na tela (Figura 8, A). Alguns resultados são exibidos com as propriedades semânticas destacadas (Figura 8, F) que permitem o refinamento da consulta com um simples clique. Os resultados também apresentam meta-dados relacionados à criticidade (Figura 8, E) de forma que o desenvolvedor pode refinar a consulta selecionando códigos com mais/menos caminhos críticos (*branches*), ou com mais/menos uso de bibliotecas (*imports*). A ferramenta apresenta também recomendações de consultas (Figura 8, B) baseado nas palavras chave da consulta atual. O usuário pode iniciar uma nova pesquisa (Figura 8, C) e visualizar o histórico de consultas realizadas anteriormente por ele (Figura 8, D).

Figura 8: Resultado interativo do *CodeExchange*



Fonte: MARTIE; HOEK (2014)

Como próximos passos, os autores pretendem realizar comparações da sua ferramenta com o *Google* e *GitHub search engine* em diferentes tipos de buscas por código.

3.10 StackMine: a Microsoft Project

StackMine (ZHANG et al., 2013) é um projeto da *Microsoft* que auxilia na identificação do alto impacto de execução de programas e problemas de *performance*. A motivação para o projeto veio da necessidade de suportar a infraestrutura de sistemas *web* da *Microsoft* e realizar a depuração e resolução de problemas de *performance*. O insumo inicial dos engenheiros de *software* era uma coleta de *logs*, realizada por uma ferramenta chamada *PerfTrack*, que identificava a sequência e o tempo de execuções de funções e enviava para análise. Todas as sequências de execuções coletadas poderiam conter mais de 1 bilhão de chamadas de funções, excedendo em muito o que um analista de *performance* consegue investigar.

Os autores consideram que, baseado nessa quantidade de dados, a depuração de *perform-*

mance pode ser considerada como um problema de *software analytics*. Dado esse montante de dados coletados, com a abordagem *analytics* poderia auxiliar os analistas a efetivamente descobrir gargalos de *performance*. Baseado nessas questões, foi desenvolvida a ferramenta *StackMine* que associa tempos de execução altos em sistemas à sequências de chamadas de funções ocorridas. Aplicando a abordagem *analytics*, a ferramenta proporciona informações úteis (*actionable information*) que permitem a rápida identificação dos problemas.

Segundo os desenvolvedores que utilizam a ferramenta, o projeto auxiliou-os nas análises de problemas de *performance* reduzindo, em alguns casos, até 90% do trabalho de investigação humana. A construção da ferramenta *StackMine* foi realizada em um processo iterativo entre os pesquisadores e engenheiros de *software* onde, o *feedback* constante foi uma informação valiosa para as melhorias no produto, inclusive na usabilidade do mesmo.

3.11 *Test and Defect Coverage Analytics Model*

HARON; SYED-MOHAMAD (2015) propuseram um modelo chamado TDCAM (*Test and Defect Coverage Analytics Model*), que possui o foco na integração de dados de cobertura de teste e resolução de defeitos para ajudar os gestores a tomar decisões. Esses dados são extraídos de artefatos de código-fonte em teste, casos de teste de ferramentas de controle de qualidade e bancos de dados de controle de defeitos. Depois de analisar esses dados, o modelo mostra o resultado através de um *dashboard* denominado *Test Analytics Dashboard*. Esse *dashboard* mostra a relação entre o número de defeitos, o índice de cobertura de testes e o tamanho dos módulos de *software* em linhas de código. A forma utilizada para representar a relação entre as informações citadas foi um gráfico de bolhas (Figura 9), aplicando uma abordagem qualitativa. O modelo foi implementado como um *plugin* da ferramenta Eclipse, e sua avaliação foi realizada por um estudo de caso sobre o Apache POI (um projeto *open source* desenvolvido pela Apache). O modelo proposto foi comparado com algumas ferramentas de cobertura de teste, tais como Clover⁴ e SonarQube⁵.

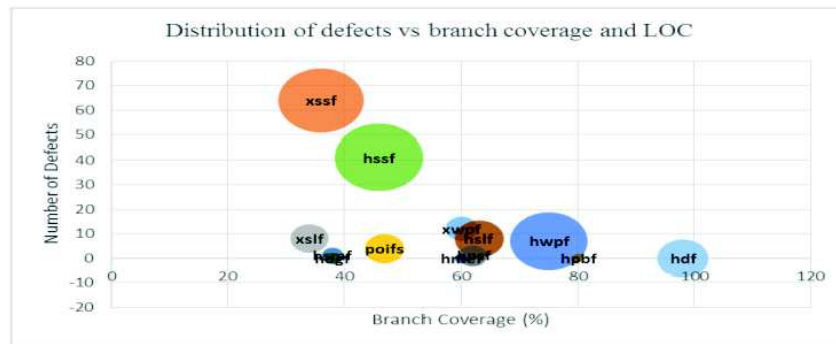
3.12 *Context-Based Retrieval in Software Development*

ANTUNES; GOMES (2009) propõem uma abordagem para capturar informação contextual de um desenvolvedor considerando todo o ambiente que o suporta no seu trabalho. Isso significa que as informações contextuais devem ser recuperadas de todos os aplicativos que o desenvolvedor geralmente utiliza, como uma IDE, um conjunto de ferramentas de escritório, um PIM (*Personal Information Manager*) e até mesmo o sistema operacional. A maior parte das ferramentas referidas proporcionam algum tipo de *plugin* de integração, o que facilita em grande parte a recuperação da informação. A informação recolhida através das várias aplica-

⁴<https://br.atlassian.com/software/clover>

⁵<https://www.sonarqube.org/>

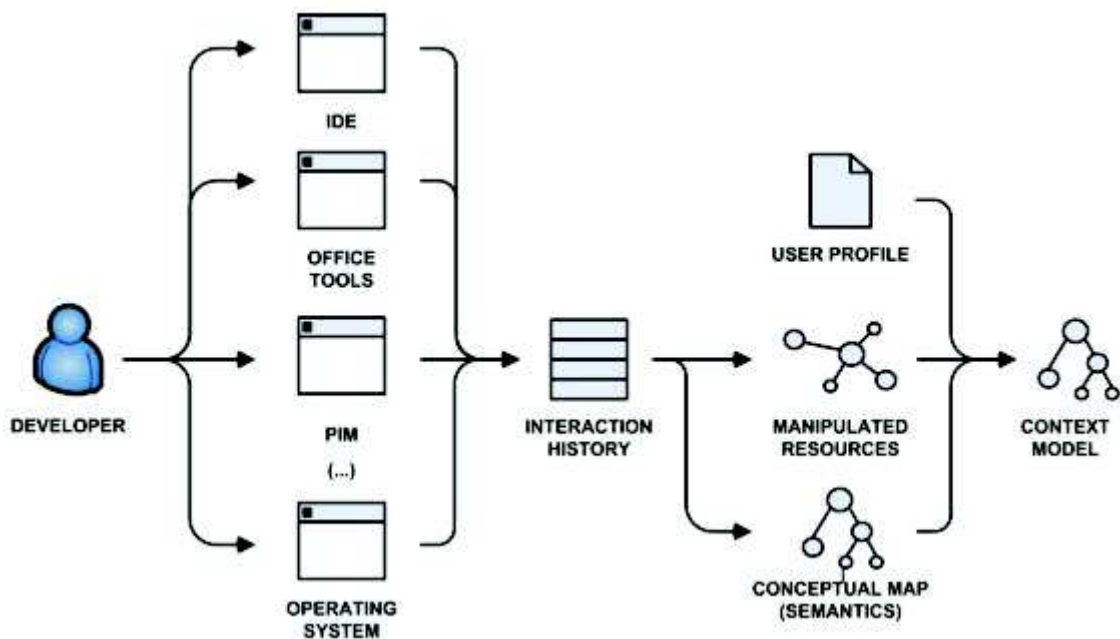
Figura 9: Gráfico de bolhas do modelo TDCAM



Fonte: HARON; SYED-MOHAMAD (2015)

ções deve, então, ser centralizada e integrada de forma coerente a um modelo de contexto, para que um instantaneamente o modelo de contexto do usuário esteja disponível em qualquer lugar. A Figura 10 representa a abordagem de captura de contexto proposta pelos autores.

Figura 10: Abordagem para captura de contexto proposta



Fonte: ANTUNES; GOMES (2009)

Um dos desafios dos autores, assim como desta dissertação, é definir o modelo de contexto para *software*. O trabalho apresenta apenas uma ideia geral da abordagem de captura de dados, sem definir exatamente que informações devem ser armazenadas e como estas seriam disponibilizadas.

3.13 Context Capture in Software Development

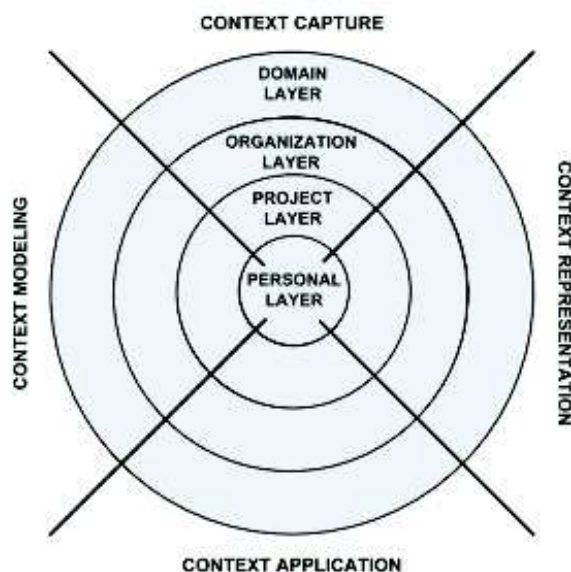
ANTUNES; CORREIA; GOMES (2011) propuseram um modelo de contexto para *software* que considera todas as dimensões que caracterizam o ambiente de trabalho do desenvolvedor. Essas dimensões são representadas em um modelo de quatro principais camadas: pessoal, projeto, organização e domínio. Conforme é representado na Figura 11, essas dimensões formam um modelo de camadas que será descrito em quatro diferentes perspectivas: captura de contexto, modelagem de contexto, representação de contexto e aplicação de contexto.

As quatro dimensões de contexto definidas pelo estudo serão detalhadas a seguir:

- **Camada pessoal:** representa o contexto do trabalho que o desenvolvedor possui, mais especificamente as atividades que o mesmo desenvolve diariamente. Para realizar essas tarefas o desenvolvedor precisa lidar com vários tipos de recursos ao mesmo tempo, como códigos fonte, documentos de especificação entre outros;
- **Camada de projetos:** representa pessoas e recursos, assim como suas relações no projeto de *software* o qual o desenvolvedor está envolvido;
- **Camada de organização:** leva em consideração o contexto organizacional que o desenvolvedor pertence. Similar a um projeto, uma organização é feita de pessoas, recursos e suas relações, mas em uma escala muito maior de complexidade. Deve-se considerar também que os conhecimentos e competências desenvolvidos em cada projeto podem ser interessantes em outros projetos;
- **Camada de domínio:** leva em consideração o conhecimento de domínio em que o desenvolvedor atua. Esta camada vai além dos níveis e organização do projeto e inclui um conjunto de fontes de conhecimento que se destacam fora destas esferas. Hoje em dia, um desenvolvedor típico usa a *web* para procurar informação e manter-se informado sobre os avanços nas tecnologias em que ele trabalha. Essas ações são baseadas em serviços e comunidades, como repositórios de código-fonte, fóruns de desenvolvimento, *sites* de notícias, blogs, etc. Essas fontes de conhecimento não podem ser separadas do contexto desenvolvedor e estão integradas na camada de domínio do modelo de contexto definido pelos autores.

As quatro dimensões de contexto detalhadas anteriormente podem ser descritas através de diferentes perspectivas: a captura de contexto, que representa as fontes de informação de contexto e a forma como essa informação é recolhida, a fim de construir o modelo de contexto; a modelagem de contexto, que representa as diferentes dimensões, entidades e aspectos da informação contextual; a representação do contexto, que representa as tecnologias e estruturas de dados usadas para representar o modelo de contexto; e a aplicação do contexto, que representa o modo como o modelo de contexto é usado e os objetivos além sua utilização.

Figura 11: Modelo de contexto proposto



Fonte: ANTUNES; CORREIA; GOMES (2011)

Os autores desenvolveram um protótipo, em forma de *plugin* para a IDE *Eclipse*, com o objetivo de demonstrar como informações de contexto podem ser integradas em ferramentas de desenvolvimento do *software* e ajudar os desenvolvedores. A Figura 12 mostra um *screenshot* do protótipo onde é exibida uma *Eclipse view* denominada "*Context*" que demonstra informações relacionadas a um recurso específico. Cada vez que o desenvolvedor abre um recurso (código-fonte) a *Eclipse View* é atualizada com a lista de desenvolvedores, recursos, e atividades que estão relacionadas a este. Dessa forma, o desenvolvedor pode obter mais informações sobre o recurso, como por exemplo atividades realizadas anteriormente e também que desenvolvedores podem fornecer mais informações que auxiliem na conclusão da atividade atual.

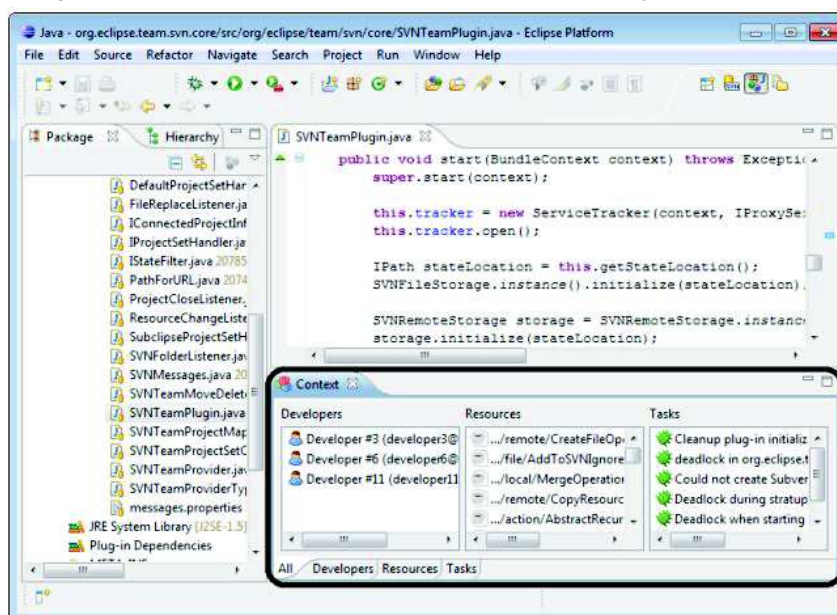
A disponibilização dessas informações dentro da IDE de desenvolvimento aumenta a consciência dos desenvolvedores sobre a atividade que precisa ser executada diminuindo o esforço de busca de informações que poderiam estar ocultas ou dispersas em outras ferramentas.

3.14 Análise comparativa dos trabalhos relacionados

Nesta seção é feita uma comparação dos trabalhos relacionados (Tabela 3) em relação à definição de contexto para desenvolvimento de *software*, bem como a utilização e representação destas informações. Foram avaliados seis grupos de características, destacando-se como o mais importante **Definição de modelo de contexto**, tendo em vista que um dos principais desafios deste estudo é a definição de que informações devem ser consideradas na composição de um modelo de contexto para *software*. Sendo assim, os grupos de características utilizados na comparação dos trabalhos relacionados foram os seguintes:

- **Definição de contexto:** grupo de características que avalia se o estudo definiu formal-

Figura 12: *Plugin no Eclipse demonstrando informações de contexto*



Fonte: ANTUNES; CORREIA; GOMES (2011)

mente que informações compõem contexto para *software* e qual a natureza das informações utilizadas. Os estudos foram avaliados em relação aos seguintes critérios:

1. Formalização da definição do modelo de contexto: este critério considera se o estudo descreve uma definição formal de contexto aplicado ao desenvolvimento de *software*, como por exemplo, definição de classes, diagrama de domínio ou ontologia;
2. Contexto de artefatos de *software*: usualmente informações capturadas da interação entre desenvolvedores, IDEs e artefatos de *software* bem como seus relacionamentos - heranças, elementos, dependências, e outros;
3. Contexto relacionado aos desenvolvedores: informações relacionadas a interação do desenvolvedor de *software* com vários dos tipos de recursos em sua atividade diária, como, por exemplo, documentos de especificação, relatórios de defeitos, cultura organizacional, metodologias de desenvolvimento e outros;
4. Contexto relacionado a projetos de *software*: um projeto de desenvolvimento de *software* pode ser considerado uma agregação de um time, um conjunto de recursos e a combinação de conhecimentos explícitos e implícitos que mantém o projeto em andamento. Normalmente, projetos possuem cronogramas apertados, escopos agressivos e outras restrições e premissas que são de extrema relevância para o processo de desenvolvimento de *software*;
5. Contexto relacionado à operação do *software*: informações sobre o número de clientes que usam determinada funcionalidade, a interação entre a funcionalidade e a

infraestrutura computacional e *performance* de execução, que podem ajudar o desenvolvedor de *software* a obter um entendimento mais completo sobre o contexto da aplicação que está sendo alterada ou avaliada no processo de manutenção de código.

- **Tipo de estudo:** verifica qual o tipo de estudo foi abordado no trabalho conforme os critérios a seguir:
 1. Estudo comparativo;
 2. Estudo de caso;
 3. Experimento controlado;
 4. Estudo de mapeamento.

- **Modelo proposto:** avalia qual o modelo de contexto proposto do estudo, considerando:
 1. Arquitetura;
 2. Modelo Conceitual;
 3. Ontologia;
 4. Algoritmo.

- **Ferramenta desenvolvida:** grupo que identifica se foi desenvolvida alguma ferramenta no estudo e se essa ferramenta foi concebida conforme os critérios abaixo:
 1. Protótipo;
 2. Sistema;
 3. *Plugin*;
 4. Heurística.

- **Licença:** se alguma ferramenta ou protótipo foi desenvolvido, é verificada a disponibilidade da mesma para a comunidade conforme a licença, podendo ser:
 1. *Software* Livre;
 2. Privado.

- **Visualização:** este grupo descreve como os dados de contexto são exibidos nas ferramentas desenvolvidas, considerando:
 1. *Analytics*: dados qualitativos e úteis que permitem decisões em tempo real;
 2. *Dashboards*: dados quantitativos demonstrando tendências e históricos;

3. Dados Estruturados: informações estruturadas que permitem a captura de dados via APIs e etc;
4. Visualizações específicas, *views*: visualizações especializadas em determinadas ferramentas ou IDEs.

A Tabela 3 mostra a comparação realizada nos estudos apresentados anteriormente, considerando os critérios adotados como parâmetros para esta pesquisa. Esses critérios foram assumidos como suportados pelos modelos quando citados explicitamente ou quando indícios relevantes de sua presença foram identificados.

Tabela 3: Tabela comparativa dos trabalhos relacionados

Estudo/ Critérios	Definição de Contexto				Tipo de Estudo				Modelo Proposto				Ferramenta			Licença		Visualização				
	Definição Formal	Relacionado a artefatos	Relacionado a desenvolvedores	Relacionado a projetos	Relacionado a <i>software operations</i>	Estudo Comparativo	Estudo de Caso	Experimento Controlado	Estudo de Mapeamento	Arquitetura	Modelo Conceitual	Ontologia	Algoritmo	Protótipo	Sistema/Ferramenta	<i>IDE Plugin</i>	Heurística	<i>Open Source</i>	Privada	<i>Analytics Dashboards</i>	Dados Estruturados	<i>Views</i>
HARON; SYED-MOHAMAD (2015)	-	+	+	-	-	-	+	-	-	+	-	-	-	-	-	+	-	∅	∅	+	-	-
LEANO; KASI; SARMA (2014)	~	+	+	-	-	-	-	-	-	-	+	-	-	-	-	-	+	∅	∅	-	-	-
TILLMANN et al. (2014)	-	-	-	-	-	-	-	-	-	-	+	-	?	-	+	-	+	+	-	-	+	-
MARTIE; HOEK (2014)	~	+	+	-	-	-	-	-	-	+	-	-	-	∅	+	-	-	+	-	-	+	+
LATOZA; TOWNE; HOEK (2014)	~	+	+	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	-	-	+	+
ZHANG et al.(2013)	~	+	+	-	+	-	+	-	-	∅	∅	∅	+	∅	+	∅	∅	-	+	+	+	-
SAWADSKY; MURPHY (2011)	-	+	+	-	-	-	-	-	-	-	-	-	+	+	-	+	-	∅	∅	-	+	+
ANTUNES; CORREIA; GOMES (2011)	~	+	+	+	-	∅	∅	∅	∅	+	+	+	+	+	-	-	-	∅	∅	-	+	+
ANTUNES; GOMES (2009)	-	+	+	+	-	-	~	-	-	?	?	?	-	-	-	-	-	∅	∅	∅	∅	∅
KERSTEN; MURPHY (2006)	+	+	+	-	-	-	+	-	-	+	-	?	+	-	-	+	+	+	-	-	+	+
PARNIN; GÖRG, 2006	~	+	+	-	-	-	+	-	-	-	-	-	+	∅	∅	∅	∅	∅	∅	∅	∅	∅
HOLMES; MURPHY (2005)	-	+	+	-	-	-	+	-	-	-	-	-	+	-	-	+	+	+	-	-	+	+

Legenda: (+) Suportado (-) Não suportado (~) Suportado parcialmente (∅) Não se aplica

Fonte: Elaborado pelo autor

Conforme indica a tabela comparativa, todos os estudos consideram as informações de artefatos de *software*, suas relações, dependências e a definições de interfaces como contexto. A utilização dessas informações visa o aumento de produtividade dos desenvolvedores, através de recomendações de código pelo estudo de SAWADSKY; MURPHY (2011) ou pela recuperação de contexto de trabalho, conforme o estudo de KERSTEN; MURPHY (2006). Somente o estudo de KERSTEN; MURPHY (2006) define formalmente contexto, entretanto sua definição se restringe à interação do desenvolvedor com a IDE de desenvolvimento, através do *plugin Mylyn*, desconsiderando outros aspectos como informações de projetos ou *software operations*.

O estudo de ZHANG et al. (2013), através da ferramenta *StackMine*, considera as informações de *software operations* como contexto ao utilizar o resultado da avaliação de *performance* de execução de programas para acelerar a análise de problemas em ambientes produtivos de sistemas. Outro fator considerado pelo estudo, é a análise de padrões de execuções de funções para o mapeamento de gargalos de desempenho. A ferramenta desenvolvida, avalia possíveis padrões de execução que apresentam tempos elevados de execução e os indica aos desenvolvedores com o objetivo de facilitar a identificação dos problemas.

Apenas os estudos de ANTUNES; CORREIA; GOMES (2011) e ANTUNES; GOMES (2009) abordam contexto para desenvolvimento de *software* de uma forma mais ampla, considerando não somente informações relacionadas aos artefatos de código. Os estudos consideram que as informações de contexto devem conter mais do que dados relativos à interação do desenvolvedor com sua IDE de desenvolvimento. Deve-se considerar aspectos pessoais, organizacionais e também de projetos onde os requerimentos, premissas e restrições compõem o conjunto de informações capaz de aprimorar a produtividade dos desenvolvedores. A dimensão de *software operations* não é considerada pelos estudos citados, bem como não há uma definição formal ou modelo de contexto em forma de ontologia ou diagrama de classes/domínio. Nestes estudos são apontadas abordagens de captura de informações e conceitos genéricos.

Em relação ao tipo de estudo adotado pelos trabalhos relacionados, ZHANG et al. (2013), HARON; SYED-MOHAMAD (2015), PARNIN; GÖRG (2006) e HOLMES; MURPHY (2005) estruturaram suas pesquisas através de estudos de caso. Já KERSTEN; MURPHY (2006) conduziram a execução de um experimento controlado em seu artigo.

Considerando o critério de modelo proposto, chama a atenção o fato de nenhum dos estudos definir uma ontologia ou representação de classes para contexto. A maioria dos estudos incluíram algoritmos como característica do seu modelo.

Em termos de visualização de informações, a maioria dos estudos abordam dados estruturados e *views* em seus modelos. Com exceção da ferramenta StackMine (ZHANG et al. (2013)) e do modelo TDCA (HARON; SYED-MOHAMAD (2015)), que disponibilizam as informações utilizando a abordagem *analytics* permitindo aos desenvolvedores e gestores a tomada de decisão em tempo real.

A partir do comparativo realizado entre os trabalhos relacionados foi possível identificar as seguintes oportunidades de pesquisa, as quais serão exploradas como contribuições científicas: a definição formal de contexto para desenvolvimento de *software*, o armazenamento de históricos de contextos de *software*, análise e disponibilização de históricos de contextos utilizando a abordagem *Software Analytics*.

No que diz respeito à definição formal de contexto para desenvolvimento de *software*, serão consideradas, além de informações relacionadas a artefatos, as dimensões de pessoas, de projetos e *software operations*. Ao se considerar uma abordagem mais ampla na definição de contexto, o modelo *SW-Context* poderá ampliar a consciência dos times de desenvolvimento a respeito das informações contextuais que cercam os artefatos de *software*.

A partir da definição formal de contexto para *software*, cria-se a possibilidade da aplicação da sensibilidade ao contexto, característica essa que irá avaliar as modificações contextuais dos artefatos que possam indicar a degradação da arquitetura de uma determinada aplicação. A análise dessas modificações poderá desencadear algumas ações e alertas aos desenvolvedores sobre mudanças contextuais da aplicação consideradas como relevantes.

Conforme citado anteriormente, a análise de históricos de contexto utilizando a abordagem *Software Analytics* também se caracteriza como uma questão de pesquisa relevante. Oportuniza-

se a criação de um modelo de análise e visualização que forneça informações úteis, em formato de *dashboard*, que auxiliem a tomada de decisão dos desenvolvedores durante suas atividades de manutenção de *software*. A utilização da abordagem *Software Analytics* permitirá a obtenção de percepções e dados, possivelmente implícitos no histórico de contexto armazenado, que indiquem relações de causa e efeito para as alterações relevantes das informações contextuais. Será possível avaliar, por exemplo, quando a arquitetura de uma determinada aplicação entrou em processo de degradação e que outros aspectos podem ter influenciado nesta questão.

3.15 Considerações sobre o capítulo

O presente capítulo apresentou uma breve descrição dos trabalhos relacionados ao objeto desta dissertação, tendo como objetivo identificar os principais aspectos relacionados aos modelos propostos, traçando um comparativo entre os mesmos. Este comparativo permitiu a identificação de oportunidades para as contribuições na definição de contexto para desenvolvimento de *software*. O próximo capítulo apresenta o modelo *SW-Context*, detalhando sua arquitetura e principais componentes.

4 MODELO SOFTWARE CONTEXT

Este capítulo apresenta o *SW-Context*, um modelo que define as informações de contexto para *software*, armazenando-as em histórico de contextos. Os históricos de contextos são disponibilizados de forma qualitativa, a fim de auxiliar o dia-a-dia dos desenvolvedores de *software* utilizando a abordagem *Software Analytics*. Para isso, a Seção 4.1 descreve alguns aspectos conceituais do modelo, bem como sua arquitetura e suas principais características. A Seção 4.2 apresenta os requisitos que serão implementados. Por fim, a Seção 4.3 introduz os principais componentes de *software* do modelo e suas relações.

4.1 Visão geral

O objetivo deste modelo é prover uma abordagem de *Software Analytics*, com informações qualitativas, baseado no histórico de contextos de *software*, a fim de auxiliar a tomada de decisão dos desenvolvedores de *software* nas suas atividades de manutenção.

Para tanto, os principais desafios do modelo são: a definição de quais informações devem compor o contexto para *software*, o armazenamento estruturado destas informações em históricos de contextos e, finalmente, a análise e disponibilização destas informações de contexto, de forma que possam auxiliar a atividade de desenvolvimento e manutenção de *software*, utilizando o conceito *Software Analytics*. O modelo provê interfaces para a entrada de dados de diferentes fontes/aplicações e centraliza essas informações, de forma a poder efetuar análises e disponibilizar visualizações qualitativas bem como alertas e avisos aos desenvolvedores de *software*.

Supondo um cenário onde um time de desenvolvimento queira controlar a degradação da arquitetura do *software* a partir da complexidade ciclomática¹. Para suportar este cenário, o modelo permite o armazenamento histórico das informações contextuais da aplicação e dos valores limites para a métrica de complexidade ciclomática. Com base nesses limites, é possível configurar alertas que indiquem que um determinado artefato de *software* (uma classe ou um programa) está perto de atingir o limite estipulado para a métrica. O modelo avalia as informações contextuais, atuais, referentes a essa métrica e compara as informações com os parâmetros configurados. De acordo com as parametrizações realizadas, executa a ação previamente parametrizada, como, por exemplo, envio de alertas via e-mail. A partir das análises realizadas no histórico de contextos da aplicação, o valor dessa métrica fica disponível também na interface de visualização qualitativa das informações, junto com as demais informações que compõem o contexto do artefato em questão.

A Figura 13 mostra a arquitetura do modelo, que está distribuída em camadas e módulos. No item 1 da Figura 13 pode-se ver a camada de serviços de entrada de dados, que permite a

¹Métrica de software desenvolvida por MCCABE (1976) que mede a quantidade de caminhos de execução independentes a partir de um código-fonte.

recepção de diferentes tipos de informações de contexto. Essas informações são usualmente oriundas de diferentes bases de dados e sistemas externos e possuem relação com os artefatos de *software* e desenvolvedores. O módulo de contexto (item 2 da Figura 13) é responsável por armazenar essas informações recebidas no repositório de histórico de contextos (item 4 da Figura 13) e avaliação das possíveis alterações contextuais dos artefatos realizando ações conforme parametrizações realizadas no módulo de configuração. O módulo de configuração (item 3 da Figura 13) do modelo permite a parametrização de métricas de *software*, consideradas fundamentais para a preservação da arquitetura da aplicação em questão, e a associação de ações relacionadas a esses parâmetros. Essas informações serão armazenadas no banco de dados de configuração denominado *Config* (item 4 da Figura 13).

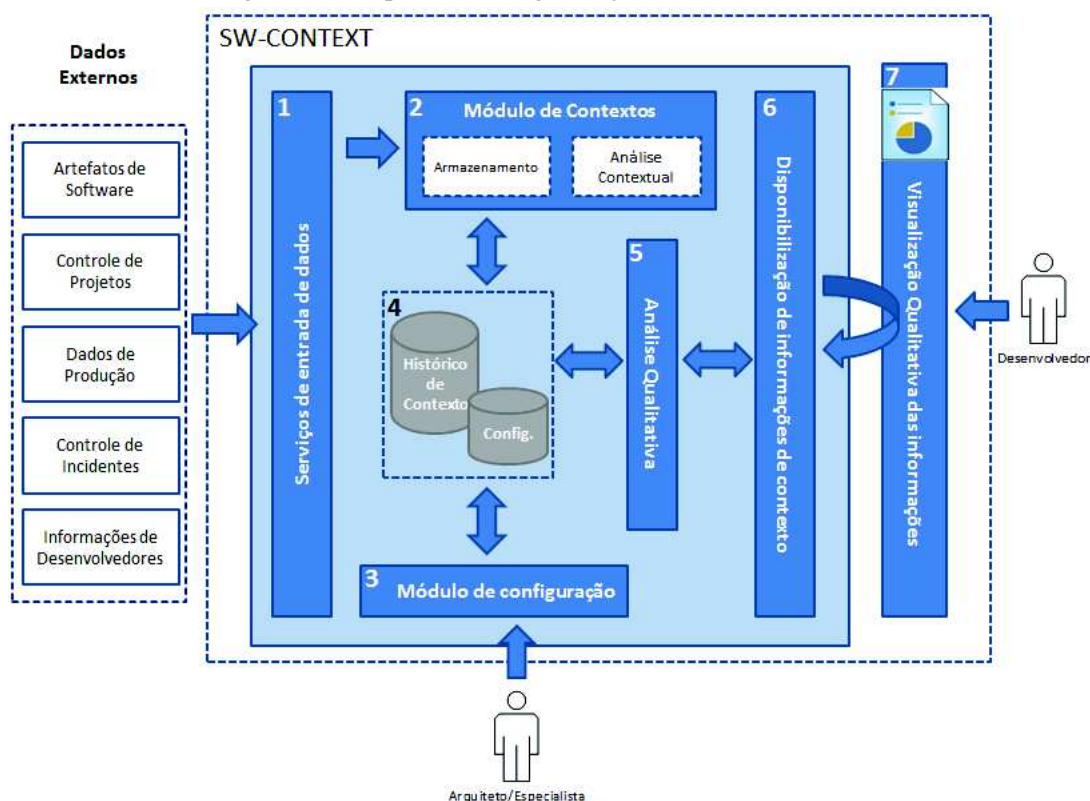
Por meio dessas configurações, o time de desenvolvimento representa suas decisões de projeto, através dos valores limites para algumas métricas de código, a fim de assegurar as melhores práticas de desenvolvimento. Por exemplo, pode haver a definição de que o índice de cobertura de testes unitários para classes deve ser no mínimo 75% ou ainda que a média de linhas de código de métodos deve ser de no máximo 50 linhas. A partir desses valores limites, o time de desenvolvimento pode monitorar a evolução da arquitetura da aplicação e tomar ações, quando a tendência das métricas estiver perto dos valores configurados.

A camada de análises qualitativas (item 5 da Figura 13) é responsável pela aplicação das técnicas de *Software Analytics* no histórico de contextos dos artefatos de *software*. Essa camada efetua algumas análises e correlações a fim de gerar percepções adicionais aos desenvolvedores de *software*. É possível relacionar o histórico de métricas da aplicação com as demais dimensões de contexto previstas no modelo. Este módulo, permite, por exemplo, a identificação dos desenvolvedores que atuaram no artefato no momento do tempo em que alguma métrica obteve uma alteração relevante de valores.

O resultado das análises qualitativas fica disponível para que a camada de disponibilização das informações de contexto (item 6 da Figura 13) possa realizar a leitura e disponibilização para o componente de visualização do modelo (item 7 da Figura 13). A camada de visualização é responsável por disponibilizar as informações contextuais avaliadas qualitativamente, em um formato amigável ao usuário, através de *dashboards* gráficos.

Desta forma, as principais características presentes no modelo são:

- **Modelo de contexto para *software*:** define a estrutura de dados (entidades) onde as informações contextuais ficam armazenadas considerando diferentes dimensões de informação;
- **Histórico de contextos para *software*:** possibilita o armazenamento histórico de todas as informações contextuais dos artefatos de *software* conforme o modelo definido, viabilizando análises e correlações;
- ***Software analytics* baseado no histórico de contextos:** efetua análises no histórico de contextos armazenado no modelo correlacionando todas as dimensões de informação do

Figura 13: Arquitetura e organização modelo *SW-Context*

Fonte: Elaborado pelo autor

contexto a fim de fornecer informações úteis aos desenvolvedores de *software*.

4.2 Requisitos

Para que o modelo atenda aos objetivos propostos, os seguintes requisitos são necessários:

1. **Interface de entrada de dados:** permitir a entrada dos dados de contexto dos artefatos de software conforme a categorização definida nas dimensões: artefatos, projetos, pessoal, manutenção e operação;
2. **Parametrizações:** permitir a configuração de parâmetros relacionados à manutenibilidade dos artefatos de *software* (complexidade ciclomática, número máximo de linhas por método e cobertura de testes unitários) e ações associadas a estes parâmetros, como por exemplo, envio de alertas. Este requerimento permite aos times de desenvolvimento a representação das suas decisões de projeto no modelo;
3. **Análise de mudanças contextuais:** avaliar as modificações contextuais armazenadas periodicamente em relação aos parâmetros configurados e processar as ações configuradas caso as alterações sejam consideradas significativas. A prevenção da degradação da arquitetura das aplicações é endereçada por este requisito, onde as decisões de projeto pa-

rametrizadas são confrontadas com a situação da aplicação em um determinado instante de tempo;

4. **Análise qualitativa do histórico de contexto:** avaliar o histórico de contextos de *software* correlacionando as informações com o objetivo de fornecer informações qualitativas (*Software Analytics*) com visões e percepções úteis ao dia-dia dos desenvolvedores;
5. **Interface de saída de dados para visualização:** permitir a obtenção dos dados de contexto armazenados no modelo e informações qualitativas para disponibilização em diferentes formatos;
6. **Camada de visualização de informações:** disponibilizar as informações de contexto e dados qualitativos, através de *dashboards*, que permitam aos desenvolvedores de *software* a descoberta oportunidades ocultas, identificação de relações de causa efeito e tomada decisões mais precisas em relação às suas atividades diárias.

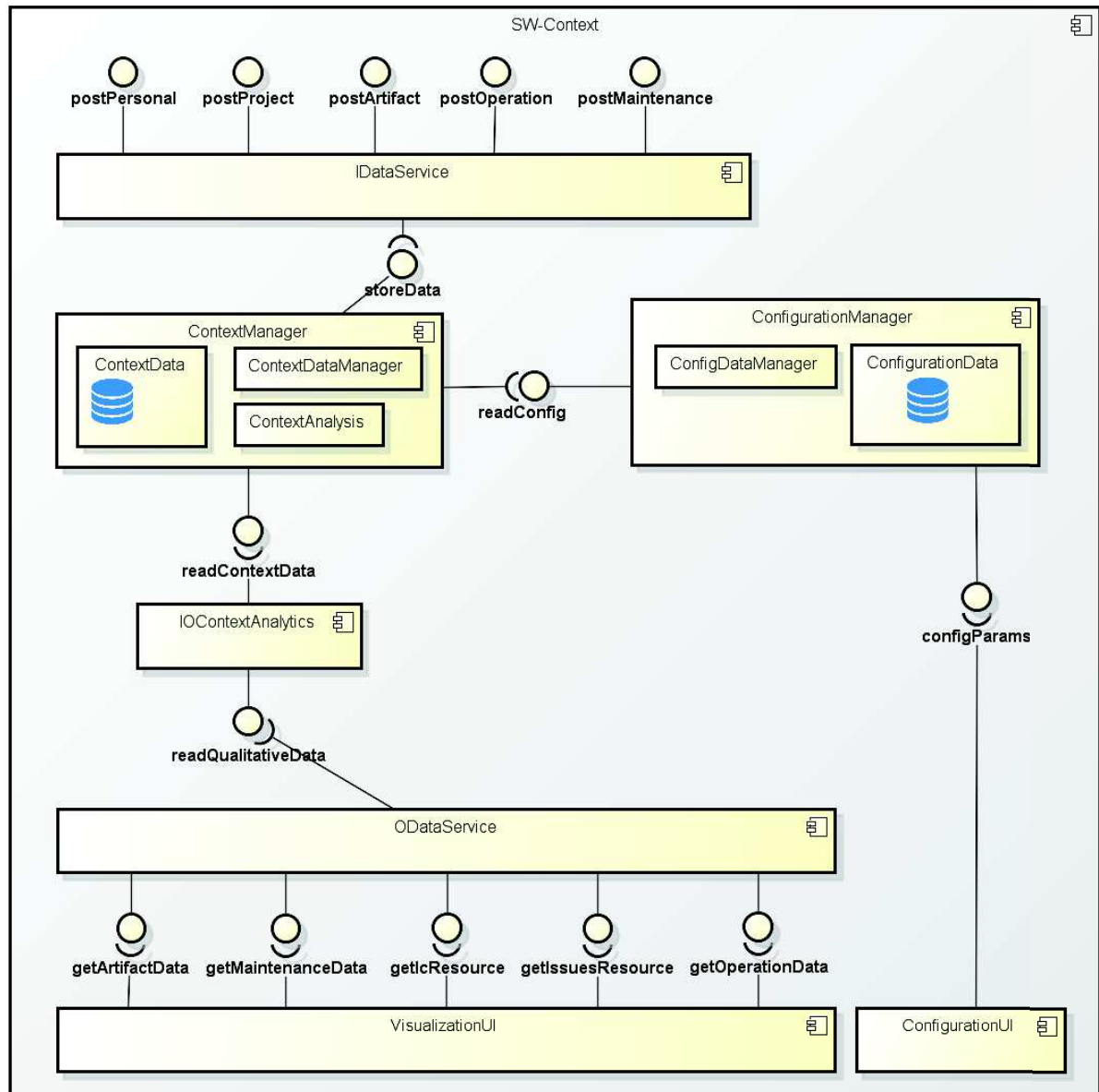
4.3 Componentes do Modelo *SW-Context*

As camadas do modelo proposto estão representadas na Figura 14, através de um diagrama de componentes. Este diagrama detalha as principais interfaces de comunicação entre os componentes, suas relações e funções. O modelo está organizado em sete componentes principais: *IDataService*, *ContextManager*, *ConfigurationManager*, *ContextAnalytics*, *ODataService*, *VisualizationUI* e *ConfigurationUI*. Adicionalmente, dois repositórios de dados compõem o modelo. O primeiro deles, conectado ao componente *ContextManager*, contém as informações de contexto coletadas periodicamente. Já o segundo repositório, relacionado ao componente *ConfigurationManager*, contém as informações de configuração do modelo. Neste repositório são armazenados os valores de métricas parametrizadas pelos times de desenvolvimento e as ações a serem realizadas a partir dos mesmos.

O componente *IDataService* é responsável por expor cinco interfaces RESTful² de entrada de dados que se referem a cada uma das dimensões de contexto descritas na Seção 4.3.2. Este componente se comunica com o *ContextManager* que gerencia as informações de contexto em relação à gravação e consulta dos dados. O componente *ContextManager* é o cerne do modelo, visto que todas as operações de atualização e consulta serão de sua responsabilidade. Outra responsabilidade deste componente é a avaliação das mudanças contextuais dos artefatos, em relação ao histórico de contextos, conforme as configurações administradas no componente *ConfigurationManager*.

²Transferência de Estado Representacional (REST do inglês *Representational State Transfer*) é o estilo de arquitetura que permite o desenvolvimento de sistemas de baixo acoplamento, utilizando o protocolo HTTP. Tradicionalmente, o design de serviços baseados em REST (ou serviços Web RESTful) tem sido descrito em termos de identificadores uniformes de recursos (URIs do inglês Uniform Resource Identifier), métodos HTTP suportados por cada recurso, e suas representações.

Figura 14: Diagrama de componentes do modelo



Fonte: Elaborado pelo autor

O componente *ODataService* provê uma interface que recuperação dos dados de contexto de um determinado artefato com métodos RESTful. Esta interface é utilizada pelo componente de visualização dados *VisualizationUI* e a partir das requisições desse, se dará a comunicação com o *IOContextAnalytics*, a fim de obter as informações qualitativas de contexto. A avaliação do histórico de contextos com o objetivo da obtenção de percepções e informações qualitativas é de responsabilidade do componente *IOContextAnalytics*. Este componente correlaciona todas as dimensões de contexto de um artefato a fim de descobrir possíveis relações de causa e efeito em relação às modificações contextuais.

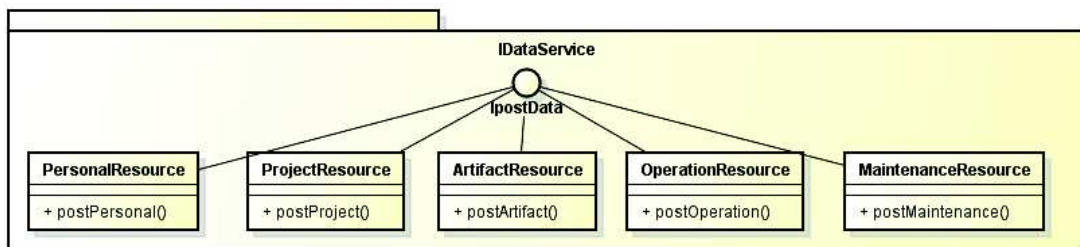
O componente *VisualizationUI* é responsável por apresentar os dados de contexto ao usuário final do modelo. Estes dados são apresentados de forma qualitativa por meio de um *dash-*

board. A interface de configurações do modelo se dá a partir do componente *ConfigurationUI* onde o usuário informa os valores limites para cada uma das informações a serem consideradas como modificações contextuais relevantes para tomada de alguma ação.

4.3.1 Componente *IDataService*

O componente *IDataService* é responsável pela entrada de dados do modelo. Através dele são disponibilizadas cinco interfaces (recursos) que permitem o armazenamento das informações de cada uma das dimensões de contexto definidas no modelo. As cinco classes, que implementam as interfaces (Figura 15), encaminham as requisições de inserção de dados ao componente *ContextManager* para o devido armazenamento dos dados no banco de dados.

Figura 15: Diagrama de classes do componente *IDataInput*



Fonte: Elaborado pelo autor

4.3.2 Componente *ContextManager*

A atribuição de armazenar as informações de forma estruturada no banco de dados de contextos é do componente *ContextManager*. Outra responsabilidade deste componente é avaliar as modificações contextuais em relação aos parâmetros de manutenibilidade armazenados e disparar ações se necessário.

A definição das informações de contexto para *software* aborda as principais dimensões que caracterizam o ambiente de trabalho do desenvolvedor. As dimensões de informações consideradas pelo modelo são: artefatos de *software*, pessoal, projetos, operação e manutenção. Algumas destas dimensões foram definidas como fundamentais pelo estudo de ANTUNES; CORREIA; GOMES (2011) e outras são oriundas das demais definições e trabalhos relacionados apresentados na Seção 3 desta dissertação.

A Figura 16 mostra a relação entre as dimensões consideradas, colocando artefato de *software* como centro do modelo caracterizando assim a entidade principal proposta. As demais dimensões possuem as informações de contexto que caracterizam o artefato, como, por exemplo: pessoas, projetos, manutenção e, por fim, operação.

As cinco dimensões de informações consideradas fundamentais para o contexto de *software* do modelo (Figura 16) serão detalhadas a seguir:

- **Dimensão de artefatos:** informações como tamanho, identificação, métricas, método, sub-rotina, componente, módulo ou aplicação de *software* são consideradas parte do contexto. As informações das demais dimensões terão relação direta com os artefatos;
- **Dimensão de projetos:** esta dimensão será composta pelas informações dos projetos que demandaram implementações nos artefatos. Requerimentos, metodologia de desenvolvimento, premissas, restrições entre outros serão considerados na composição do contexto;
- **Dimensão pessoal:** o histórico dos desenvolvedores que realizaram alguma modificação nos artefatos, seu perfil, cargo que ocupa, localização física e identificação de equipe são informações que irão compor a dimensão pessoal do modelo de contextos;
- **Dimensão de operação:** o tempo de execução de uma determinada funcionalidade em ambientes produtivos, o número de clientes que utiliza a funcionalidade ou aplicação bem como a frequência de utilização fornecem informações relevantes sobre o artefato compondo o seu contexto de forma complementar;
- **Dimensão de manutenção:** a lista das últimas modificações no artefato, seja por projeto ou correção de problemas, pode ser considerada com um histórico de eventos que ocorreram no artefato fornecendo uma visão evolutiva do mesmo e contribuindo para a formação do contexto.

Figura 16: Dimensões consideradas no modelo

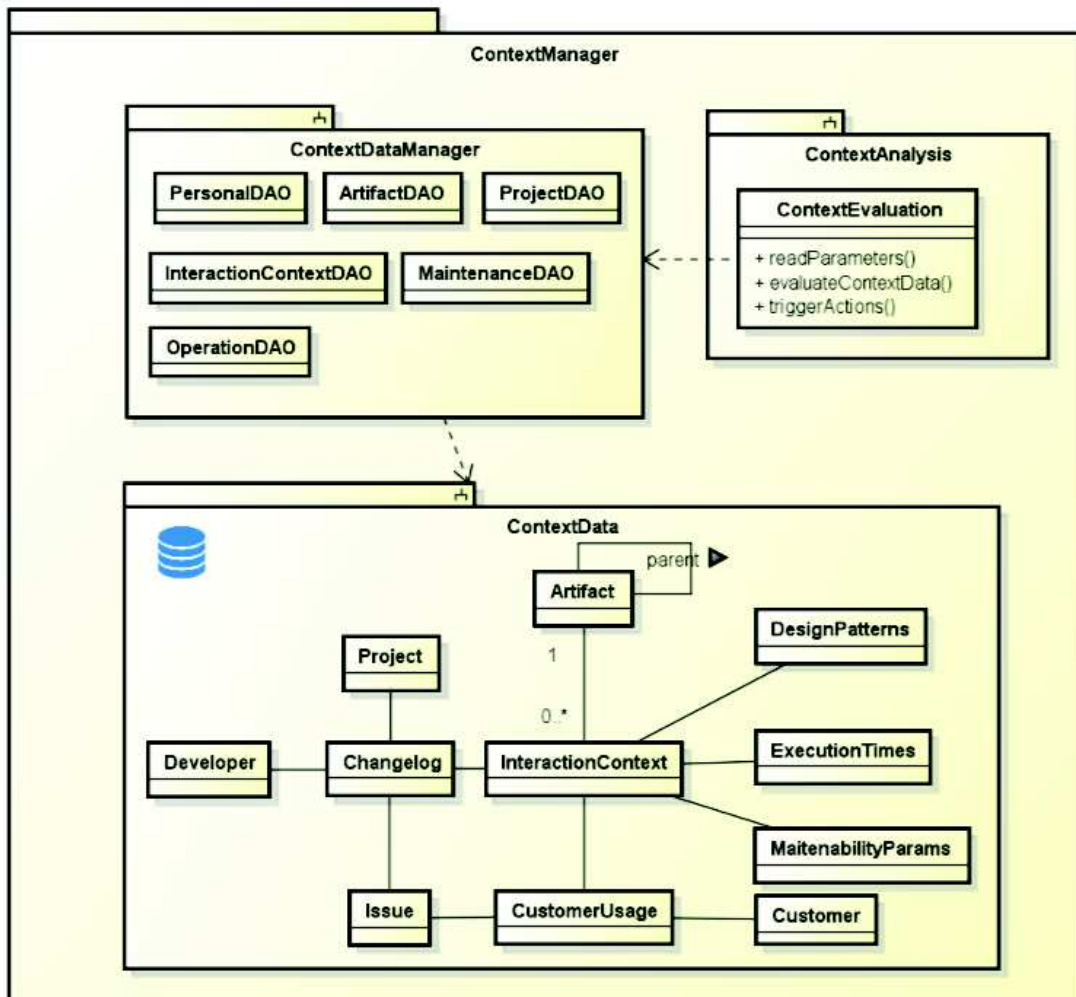


Fonte: Elaborado pelo autor

A Figura 17 representa os principais módulos do componente *ContextManager*, bem como suas classes. O módulo *ContextDataManager* contém as classes responsáveis pela administração dos dados de contexto do modelo. Essas classes implementam as principais operações de leitura de persistência dos dados.

As decisões de projeto dos times de desenvolvimento poderão estar representadas pelos limites de valores de métricas configurados no componente *ConfigurationManager*. O módulo *ContextAnalysis* realiza a avaliação das modificações contextuais dos artefatos e de acordo com os limites configurados e executa as ações (envio de alertas), igualmente configuradas, a fim de prevenir a degradação da arquitetura da aplicação.

Figura 17: Diagrama de classes do componente *ContextManager*

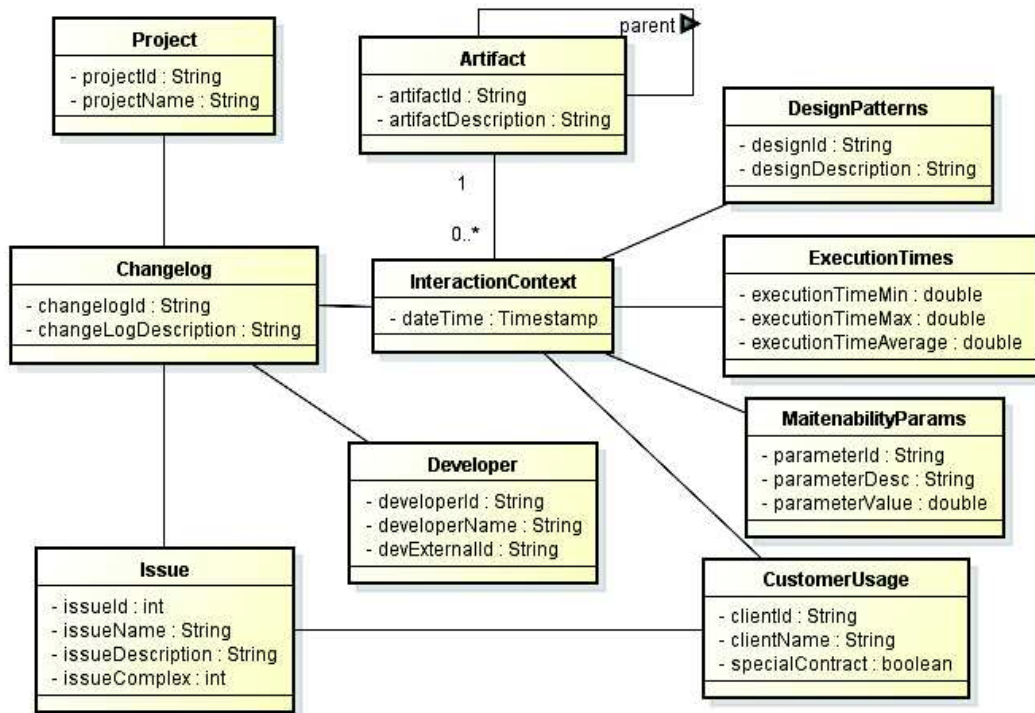


Fonte: Elaborado pelo autor

As informações contextuais, que representam as cinco dimensões consideradas pelo modelo mostradas na Figura 16, são armazenadas de forma histórica em um banco de dados denominado *ContextData*, que é parte integrante do componente. As entidades que serão armazenadas, nesta base de dados, estão representadas no diagrama de domínio da Figura 18. A entidade *InteractionContext* representa as informações de contexto de um artefato (*Artifact*) ao longo do tempo, cada registro desta entidade representa uma fotografia do artefato em relação às demais informações compondo o contexto completo de um artefato e suas dimensões. O conjunto de entradas da entidade *InteractionContext* representa o histórico de contexto de um determinado artefato.

Normalmente, as entradas de contexto (*InteractionContext*) são motivadas por modificações no código-fonte (entidade *ChangeLog*), porém uma entrada em específico pode não estar necessariamente relacionada à todas as demais entidades (dimensões) em um mesmo instante. Pode-se ter o cenário onde há um novo cliente que começou a utilizar a aplicação que contém o artefato. Este cenário adiciona informações ao contexto sem que o artefato em questão possua modificações.

Figura 18: Diagrama de domínio de contexto para *software*



Fonte: Elaborado pelo autor

Cada uma das entidades, que representam as informações de contexto, será detalhada conforme segue:

1. **Artifact**: identifica a principal entidade do modelo, onde todas as demais possuem relação. As classes, métodos, programas, *includes* e demais artefatos de código serão representados e armazenados com um identificador único e uma descrição;
2. **InteractionContext**: representa a informação de contexto do artefato em determinado instante de tempo. O conjunto de registros desta entidade irá compor a trilha de contextos de um determinado artefato. Esta entidade tem a principal função de relacionar o artefato às demais informações;
3. **ChangeLog**: contém a identificação das alterações do artefato (versionamento) e uma descrição. As alterações em código normalmente estão associadas à algum registro técnico que identifica a nova versão, como por exemplo, o identificador (*Id*) de um *commit*

em um repositório de fontes. A composição desta informação depende da tecnologia utilizada pelo desenvolvedor/empresa para administrar o versionamento dos artefatos;

4. **Project**: as alterações em artefatos de código podem ter origens diferentes. Quando a alteração do artefato é oriunda de um projeto ou requerimento, essa informação será armazenada nesta entidade;
5. **Issue**: incidentes de clientes descrevendo comportamentos inesperados são fontes de constantes modificações de código. Quando a alteração do artefato se dá em virtude de algum erro reportado por clientes ou outras equipes, é necessária a identificação do problema a fim de compor o contexto do artefato;
6. **Developer**: identifica o desenvolvedor que alterou o código;
7. **DesignPatterns**: contém as informações relacionadas aos padrões de projeto implementados no artefato;
8. **ExecutionTimes**: relaciona o tempo de execução de um artefato em ambientes de teste e produção;
9. **MaintainabilityParams**: é composto pelas métricas de manutenibilidade do artefato, como por exemplo, número de linhas de código, índice de cobertura de testes unitários ou complexidade ciclomática;
10. **CustomerUsage**: possui a identificação dos clientes que utilizam as funcionalidades implementadas pelo artefato por meio de alguma aplicação de *software*.

4.3.3 Componentes *ConfigurationManager* e *ConfigurationUI*

O armazenamento das configurações do modelo e a interface com o usuário para manter essas informações são atribuições dos componentes *ConfigurationManager* e *ConfigurationUI*. Através destes, os times de desenvolvimento alimentam o modelo com os valores a serem considerados nos serviços de alerta. Será possível configurar o valor limite para as métricas primárias do artefato (complexidade ciclomática, linhas de código, número de dependências e etc) de uma classe bem como composições. As composições de métricas previstas pelo modelo são as seguintes:

- **Composição simples**: as métricas primárias podem ser agregadas em uma nova composição onde os valores dessas métricas são comparados com seus limites. Se todas as métricas estiverem de acordo com o limite parametrizado, essa composição é considerada dentro do esperado. Pode-se considerar, por exemplo, uma composição onde o percentual de cobertura de testes unitários automatizados de uma classe deve ser maior do que 80%, caso o número de objetos que utilizam a mesma, seja maior do que 10;

- **Composição proporcional:** esta composição permite a agregação de duas métricas primárias em uma operação de divisão permitindo uma visão mais detalhada sobre medidas globais. Um exemplo de utilização desta composição é obter um valor resultante da divisão do número de linhas de código de uma classe pela quantidade de métodos;
- **Composição ponderada:** o modelo permite a multiplicação das métricas primárias por um fator de cálculo e posterior multiplicação de todas métricas recalculadas. Pode-se utilizar essa funcionalidade para, por exemplo, elaborar um fator de complexidade das classes, onde a complexidade ciclomática teria um peso bem maior do que o percentual de cobertura de testes unitários automatizados.

Tanto as métricas primárias, como as composições podem ser configuradas em relação a um índice de variação e periodicidade. Esta característica permite o monitoramento de uma possível degradação arquitetural de uma aplicação possibilitando a atuação proativa dos desenvolvedores em relação a este aspecto. A Tabela 4 mostra um exemplo de utilização desta configuração, na qual três métricas hipotéticas estão configuradas. A métrica *Cobertura de Testes Unitários* possui valores limites configurados de acordo com a classificação do artefato, permitindo assim, o controle de um mesmo tipo de métrica adequado à responsabilidade atribuída ao artefato. As colunas *Índice de Variação* e *Periodicidade* possibilitam a avaliação do histórico de contexto em relação à degradação da arquitetura conforme descrito anteriormente. Por fim, a coluna *Operador* indica como deve ser realizada a comparação dos valores em relação aos limites configurados (coluna *Valor Limite*).

Tabela 4: Exemplo de configuração

Id	Descrição	Valor Limite	Índice de Variação	Periodicidade (dias)	Classificação do Artefato	Operador
1	Cobertura de Testes Unitários	75	10	7	Implementação	<
2	Cobertura de Testes Unitários	85	10	7	Framework	<
3	Número de Linhas de Código	900	90	5	Implementação	>

Fonte: Elaborado pelo autor

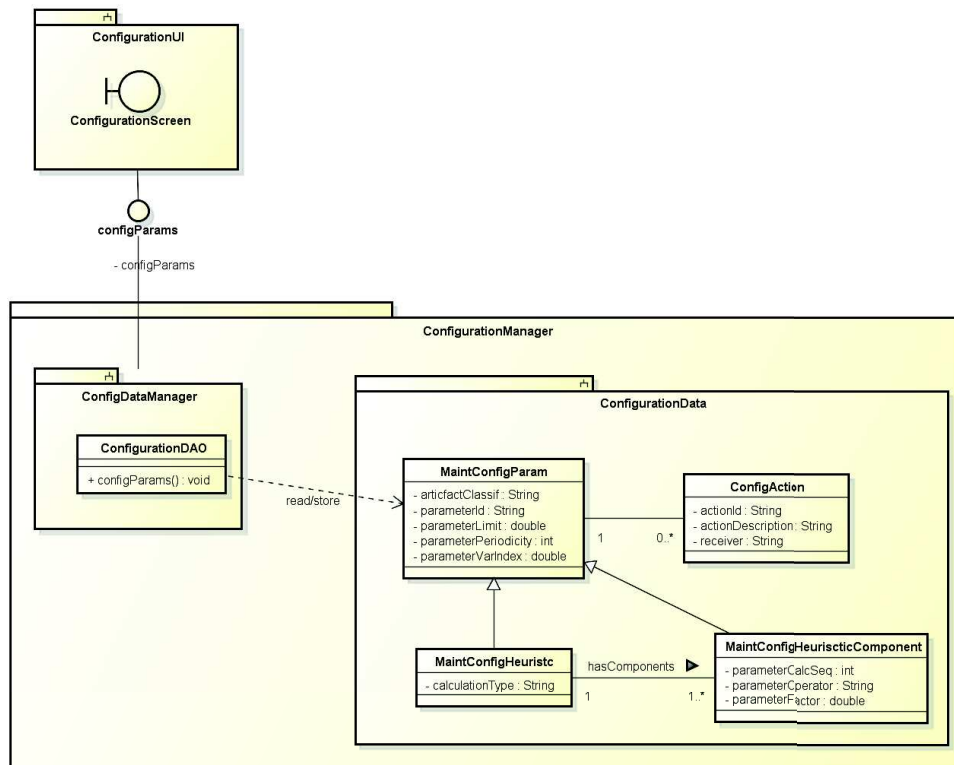
Pode-se, ainda, configurar alertas de e-mail a serem enviados caso algum artefato atinja o valor limite configurado para as métricas primárias e composições em algum momento. As informações de configuração do modelo são armazenadas em um banco de dados (*ConfigurationData*) que é parte integrante do componente *ConfigurationManager*. As entidades que armazenam essas informações estão representadas no módulo *ConfigurationManager*. Este módulo é responsável por todas as operações de banco de dados em relação a essas entidades.

A seguir, as classes relacionadas aos componentes de configuração do modelo (Figura 19) serão descritas:

- **ConfigurationDAO:** classe responsável pelas operações de banco de dados como leitura e atualização das entidades de configuração do modelo;
- **MaintConfigParam:** representa a informação de valor limite para os parâmetros de manutenabilidade de um determinado conjunto de artefatos;

- **MaintConfigHeuristic**: classe que representa um composição de métricas, herdando as características da classe *MaintConfigParam*, com a adição do tipo de cálculo a ser realizados nos seus componentes;
- **MaintConfigHeuristicComponent**: as métricas que compõem uma heurística são representadas por esta classe. Além das características herdadas da classe *MaintConfigParam*, a sequencia de cálculo, a operação e o fator de cálculo são controlados por essa entidade;
- **ConfigAction**: identifica as ações que devem ser consideradas pelo modelo, para quando um determinado valor de parâmetro atinge o limite configurado ou varia de acordo com as configurações.

Figura 19: Diagrama de classes dos componentes de configuração



Fonte: Elaborado pelo autor

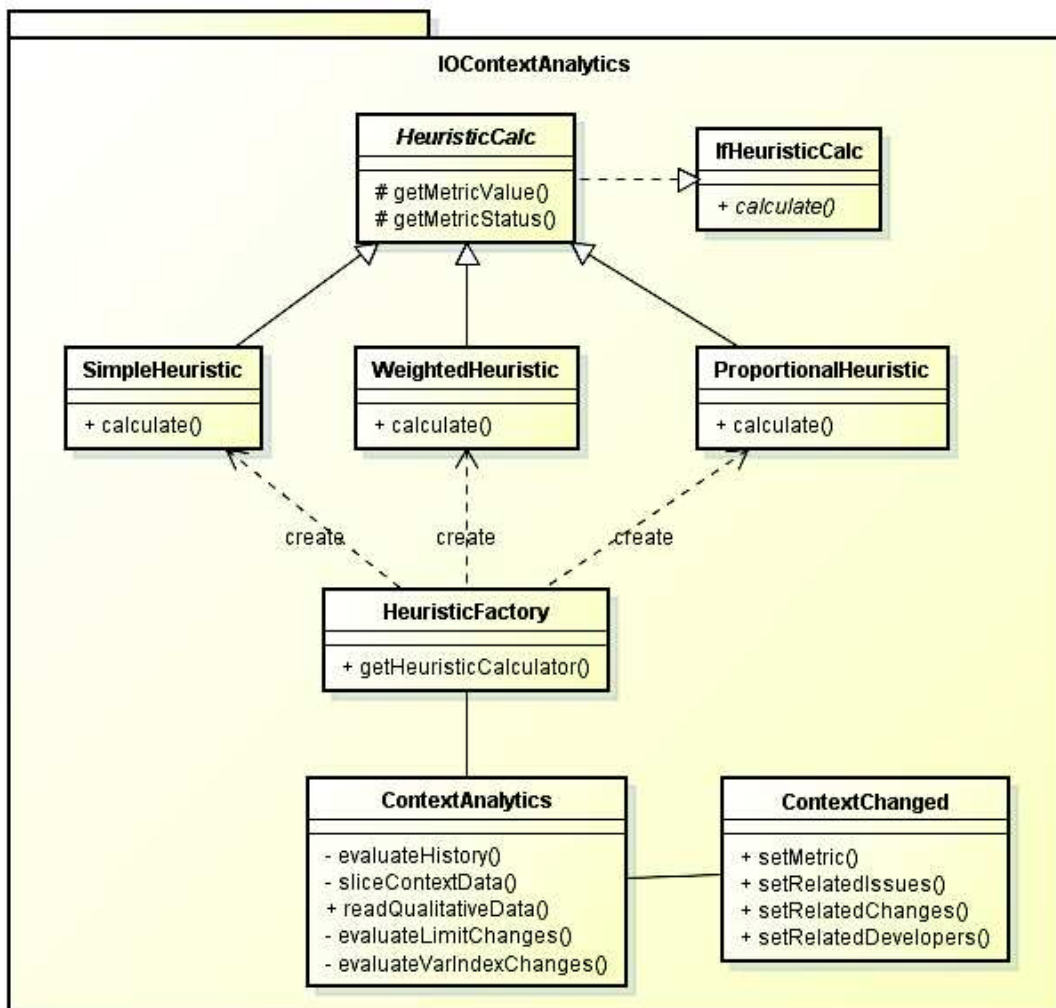
4.3.4 Componente *IOContextAnalytics*

O componente *IOContextAnalytics* é responsável pela avaliação do histórico de contextos de *software* a fim de fornecer dados quantitativos e qualitativos que representem percepções através da correlação das informações armazenadas. Essas informações poderão fornecer relações de causa e efeito, como por exemplo, quando ocorrer alguma alteração contextual significativa. Supondo que as alterações contextuais indiquem a degradação da arquitetura de uma

aplicação, através das informações qualitativas, os times de desenvolvimento poderão averiguar que aspectos adicionais podem ter causado tal efeito como, por exemplo: que desenvolvedores estavam envolvidos na alteração e qual sua experiência; que projeto ou ajuste estava sendo implementado naquele momento; que clientes utilizavam a funcionalidade. Ou seja, o módulo *IOContextAnalytics* aplicará a abordagem *Software Analytics* nos dados contextuais armazenados historicamente.

A Figura 20 mostra a classe *ContextAnalytics*, que, através de seus métodos, interage com o componente *ContextManager* para obtenção dos dados de contexto armazenados historicamente. Após realizar as análises, correlações e cálculos, a classe disponibiliza as informações quantitativas e qualitativas para consumo do componente *ODataService*.

Figura 20: Diagrama de classes do *IOContextAnalytics*



Fonte: Elaborado pelo autor

Outra função deste componente é o cálculo das indicadores compostos configurados no módulo *ConfigurationManager*. O modelo permite agregar medidas de métricas extraídas dos artefatos para gerar uma nova visão em relação à manutenibilidade do artefato.

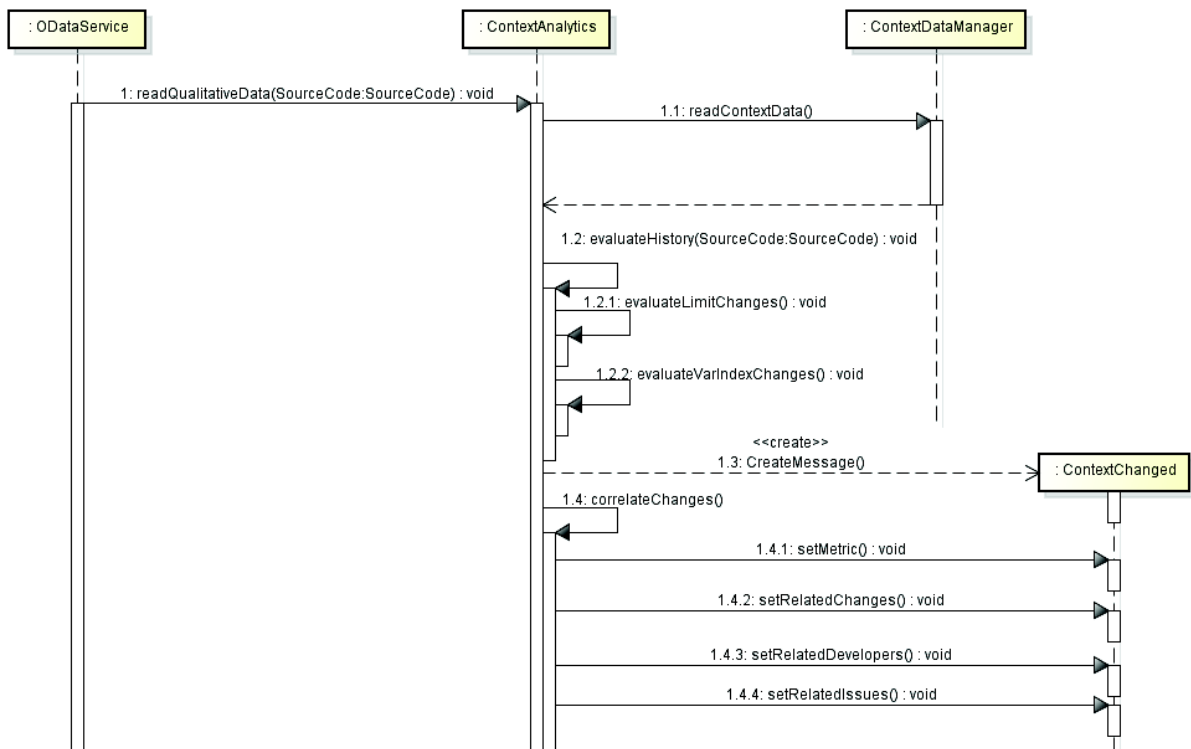
As seguintes classes (Figura 20) são responsáveis pelo cálculo das composições de métricas

configuradas no modelo:

- **SimpleHeuristic**: essa classe efetua o cálculo da composição simples, onde os valores das métricas primárias são comparados com os limites configurados;
- **ProportionalHeuristic**: classe que calcula as composições proporcionais, onde o valor de uma métrica pode ser dividida por outra;
- **WeightedHeuristic**: classe que calcula a composição ponderada, onde o valor de cada métrica primária é multiplicada por um fator de cálculo e depois recalculadas para obter um valor final.

A interação entre as classes do componente *IOContextAnalytics* e os componentes *Context-Manager* e *ODataService* pode ser verificada através do diagrama de sequência representado pela Figura 21.

Figura 21: Diagrama de sequência do *IOContextAnalytics*

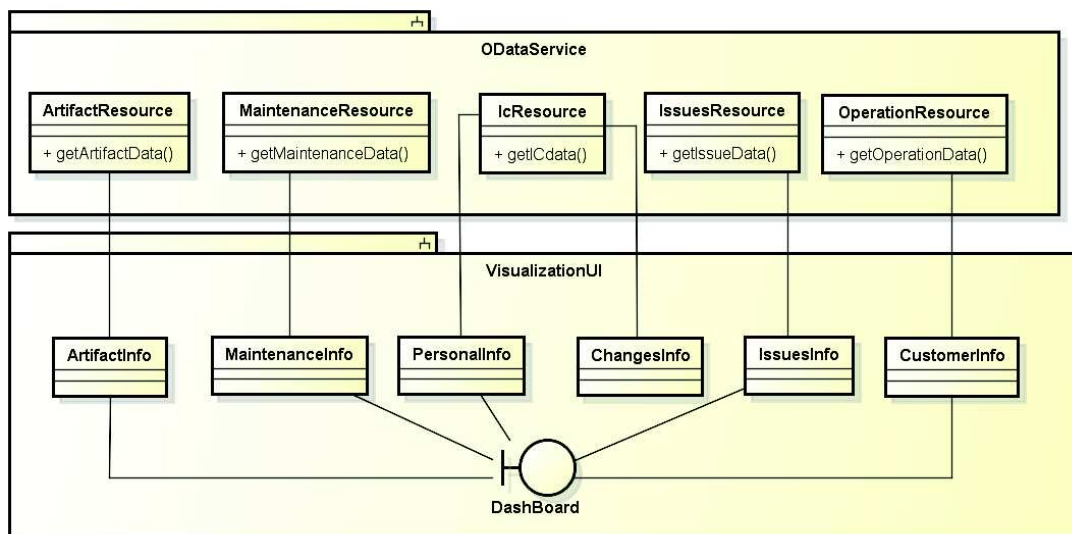


Fonte: Elaborado pelo autor

4.3.5 Componentes *ODataService* e *VisualizationUI*

A Figura 22 mostra a relação entre os componentes *ODataService* e *VisualizationUI*. O componente *ODataService* é responsável por expor seis interfaces de dados de contexto (informações qualitativas geradas pelo componente *IOContextAnalytics*, através das análises e correlações do histórico contextual) que podem ser acessadas por diferentes camadas de visualização de dados.

Figura 22: Diagrama de classes dos componentes de visualização



Fonte: Elaborado pelo autor

Todas as informações disponibilizadas pelas interfaces estarão relacionadas a um determinado artefato de *software*, usualmente, código-fonte. Os dados são disponibilizados, via *web-service* RESTful, no formato JSON através das seguintes classes:

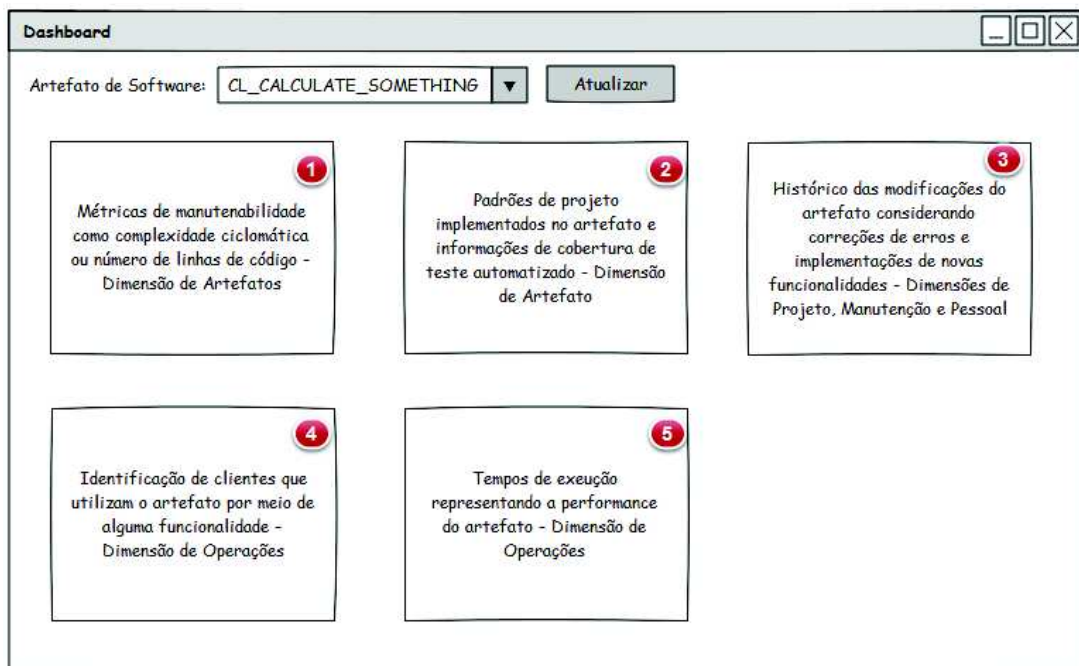
- ***ArtifactResource***: proverá as informações relativas aos artefatos de código, sua identificação, métricas relevantes entre outros;
- ***MaintenanceResource***: os valores atuais das métricas primárias e indicadores compostos, bem como o histórico dos valores e suas condições em relação aos limites configurados serão disponibilizados por esta classe.
- ***IcResource***: as informações relacionadas às alterações de código relacionadas à correções de erros serão disponibilizadas por esta classe. A identificação do erro ou projeto, descrição e complexidade serão as informações disponibilizadas. Além disso, os dados dos desenvolvedores que realizaram alterações no artefato em questão serão disponibilizados;
- ***IssuesResource***: esta classe disponibilizará as informações dos incidentes e chamados

relacionados ao artefato em questão, a data e hora de criação dos incidentes e os clientes que originaram os mesmos;

- **OperationResource**: disponibilizará as informações da execução do artefato de código em ambiente produtivos. Os dados disponibilizados serão tempos de execução e número de clientes que utilizam o artefato através de alguma funcionalidade do sistema.

Através das classes descritas anteriormente, o componente *VisualizationUI* consome os dados e disponibiliza os mesmos em um formato de *dashboard* provendo informações úteis que permitam a tomada de decisão do desenvolvedor em tempo real. A Figura 23 mostra como as informações serão organizadas e disponibilizadas com o objetivo de capacitar desenvolvedores e equipes a obter e compartilhar uma visão de seus dados para tomar melhores decisões. Como pode ser verificado, as informações estão organizadas em cinco áreas relacionadas diretamente a cada uma das dimensões descritas na Seção 4.3.2.

Figura 23: Organização das informações no *dashboard* analítico



Fonte: Elaborado pelo autor

4.4 Considerações sobre o capítulo

Neste capítulo foram apresentadas visão geral e arquitetura do modelo *SW-Context*, bem como a organização dos seus componentes de *software*. Com o avanço da pesquisa, foram acrescentados os detalhes de implementação de cada componente da arquitetura. Após a conclusão da especificação, foi desenvolvido um protótipo para validação do modelo, conforme será descrito a seguir.

5 ASPECTOS DE IMPLEMENTAÇÃO

Neste capítulo são descritos os aspectos do desenvolvimento do protótipo que foi construído para avaliar o modelo *SW-Context*, bem como as tecnologias empregadas no desenvolvimento e os detalhes técnicos. Este capítulo está dividido em quatro seções. Na Seção 5.1 são descritas as tecnologias e metodologias que foram utilizadas no desenvolvimento bem como algumas ferramentas auxiliares. A Seção 5.2 descreve os serviços de disponibilização de dados qualitativos desenvolvidos. A Seção 5.3 apresenta as telas de interface desenvolvidas no protótipo que, consomem os dados disponibilizados pelos serviços e transformam estes dados em informações amigáveis ao usuário final. Por fim, a Seção 5.4 apresenta as considerações finais deste capítulo.

5.1 Tecnologias utilizadas

Um protótipo do modelo *SW-Context* foi desenvolvido utilizando a linguagem JAVA. Os serviços de disponibilização dos dados de contexto foram implementados com as bibliotecas JAVA que suportam a implementação do protocolo RESTful, mais precisamente JAX-RS¹.

Como sistema gerenciador de banco de dados (SGBD) foi utilizado o MySQL, um banco de dados *Open Source* utilizado largamente por desenvolvedores e empresas de tecnologia. Para realizar as operações de leitura e gravação no banco de dados, utilizou-se o *framework* de persistência *Hibernate*². Esse *framework* abstrai o acesso ao banco de dados e gera consultas nas tabelas conforme o serviço requisitado pelo usuário ou por uma requisição interna do *software*. Além disso, o *framework* é o responsável por estabelecer a conexão com o banco de dados e controlar a quantidade de conexões permitidas em determinado momento.

As entidades do banco de dados, que representam as informações contextuais, foram implementadas considerando a definição de classes (Figura 17) descrita no pelo modelo proposto. Já as entidades do banco de dados que permitem a configuração dos parâmetros para avaliação do histórico de contextos foram implementadas considerando as definições previstas no modelo para o componente *ConfigurationManager* (Figura 19).

Por fim, os componentes de visualização qualitativa de informações foram implementados com a utilização de tecnologias como *Hypertext Markup Language* (HTML), HTML5, *Javascript* e *Cascading Style Sheets* (CSS) através do *framework AngularJS*³.

A aplicação foi desenvolvida apenas em formato *web*, e ficou hospedada localmente no computador do autor durante o período de avaliação. Como os dados do protótipo consideraram o ambiente real de uma empresa de desenvolvimento de *software*, não foi autorizada a hospedagem da aplicação em algum provedor público devido à possível publicação de dados sensíveis (exemplo: identificação de projetos ou desenvolvedores). Para o gerenciamento das

¹<http://docs.oracle.com/javase/6/tutorial/doc/gijqy.html>

²<http://hibernate.org/>

³<https://angularjs.org/>

versões e alterações do protótipo, foi utilizado o versionador BitBucket⁴.

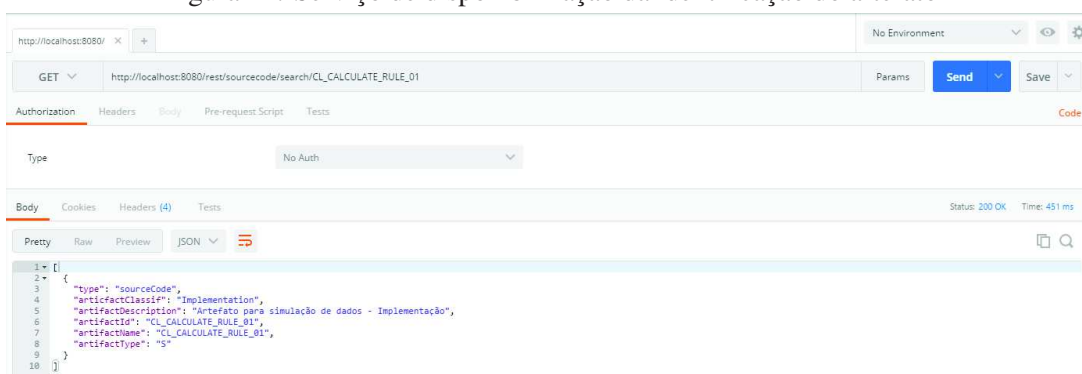
A seguir serão descritos os detalhes de implementação do protótipo em relação aos serviços construídos e as interfaces de usuário que se integram a estes serviços, a fim de prover as informações contextuais qualitativas ao desenvolvedor de *software*.

5.2 Serviços implementados

Os serviços de disponibilização de dados do protótipo foram implementados, conforme descrito no modelo proposto, utilizando o protocolo RESTful. Estes serviços disponibilizam as informações contextuais relacionadas à identificação e caracterização dos artefatos (*ArtifactResource*), seus dados e métricas relacionadas a manutenibilidade (*MaintenanceResource*), controle de alterações do código (*ICResource*), clientes que utilizam o artefato através de um produto de *software* (*CustomerResource*), bem como os incidentes e projetos que deram origem as alterações (*IssuesResource*). A seguir serão detalhados cada um destes serviços:

- **ArtifactResource:** o primeiro serviço (*ArtifactResource*) implementado no protótipo é o responsável pela busca e disponibilização dos dados de identificação do artefato. A Figura 24 mostra o resultado de uma requisição de busca de uma classe através da ferramenta *Postman - REST Client*⁵, onde são disponibilizadas informações que classificam e identificam um artefato de *software* como um código-fonte/classe de implementação final;

Figura 24: Serviço de disponibilização da identificação do artefato



Fonte: Elaborado pelo autor

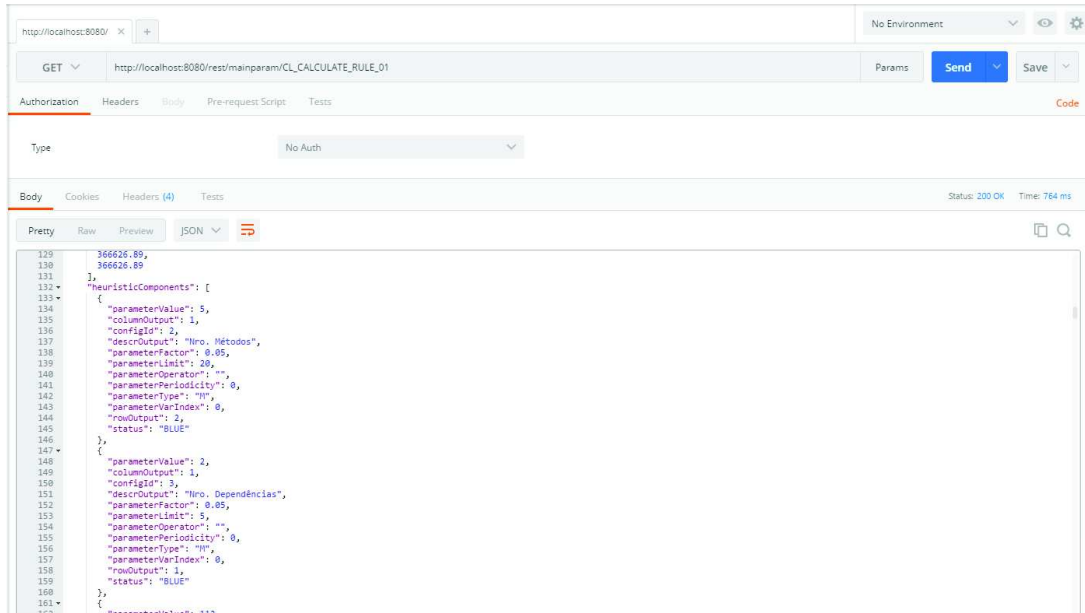
- **MaintenanceResource:** o segundo serviço construído no protótipo refere-se a disponibilização das informações relacionadas aos parâmetros de manutenção (métricas e composições de *software*), seus valores atuais e situação (*status*) em relação aos limites parametrizados no módulo de configuração do modelo. A Figura 25 mostra uma parte do resultado de uma requisição deste serviço onde duas métricas são exibidas, seus valores

⁴BitBucket: controlador de versões de código. (<https://bitbucket.org/>).

⁵<https://www.getpostman.com/>

atuais e outros dados de controle necessários para correta exibição dos dados pela camada de visualização de dados do modelo;

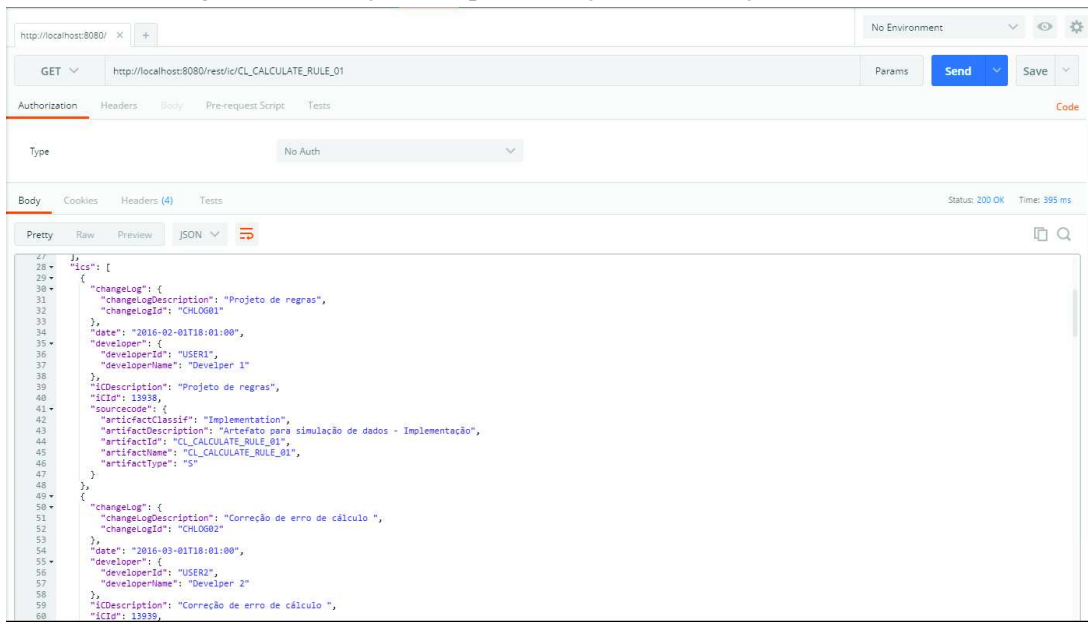
Figura 25: Serviço de disponibilização das métricas do artefato



Fonte: Elaborado pelo autor

- IcResource***: todas as alterações realizadas no artefato de código são disponibilizadas pelo terceiro serviço desenvolvido no protótipo. Além da data de alteração, este serviço expõe qual a origem da alteração (incidente ou projeto), bem como o desenvolvedor que realizou a alteração. A Figura 26 mostra parte do resultado de uma requisição deste serviço;
- IssuesResource***: os incidentes relacionados ao artefato de código são disponibilizados pelo quarto serviço implementado no protótipo. A Figura 27 mostra parte do resultado do serviço, onde dois incidentes relacionados ao código-fonte são disponibilizados. Pode ser verificado também a informação do cliente que originou o incidente, bem como a data de criação do incidente, número de identificação e descrição;
- CustomerResource***: o quinto serviço implementado no protótipo é o último tem termos de disponibilização de dados contextuais. Este serviço expõe as informações dos clientes que utilizam o artefato por meio de um produto de *software*. A Figura 28 mostra uma parte do resultado deste serviço, onde é possível verificar a identificação do cliente, o nome e a data em que a informação foi apurada. Além disso o serviço disponibiliza a relação Cliente X Artefato.

Figura 26: Serviço de disponibilização das alterações do artefato



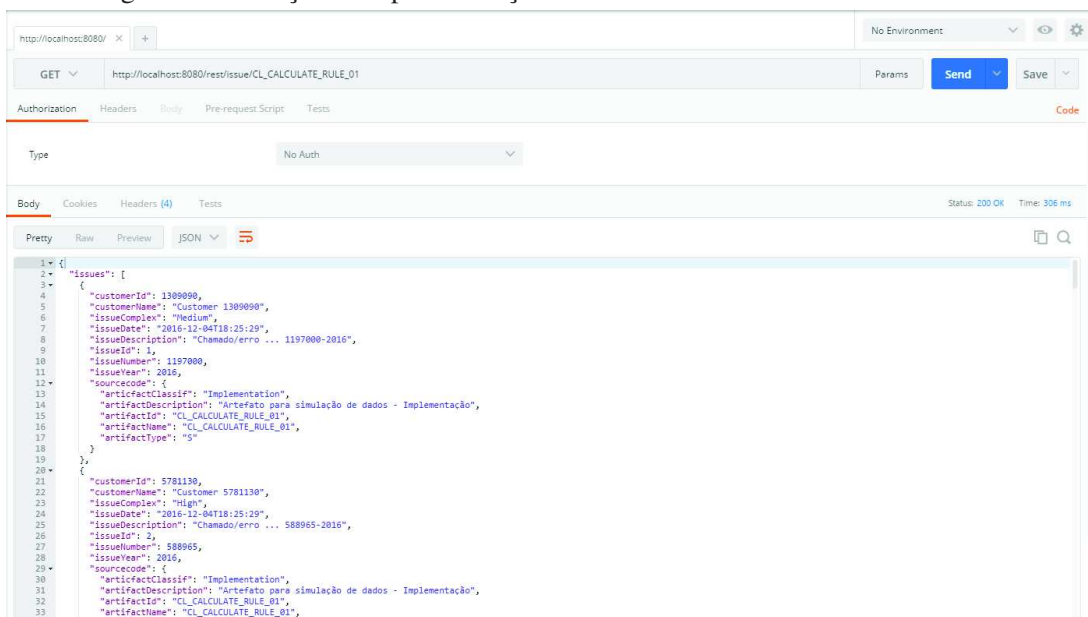
```

27  },
28  "ics": [
29    {
30      "changeLog": {
31        "changeLogDescription": "Projeto de regras",
32        "changeLogId": "CHL0001"
33      },
34      "date": "2016-02-01T18:01:00",
35      "developer": {
36        "developerId": "USER1",
37        "developerName": "Developer 1"
38      },
39      "icDescription": "Projeto de regras",
40      "icId": "13938",
41      "sourcecode": {
42        "artifactClassif": "Implementation",
43        "artifactDescription": "Artefato para simulação de dados - Implementação",
44        "artifactId": "CL_CALCULATE_RULE_01",
45        "artifactName": "CL_CALCULATE_RULE_01",
46        "artifactType": "S"
47      }
48    },
49    {
50      "changeLog": {
51        "changeLogDescription": "Correção de erro de cálculo",
52        "changeLogId": "CHL0002"
53      },
54      "date": "2016-03-01T18:01:00",
55      "developer": {
56        "developerId": "USER2",
57        "developerName": "Developer 2"
58      },
59      "icDescription": "Correção de erro de cálculo",
60      "icId": "13939"
61    }
62  ]

```

Fonte: Elaborado pelo autor

Figura 27: Serviço de disponibilização dos incidentes relacionados ao artefato



```

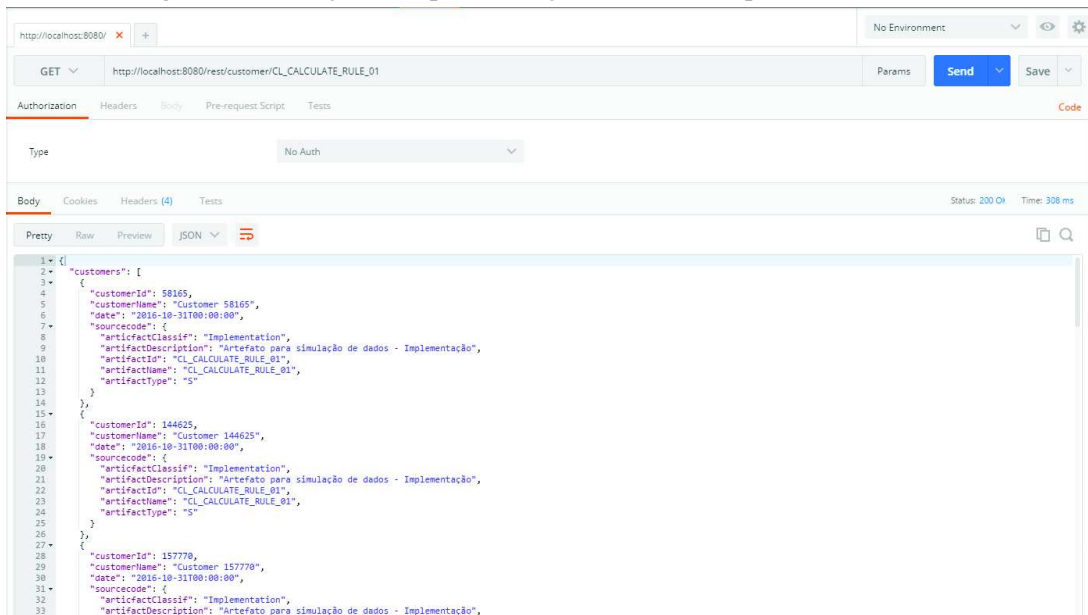
1  {
2    "issues": [
3      {
4        "customer-id": 1389090,
5        "customerName": "Customer 1389090",
6        "issueComplex": "Medium",
7        "issueDate": "2016-12-04T18:25:29",
8        "issueDescription": "Chamado/erro ... 1197000-2016",
9        "issueId": 1,
10       "issueNumber": 1197000,
11       "issueYear": 2016,
12       "sourcecode": {
13         "artifactClassif": "Implementation",
14         "artifactDescription": "Artefato para simulação de dados - Implementação",
15         "artifactId": "CL_CALCULATE_RULE_01",
16         "artifactName": "CL_CALCULATE_RULE_01",
17         "artifactType": "S"
18       }
19     },
20     {
21       "customer-id": 5781130,
22       "customerName": "Customer 5781130",
23       "issueComplex": "High",
24       "issueDate": "2016-12-04T18:25:29",
25       "issueDescription": "Chamado/erro ... 588965-2016",
26       "issueId": 2,
27       "issueNumber": 588965,
28       "issueYear": 2016,
29       "sourcecode": {
30         "artifactClassif": "Implementation",
31         "artifactDescription": "Artefato para simulação de dados - Implementação",
32         "artifactId": "CL_CALCULATE_RULE_01",
33         "artifactName": "CL_CALCULATE_RULE_01",
34         "artifactType": "S"
35       }
36     }
37   ]
38 }

```

Fonte: Elaborado pelo autor

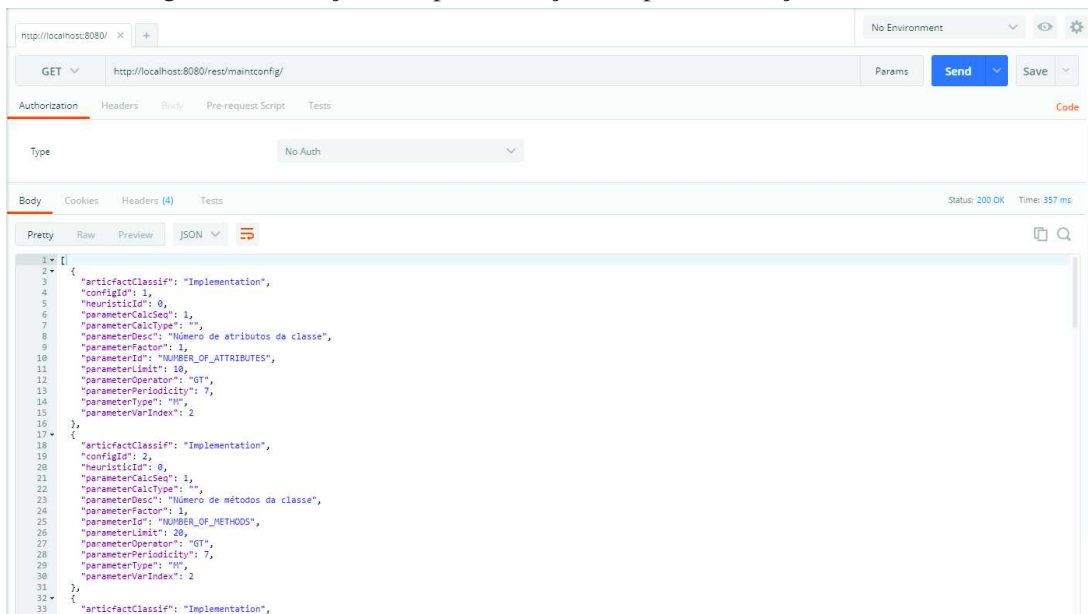
Além dos serviços de disponibilização de dados contextuais, foi implementado um serviço que expõe as informações armazenadas no módulo de configuração previsto pelo modelo. Neste serviço, as configurações de valores limites e demais parametrizações das métricas primárias e compostas são disponibilizadas. Este serviço é utilizado pelo componente de visualização de informações, bem como pelos processos que efetuam a análise histórica das alterações do artefato e alertas. A Figura 29 mostra parte do resultado de uma requisição neste serviço.

Figura 28: Serviço de disponibilização dos clientes que usam o artefato



Fonte: Elaborado pelo autor

Figura 29: Serviço de disponibilização das parametrizações do modelo



Fonte: Elaborado pelo autor

Através destes serviços, o componente de visualização de dados do protótipo realiza a obtenção dos dados, interpreta as respostas JSON transformando essas informações em telas amigáveis para o usuário final. A seguir serão descritas as telas implementadas no protótipo, que consomem os dados dos serviços descritos.

5.3 Telas implementadas

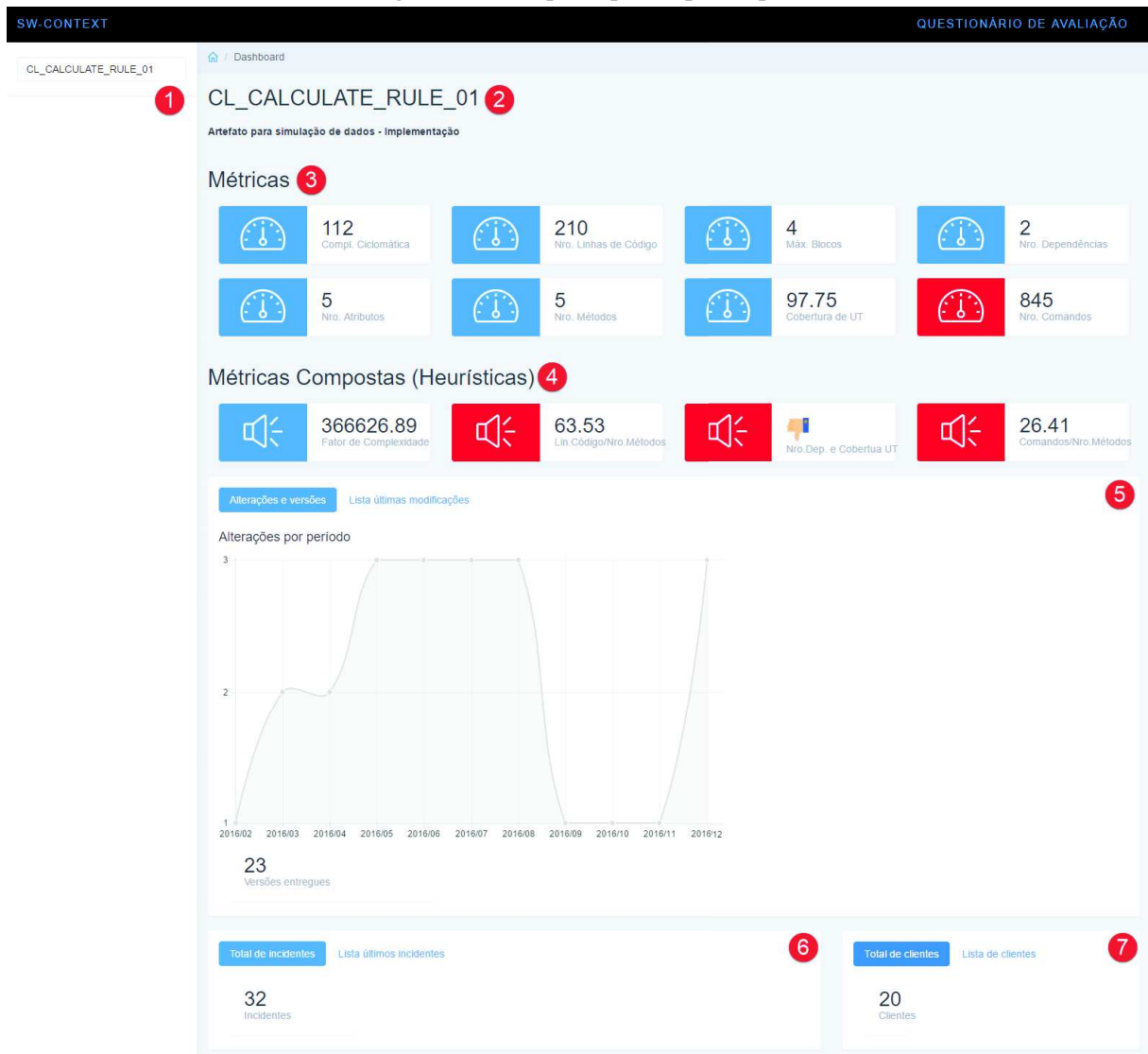
Conforme descrito no capítulo 4.3.5, as telas do componente *VisualizationUI* foram desenvolvidas de forma a consumir os dados contextuais disponibilizados pelos serviços e transformá-los em um formato gráfico e qualitativo a fim de fornecer informações úteis aos desenvolvedores. Todas as telas detalhadas neste capítulo fazem parte de um conjunto total de 13 funcionalidades de interface com o usuário, que foram desenvolvidas para permitir a utilização e avaliação do protótipo pelos desenvolvedores de uma empresa de desenvolvimento de *software*.

A Figura 30 mostra a tela principal do protótipo dividida em 7 áreas para melhor compreensão. Essa tela foi construída no formato de *dashboard* e pretende dispor todas informações contextuais de um determinado artefato. A área 1 da figura identifica o campo disponibilizado para o desenvolvedor pesquisar a classe que deseja. Ao localizar e selecionar a classe, o seu nome e descrição são apresentados na área identificada pelo número 2.

Os parâmetros de manutenção representados pelas métricas ficam disponíveis na área 3, já as composições de métricas são apresentadas logo a seguir, na área 4. As cores diferentes nas áreas 3 e 4 indicam a situação da métrica/composição em relação aos valores limites configurados, na qual a cor azul representa que o valor está dentro dos limites. Já a cor vermelha mostra que a métrica/composição ultrapassou os valores limites, alertando desvios em relação às boas práticas configuradas e uma possível degradação arquitetural da aplicação.

A área 5 da figura identifica o gráfico com a quantidade de alterações realizadas no artefato por mês e o número total de versões entregues para ambientes produtivos. O usuário pode obter o detalhamento dessas alterações através do atalho *Lista últimas modificações*, em que são apresentadas a identificação do usuário que realizou a alteração, data da alteração, descrição e outras informações de controle. As áreas 6 e 7 da figura identificam os totais de incidentes relacionados ao artefato e clientes que criaram os incidentes. Essas informações também podem ser detalhadas pelos atalhos disponibilizados em cada uma das áreas.

Figura 30: Tela principal do protótipo



Fonte: Elaborado pelo autor

A situação das métricas/composições é determinada pela comparação entre histórico de contexto do artefato e os valores limites configurados pelas telas do componente *ConfigurationUI*. A Figura 31 mostra a tela do protótipo que lista os valores configurados conforme a classificação do artefato (*Implementation/Framework*). Através do botão vermelho, o usuário seleciona uma configuração para alterar os valores limites, índice de variação e periodicidade da variação.

Figura 31: Limites das métricas/composições

Id	Descrição	Valor Limite	Variação	Periodicidade da variação(dias)	Classificação do artefato
NUMBER_OF_ATTRIBUTES	Número de atributos da classe	10	2	7	Implementation
NUMBER_OF_METHODS	Número de métodos da classe	20	2	7	Implementation
NUMBER_OF_DEPENDENCIES	Número de objetos que utilizam a classe	5	2	7	Implementation
CICLOMATIC_COMPLEXITY	Complexidade ciclomática do objeto	400	40	7	Implementation
NUMBER_OF_STATEMENTS	Número de comandos executáveis	300	30	7	Implementation
NESTED_BLOCK_DEPTH	Profundidade de blocos aninhados	4	1	7	Implementation
LOC	Total de linhas de código	900	90	7	Implementation
UT_COVERAGE	Percentual de cobertura de testes	75	10	7	Implementation

Fonte: Elaborado pelo autor

A Figura 32 mostra a tela que permite a configuração de uma composição proporcional, onde podem ser seleccionadas duas métricas primárias que serão proporcionalizadas. Adicionalmente, podem ser configurados valor limite, índice de variação e periodicidade da variação.

Figura 32: Composição proporcional

LOC/NUMBER_OF_METHODS

Descrição: Linhas de código por método

Classificação do Artefato: Implementation

Tipo de Cálculo: Proporcional

Cálculo da Heurística: LOC / NUMBER_OF_METHODS

Valor Limite: 45

Índice de Variação: 10

Periodicidade da Variação: 7 dias

Gravar

Fonte: Elaborado pelo autor

A configuração de composições simples é caracterizada pela seleção de múltiplas métricas primárias, indicação de valores limites e critérios de comparação. A Figura 33 mostra a tela do protótipo que permite a parametrização descrita. Os campos do tipo *combobox*, localizados mais a esquerda da tela, listam todas as métricas primárias existentes no banco de dados, já os campos centralizados disponibilizam os critérios de comparação maior (>) e menor (<). Por fim, o campo posicionado mais a direita da tela permite a digitação do valor limite a ser comparado.

Figura 33: Composição simples

The screenshot shows a configuration window titled "NUMBER_OF_DEPENDENCIES AND UT_COVERAGE". It contains the following information:

- Descrição:** Número de objetos que utilizam a classe e Percentual de cobertura de testes
- Classificação do Artefato:** Implementation
- Tipo de Cálculo:** Simple
- Cálculo da Heurística:**
 - NUMBER_OF_DEPENDENCIES > 10
 - UT_COVERAGE < 80

At the bottom, there is a green "+ Adicionar" button and a grey "Gravar" button.

Fonte: Elaborado pelo autor

A tela do protótipo que permite a configuração das composições ponderadas pode ser verificada na Figura 34. O usuário pode selecionar múltiplas métricas através dos campos *combobox* posicionados mais a esquerda da tela. Para cada métrica selecionada, a tela permite a indicação de uma operação matemática básica (soma, subtração, divisão ou multiplicação), um fator de cálculo e um valor máximo para ser utilizado quando a métrica é marcada como inversamente proporcional. Recomenda-se a utilização de métricas inversamente proporcionais, por exemplo, na configuração de uma composição que representaria o fator de complexidade do artefato. Para este cálculo hipotético, quanto maior o índice de cobertura de testes unitários automatizados menor seria a complexidade da classe em questão.

Figura 34: Composição ponderada

The screenshot shows a configuration window titled "COMPLEXITY_FACTOR". It contains the following information:

- Descrição:** Complexidade geral do objeto
- Classificação do Artefato:** Implementation
- Tipo de Cálculo:** Ponderada - Multiplicação de todos os elementos considerando o fator.
- Cálculo da Heurística:**

NUMBER_OF_METHODS	*	0.05	Inv. Prop. <input type="checkbox"/>	<input type="button" value="⊖"/>
NUMBER_OF_DEPENDENCIES	*	0.05	Inv. Prop. <input type="checkbox"/>	<input type="button" value="⊖"/>
CICLOMATIC_COMPLEXITY	*	0.05	Inv. Prop. <input type="checkbox"/>	<input type="button" value="⊖"/>
LOC	*	0.02	Inv. Prop. <input type="checkbox"/>	<input type="button" value="⊖"/>
UT_COVERAGE	*	0.01	Inv. Prop. <input checked="" type="checkbox"/>	Valor máximo: <input type="text" value="0"/> <input type="button" value="⊖"/>

At the bottom, there is a green "+ Adicionar" button and a grey "Gravar" button.

Fonte: Elaborado pelo autor

Quando o valor de alguma métrica ou composição atinge os valores configurados como limite ou índice de variação, além das indicações visuais do *dashboard*, um alerta é enviado via *e-mail* para o endereço parametrizado. Nesta mensagem de alerta, é identificado o artefato que obteve o desvio, as alterações realizadas pelos desenvolvedores que causaram o desvio, bem como os valores atuais e configurados das métricas e composições. Na Figura 35 é possível verificar um *e-mail* enviado automaticamente pelo protótipo.


Figura 35: Tela de e-mail de alerta enviado automaticamente

Alerta SW-Context 🔍 🔄 🗑️ ✓ ⋮

swcontext para mim 14:29 ⋮

Artefato: CL_CALCULATE_RULE_01

Tipo de Alteração	Data de Alteração	Métrica	Valor atual da métrica	Valor limite métrica	Índice de variação
Limite	2016-11-22 12:14:58.0	CICLOMATIC_COMPLEXITY	430.0	400.0	40.0
Alterações relacionadas:					
Data	Descrição	Usuário			
2016-10-20 18:01:00.0	Correção de exceção inesperada	USER2 - Developer 2			
2016-11-12 18:01:00.0	Melhoria de performance - Correção de incidente	USER1 - Developer 1			
Tipo de Alteração	Data de Alteração	Métrica	Valor atual da métrica	Valor limite métrica	Índice de variação
Limite	2016-11-22 10:01:23.0	LOC/NUMBER_OF_METHODS	63.53	45.0	10.0
Alterações relacionadas:					
Data	Descrição	Usuário			
2016-10-20 18:01:00.0	Correção de exceção inesperada	USER2 - Developer 2			
2016-11-12 18:01:00.0	Melhoria de performance - Correção de incidente	USER1 - Developer 1			
Tipo de Alteração	Data de Alteração	Métrica	Valor atual da métrica	Valor limite métrica	Índice de variação
Variação	2016-11-22 12:14:58.0	CICLOMATIC_COMPLEXITY	430.0	400.0	40.0
Alterações relacionadas:					
Data	Descrição	Usuário			

 Responder ➔

Fonte: Elaborado pelo autor

5.4 Considerações sobre o capítulo

Este capítulo abordou em três seções os aspectos de implementação do protótipo utilizado na avaliação realizada através de um estudo de caso, que será descrito a seguir. As principais tecnologias utilizadas na implementação foram descritas na Seção 5.1 e a utilização destas tecnologias nas funcionalidades de disponibilização e visualização das informações contextuais foi detalhada nas Seções 5.2 e 5.3 .

6 ASPECTOS DE AVALIAÇÃO

Neste capítulo são descritos os aspectos de avaliação do modelo proposto. A Seção 6.1 descreve a metodologia de avaliação do modelo. A Seção 6.2 descreve os detalhes da execução de um experimento controlado para realização da primeira avaliação. A segunda etapa de avaliação compreende um estudo de caso na indústria que será detalhado na Seção 6.3. Por fim, a Seção 6.4 apresenta as considerações finais deste capítulo.

6.1 Metodologia

A avaliação do modelo *SW-Context* foi realizada em duas etapas. A primeira avaliação deu-se através de um experimento controlado. A condução deste experimento ocorreu com a participação de 30 voluntários, divididos em 2 grupos de 15 pessoas, que realizaram atividades de manutenção em uma aplicação. Apenas um dos grupos realizou as atividades utilizando um *dashboard* que continha dados contextuais qualitativos em relação à aplicação. Posteriormente, ambos grupos tiveram seus resultados comparados e testados estatisticamente.

A segunda etapa da avaliação no modelo ocorreu através de um estudo de caso em uma empresa de desenvolvimento de *software*, por meio de uma equipe de desenvolvimento que se voluntariou para tal. Esta equipe, composta por 12 desenvolvedores, utilizou o protótipo do modelo *SW-Context* como ferramenta de apoio pelo período de um mês em suas atividades de manutenção de código (implementação de novas funcionalidades ou correções código). A avaliação do modelo foi organizada de forma que, a cada atividade de desenvolvimento realizada por algum desenvolvedor, o protótipo fosse utilizado como apoio. Ao final da atividade, os desenvolvedores responderam a um questionário elaborado pelo autor que avaliou a utilidade e facilidade de uso da ferramenta.

A Seção 6.2 descreve o experimento controlado conduzido na primeira avaliação do modelo e seus resultados. A Seção 6.3 detalha o estudo de caso na indústria realizado para avaliação final do modelo *SW-Context*. Por fim, a Seção 6.4 descreve as considerações sobre o capítulo.

6.2 Experimento controlado

Um experimento controlado foi conduzido com o objetivo de avaliar a utilização de informações de contexto disponibilizadas em formato de *dashboard* em atividades de manutenção de *software*. Este experimento avaliou os efeitos do uso de informações de contexto em três variáveis: a corretude, esforço e manutenibilidade do código-fonte alterado de uma aplicação adaptada pelo autor para uso específico no experimento. Estes efeitos foram investigados baseados em cenários reais de manutenção de software a partir de requerimentos de mudança, assim gerando conhecimento empírico sobre os benefícios de *dashboards* qualitativos para melhoria da consciência situacional dos desenvolvedores. Com base nisso, o objetivo do experimento foi

definido baseado na *template GQM* (SOLINGEN et al., 2002) conforme segue:

*Analisar o uso de informações qualitativas de contexto
para o propósito de **investigar os seus efeitos**
em relação a **corretude, esforço e manutenibilidade**
da perspectiva de **desenvolvedores**
no contexto de **manutenção de software**.*

A partir do objetivo do experimento definido, três hipóteses foram formuladas a fim de avaliar os efeitos das informações qualitativas disponibilizadas em relação às variáveis descritas anteriormente. A primeira se refere a avaliar a corretude do código alterado usando informações contextuais através de *dashboard*. A segunda hipótese considera o uso do mesmo *dashboard* e seus efeitos no esforço de implementação das solicitações de alteração. A terceira hipótese considera o uso do *dashboard* e seus efeitos sobre a manutenibilidade do código alterado. A formulação dessas hipóteses foi feita com base no pressuposto de que informações contextuais dispostas em um formato qualitativo melhoram a consciência dos desenvolvedores sobre a aplicação que está sendo alterada e o ambiente relacionado. A formulação das três hipóteses será descrita a seguir.

A experiência do autor em desenvolvimento de *software* permite afirmar que, na prática, os desenvolvedores costumam alterar o código-fonte sem qualquer informação qualitativa para apoiar as mudanças a serem feitas, confiando apenas em sua experiência sobre a aplicação. Embora essa estratégia para implementar as solicitações de mudança seja frequentemente usada na prática, conjectura-se que ela não é suficientemente eficaz para suportar corretamente a implementação de solicitações de alterações complexas. Em parte, porque geralmente os desenvolvedores precisam manter requisitos não-funcionais, por exemplo, *performance* e segurança, cujas implementações são amplamente conhecidas por impactar as aplicações de forma transversal. Ou seja, as implementações dos requisitos não-funcionais acabam se espalhando sobre muitos módulos do sistema, dando origem a duplicidade de código e dependências significativas entre módulos. Além disso, é difícil reunir informações qualitativas sobre como esses requisitos são implementados, além do que é encontrado na análise do código-fonte, para realizar adequadamente as solicitações de alteração. Consequentemente, foi criada a hipótese que os desenvolvedores tendem a produzir um número maior de código-fonte corretamente alterado, fazendo uso de informações contextuais qualitativas, em vez de usar apenas indicadores mentais. No entanto, não é óbvio que esta hipótese se mantenha. Talvez, esses indicadores possam ser mais eficazes para suportar os diferentes aspectos das solicitações de alteração, o que pode ajudar os desenvolvedores a implementar as alterações mais adequadamente. Desse modo, a primeira hipótese avalia se o uso de informações contextuais qualitativas produz um número maior de código-fonte corretamente alterado em comparação com o uso de indicadores mentais. Com base nessa afirmação, as hipóteses nula e alternativa são apresentadas da seguinte forma:

- **Hipótese Nula 1, H_{1-0} :** O uso de informações qualitativas de contexto (*Context*) produz um número menor (ou igual) de código-fonte corretamente alterado do que apenas o uso de indicadores mentais (*no-Context*).

$$H_{1-0}: Corr(Code)_{Context} \leq Corr(Code)_{No-Context}$$

- **Hipótese Alternativa, H_{1-1} :** O uso de informações qualitativas de contexto (*Context*) produz um número maior de código-fonte corretamente alterado do que apenas o uso de indicadores mentais (*no-Context*).

$$H_{1-1}: Corr(Code)_{Context} > Corr(Code)_{No-Context}$$

As necessidades de clientes que mudam rapidamente e são imprevisíveis na maioria dos projetos de desenvolvimento de *software*, originam um número elevado de solicitações de mudança. Os desenvolvedores são muitas vezes desafiados a acomodar essas solicitações em códigos-fonte em evolução. Na prática, os desenvolvedores precisam implementar as mudanças necessárias sem entender os efeitos colaterais das mudanças e a própria arquitetura do *software*. Além disso, os desenvolvedores trabalham sob pressão para entregar o código-fonte modificado mais rapidamente. Devido a restrições de tempo e falta de informações qualitativas sobre o código-fonte (por exemplo, padrões de projeto usados e decisões de *design* a serem considerados), os desenvolvedores acabam implementando as solicitações de mudança considerando apenas as informações encontradas no código-fonte. Com isso em mente, conjectura-se que o uso de informações qualitativas de contexto pode reduzir significativamente o esforço de implementação. Ou seja, suspeita-se que o esforço para implementar as requisições de mudança tende a ser menor se os desenvolvedores utilizarem informações qualitativas sobre o código-fonte, além da sua experiência. Sendo assim, a segunda hipótese avalia se o uso de informações qualitativas de contexto reduz significativamente o esforço da implementação de solicitações de mudança, em comparação com o uso de indicadores mentais de desenvolvedores e a análise de código-fonte necessária. Com base nessa afirmação, as hipóteses nula e alternativa são apresentadas da seguinte forma:

- **Hipótese Nula 2, H_{2-0} :** O uso de informações qualitativas de contexto (*Context*) não reduz significativamente o esforço de implementação de requisições de mudanças em comparação ao uso apenas da experiência dos desenvolvedores (*no-Context*).

$$H_{2-0}: Effort(Code)_{Context} > Effort(Code)_{No-Context}$$

- **Hipótese Alternativa 2, H_{2-1} :** O uso de informações qualitativas de contexto (*Context*) reduz significativamente o esforço de implementação de requisições de mudanças em comparação ao uso apenas da experiência dos desenvolvedores (*no-Context*).

$$H_{2-1}: Effort(Code)_{Context} \leq Effort(Code)_{No-Context}$$

No ambiente corporativo, as atividades de manutenção de *software* geralmente são originadas de demandas urgentes, exigências legais, mudanças de ambiente tecnológico e outras.

Os desenvolvedores precisam adaptar o sistema a essas mudanças e tentar manter a arquitetura do *software* de acordo com as melhores práticas e o *design* anteriormente definido. De acordo com a natureza das solicitações, os desenvolvedores precisam ir mudando o código-fonte com o foco em resolver o problema, o mais rapidamente possível, sem considerar adequadamente a arquitetura do *software* e o efeito colateral das mudanças para o clientes, usuários e integrações de módulos/sistemas. Além disso, a pressão do cliente muitas vezes pode reduzir o tempo que os desenvolvedores têm para analisar e implementar as mudanças necessárias do sistema. Para lidar com essa complexa rede de fatores, os desenvolvedores realizam as mudanças com base na sua própria experiência e conhecimento sobre um determinado módulo. As informações sobre a arquitetura adotada, padrões de projeto e dependências de *software* são de complexa obtenção tornando difícil a execução adequada das atividades de manutenção no local correto, significa na classe correta, método, função ou procedimento. Desta forma, conjectura-se que a manutenibilidade de um código-fonte modificado pode ser diferente se houver informações qualitativas de contexto disponíveis sobre este artefato durante o processo de manutenção mesmo. Suspeita-se que o código pode ter uma maior manutenibilidade se a mudança for realizada com a utilização de informações qualitativas de contexto sobre o código-fonte e não apenas considerando a experiência do desenvolvedor. Portanto, a terceira hipótese avalia se o uso de informações qualitativas de contexto melhora a manutenibilidade do código-fonte em comparação com o uso da experiência do desenvolvedor apenas. Com base nessa afirmação, as hipóteses nula e alternativa são apresentadas da seguinte forma:

- **Hipótese Nula 3**, H_{3-0} : O uso de informações qualitativas de contexto (*Context*) não melhora significativamente (nem mesmo compromete) a manutenibilidade de códigos-fonte em comparação ao uso apenas da experiência dos desenvolvedores (*no-Context*).

$$H_{3-0}: \text{Maint}(\text{Code})_{\text{Context}} \leq \text{Maint}(\text{Code})_{\text{No-Context}}$$

- **Hipótese Alternativa 3**, H_{3-1} : O uso de informações qualitativas de contexto (*Context*) melhora significativamente a manutenibilidade de códigos-fonte em comparação ao uso apenas da experiência dos desenvolvedores (*no-Context*).

$$H_{3-1}: \text{Maint}(\text{Code})_{\text{Context}} > \text{Maint}(\text{Code})_{\text{No-Context}}$$

A Tabela 5 mostra cada uma dessas hipóteses formuladas para avaliação dos efeitos das informações qualitativas de contexto em relação à consciência situacional dos desenvolvedores durante as atividades do experimento. Todas essas hipóteses foram avaliadas estatisticamente através de testes específicos que serão detalhados posteriormente.

Tabela 5: Hipóteses formuladas

Hipótese nula	Hipótese alternativa
$H_{1-0}: Corr(Code)_{Context} \leq Corr(Code)_{No-Context}$	$H_{1-1}: Corr(Code)_{Context} > Corr(Code)_{No-Context}$
$H_{2-0}: Effort(Code)_{Context} > Effort(Code)_{No-Context}$	$H_{2-1}: Effort(Code)_{Context} \leq Effort(Code)_{No-Context}$
$H_{3-0}: Maint(Code)_{Context} \leq Maint(Code)_{No-Context}$	$H_{3-1}: Maint(Code)_{Context} > Maint(Code)_{No-Context}$

Corr: Corretude

Effort: Esforço

Maint: Manutenibilidade

Fonte: Elaborado pelo autor

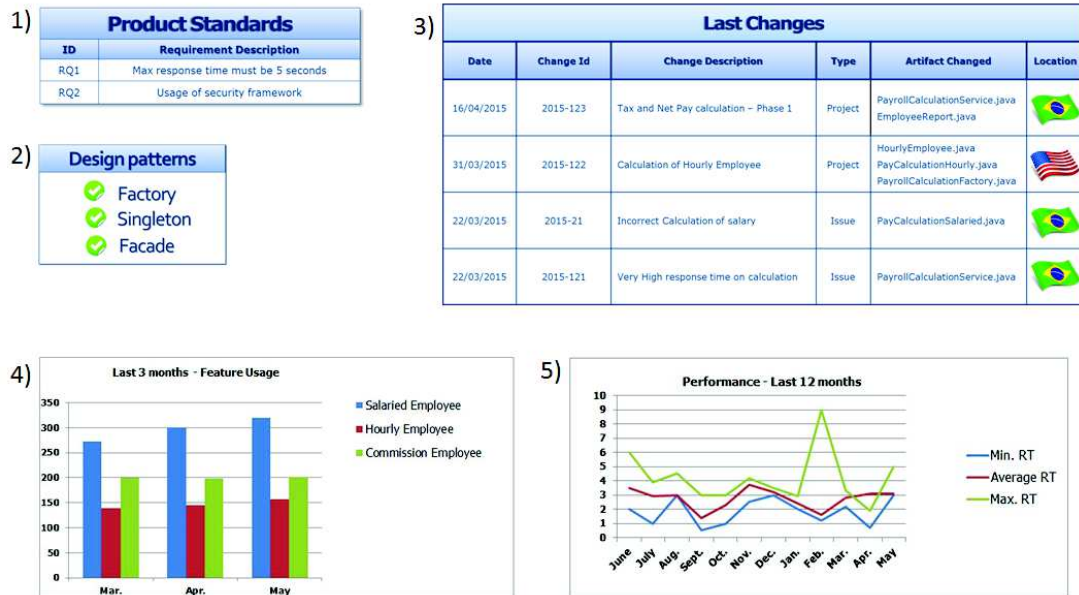
O *dashboard* (Figura 36) utilizado no experimento foi construído manualmente, a partir da pesquisa de alguns estudos sobre contexto em *software* descritos na Seção 3 e conhecimento empírico do autor. As informações disponibilizadas neste *dashboard* se referem à aplicação protótipo, e foram caracterizadas da seguinte forma:

- **Requerimentos não-funcionais** implementados na aplicação protótipo – seção 1 da Figura 36;
- **Padrões de projeto** utilizados na implementação da aplicação protótipo – seção 2 da Figura 36;
- Um histórico das **últimas alterações** supostamente realizadas na aplicação ao longo do tempo – seção 3 da Figura 36;
- Um gráfico representando a **utilização das funcionalidades** por clientes durante um período de 3 meses – seção 4 da Figura 36;
- Um gráfico representando a **performance de execução** da aplicação durante 12 meses – seção 5 da Figura 36.

Para a execução do experimento, foram convidados 30 participantes com experiência profissional em programação e orientação a objetos. Os participantes foram divididos em 2 grupos de 15 no qual um dos grupos (Grupo 1) realizou as atividades de manutenção utilizando o *dashboard* representado na Figura 36 e o outro grupo (Grupo 2) realizou as mesmas atividades sem a utilização das informações qualitativas.

Todos os 30 participantes receberam um treinamento básico sobre a aplicação utilizada no protótipo, onde o autor apresentou a arquitetura da aplicação e suas funcionalidades. Dessa forma, todos obtiveram o mesmo grau de experiência e conhecimento em relação a aplicação que foi alterada por eles. Posteriormente, os participantes foram incentivados a alterar a aplicação do experimento incluindo duas novas funcionalidades. A primeira funcionalidade foi calcular o imposto e valor líquido com base no cálculo de salários já existente. A segunda atividade de manutenção foi a criação de um novo tipo de cálculo de salário composto por um salário base acrescido de uma comissão. Estes dois requisitos definem a solicitação de mudança a ser implementada.

Figura 36: Dashboard construído para o experimento



Fonte: Elaborado pelo autor

O procedimento de análise dos resultados foi a partir da comparação dos dados dos dois grupos. A variável independente do estudo é a utilização das informações qualitativas durante as atividade de manutenção. Foi investigado o impacto desta variável independente nas seguintes variáveis dependentes:

- **Corretude:** a corretude da atividade de manutenção da aplicação é garantida quando o código produzido pelos grupos produz o resultado esperado conforme o requisito definido anteriormente. Se a primeira atividade produz o resultado correto, é atribuído o valor 1. Sendo esse, o mesmo critério para segunda atividade. Considerando que um participante implementou de forma incorreta as duas atividades, o resultado final será 0. Esta variável foi verificada para cada uma das atividade de manutenção;
- **Esforço:** esta variável mede o tempo gasto pelos participantes para implementar as solicitações de mudança descritas na Tabela 2. Cada atividade de implementação considera os diferentes requisitos e o esforço de implementação de cada atividade. Comparando os valores (em minutos) assumidos por essas variáveis, podemos entender o impacto do uso das informações qualitativas no esforço de implementação;
- **Manutenabilidade:** esta variável mede se o participante implementou o código de acordo com a arquitetura do sistema considerando os padrões de projeto já existentes. Cada atividade de implementação considera diferentes aspectos de manutenção para a definição do resultado final de acordo com o descrito na Tabela 6.

Ou seja, a pontuação final em relação à manutenabilidade pode ser 5 para cada um dos participantes. Entretanto, essa variável é medida a através de uma taxa que será calculada pela

Tabela 6: Pontuação das atividades em relação a Manutenibilidade

Atividade	Requerimento	Alterações necessárias	Pontuação
1	Cálculo de imposto	Implementar o cálculo na classe <i>PayrollCalculationService</i>	1
		Utilizar as variáveis pré-definidas	1
2	Novo tipo de salário	Estender a classe <i>CommissionEmployee</i> ou a classe <i>Employee</i>	1
		Utilizar a classe <i>PayrollCalculationFactory</i> para implementar o novo cálculo	2

Fonte: Elaborado pelo autor

soma das pontuações obtidas nas duas atividades, dividindo-se o resultado por 5. A Equação 1 mostra como a taxa de manutenibilidade será calculada.

$$\left(\sum (PontuacaoAtividade1) + \sum (PontuacaoAtividade2) \right) / 5. \quad (1)$$

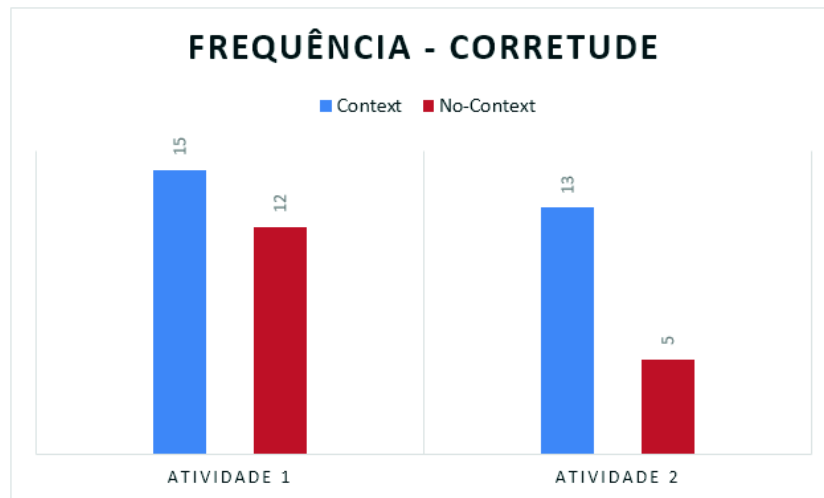
Para obtenção das pontuações dos dois grupos de participantes, foram realizadas análises de revisão e teste de código nos 30 projetos de implementação em relação aos itens descritos. Os dados foram tabulados, avaliados e as hipóteses testadas. Ao final, foram comparados os desenvolvimentos dos dois grupos. Os resultados deste experimento controlado sugerem que o uso de informações do contexto em formato de *dashboard* qualitativo geraram um código alterado com maior corretude e manutenibilidade. Verificou-se também que o Grupo 1 realizou as implementações com menor esforço. A análise de cada uma das variáveis será descrita a seguir:

- **Corretude:** a Figura 37 mostra a pontuação referente à corretude obtida pelos participantes nas atividades de manutenção. Neste gráfico, *Context* representa a soma de tarefas realizadas corretamente pelos participantes do Grupo 1, em que as atividades foram realizadas com o uso das informações qualitativas em formato de *dashboard*. *No-Context* representa a soma das tarefas implementadas corretamente realizadas pelo Grupo 2, onde as atividades foram realizadas sem o uso das informações qualitativas. A pontuação final do Grupo 1 foi de 28 e a pontuação do Grupo 2 foi 17. A expectativa inicial era de que a soma das tarefas implementadas corretamente pelo Grupo 1 poderia ser superior a pontuação do Grupo 2. Com a expectativa confirmada, é possível considerar que o uso de informações qualitativas em formato de *dashboard* melhora a corretude do código produzido quando comparado com o contrário.

A hipótese H_{1-1} foi testada com a utilização do teste *Pearson Chi-square* e a Tabela 7 mostra os resultados obtidos. O valor do *Chi-square* obtido pelo teste foi de 10,756 e o valor do *p-value* foi 0,001. Considerando que o *p-value* é menor que 0,05, é possível concluir que há uma diferença estatisticamente significativa para a corretude do código-fonte produzido, associadas ao uso das informações qualitativas de contexto (*dashboard*).

- **Esforço:** a Tabela 8 mostra a estatística descritiva do esforço (em minutos) investido pelos participantes nas atividades de manutenção. Nesta tabela, a linha *Context* representa

Figura 37: Corretude: pontuação geral



Fonte: Elaborado pelo autor

Tabela 7: Teste *Pearson Chi-square* para corretude

Atividade	Estatística	Corretude
Todas	<i>p-value</i>	0,001
	X^2	10,756

X^2 = Pearson's Chi-square, $\alpha = 0.05$

as estatísticas esforço de manutenção do Grupo 1, em que as atividades foram realizadas com o uso do *dashboard*. A linha *No-Context* contém as estatísticas do esforço de manutenção do Grupo 2, onde as atividades foram implementadas sem o uso do *dashboard*. O tempo médio gasto para o Grupo 1 implementar ambas as atividades de manutenção foi de 19,67 minutos, enquanto o Grupo 2 levou, em média, 48,47 minutos. Percebe-se uma diferença significativa entre os grupos em termos de esforço. O Grupo 1 finalizou as atividades de manutenção, em média, 28,28 minutos (40,58%) mas rapidamente que o Grupo 2. A expectativa inicial era que o esforço investido pelo Grupo 1 pudesse ser menor do que o Grupo 2. Com a expectativa confirmada, é possível considerar que o uso das informações qualitativas reduziu o esforço de manutenção necessário para realizar a mudanças solicitadas. O próximo passo é testar se esta diferença é estatisticamente significativa.

Para aplicar o teste de hipóteses correto, utilizamos os testes de *Kolmogorov-Smirnov* e *ShapiroWilk* (DEVORE; FARNUM, 1999) para verificar a distribuição normal dos dados. A Tabela 9 mostra os resultados. Observando que os valores de *p-value* são inferiores a 0,05, os dados coletados não estão normalmente distribuídos. Assim, o teste de *Mann-Whitney* foi aplicado para testar H_{2-1} . O *p-value* obtido pelo teste foi 0,002. Considerando que o valor *p-value* é menor que 0,05, há evidência suficiente para rejeitar a hipótese nula. Portanto, os resultados sugerem que o esforço de manutenção investido pelos

Tabela 8: Estatística descritiva do esforço de manutenção

	N	Min	Med	IQR	Max	Mean	SD
Context	30	2	17	20.25	70	19.67	16.12
No-Context	30	3	30.35	47	180	48.47	47.87
Diferença	0	1	13.50	26.75	110	28.28	31.75

N #atividades, Min: mínimo, Med: mediana, Max: máximo

SD: desvio padrão, IQR: variação interquartil, Mean: média

Fonte: Elaborado pelo autor

participantes do Grupo 1 (com *dashboard*) para implementar solicitações de mudança é significativamente menor do que o esforço investido pelo Grupo 2 (sem *dashboard*) para realização das mesmas atividades.

Tabela 9: Teste de normalidade - Esforço

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Estatística	N	p-value	Estatística	N	p-value
Context	0.161	30	0.046	0.860	30	0.001
No-Context	0.278	30	0.000	0.777	30	0.000

Fonte: Elaborado pelo autor

- **Manutenabilidade:** a Tabela 10 mostra a estatística descritiva referente a taxa de manutenção do código-fonte alterado pelos participantes. A linha *Context* representa as estatísticas de manutenabilidade do Grupo 1, em que as atividades foram realizadas com a utilização do *dashboard*. A linha *No-Context* contém as estatísticas de taxa de manutenabilidade do Grupo 2, onde as atividades foram implementadas sem o uso do *dashboard*. A média da taxa de manutenabilidade do código-fonte alterado pelo Grupo 1 foi de 0,8133 enquanto que a taxa média de manutenabilidade produzida pelo Grupo 2 foi de 0,6133. Percebe-se uma diferença significativa entre as taxas obtidas pelos dois grupos de participantes. O Grupo 1 produziu um código-fonte com uma taxa de manutenabilidade média de 24,60% superior à taxa do Grupo 2. A expectativa inicial era de que a manutenabilidade do código-fonte alterado pelo Grupo 1 poderia ser superior ao Grupo 2. Esta expectativa foi confirmada. É possível considerar que o uso de informações qualitativas melhorou a manutenabilidade do código-fonte produzido pelo Grupo 1.

Os testes de *Kolmogorov-Smirnov* e *ShapiroWilk* foram utilizados novamente para verificar a distribuição normal dos dados coletados. A Tabela 11 mostra os resultados. Uma vez que os resultados indicam desvios de normalidade, o teste *Mann-Whitney* foi aplicado para avaliação da hipótese H_{3-1} . O *p-value* obtido foi 0,026. Considerando que o valor do *p-value* é menor do que 0.05, a hipótese nula pode ser rejeitada. Isto significa que a manutenabilidade do código-fonte alterado pelos participantes que utilizaram as informações qualitativas (*dashboard*) é significativamente maior do que a manutenabilidade do

Tabela 10: Estatística descritiva da taxa de manutenibilidade

	N	Min	Med	IQR	Max	Mean	SD
Context	15	0.40	0.8000	0.20	1.00	0.8133	0.15976
No-Context	15	0.20	0.6000	0.40	1.00	0.6133	0.24456
Diff	0	0.20	0.2000	0.20	0	0.2000	0.08480

N #atividades, Min: mínimo, Med: mediana, Max máximo

SD: desvio padrão, IQR: variação interquart, Mean: média

Fonte: Elaborado pelo autor

código produzido pelo Grupo 2.

Tabela 11: Teste de normalidade - Manutenibilidade

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Estatística	N	p-value	Estatística	N	p-value
Context	0.333	15	0.000	0.782	15	0.002
No-Context	0.208	15	0.079	0.914	15	0.155

Fonte: Elaborado pelo autor

6.3 Estudo de caso na indústria

Na primeira avaliação realizada, todos os aspectos de desenvolvimento foram controlados através do experimento. Todos os participantes realizaram as mesmas atividades utilizando versões iguais de código-fonte e em um ambiente artificial. A segunda avaliação deste estudo se deu a partir da utilização do protótipo em um ambiente real de uma empresa de desenvolvimento de *software*. Nesta avaliação não foi possível controlar as variáveis citadas anteriormente. Desta forma, foi necessário preparar o protótipo através de cargas no banco de dados bem como a configuração dos valores limites das métricas. A Seção 6.3.1 descreve como ocorreu a preparação do protótipo para a avaliação. A Seção 6.3.2 apresenta o questionário que foi abordado no estudo de caso, bem como uma análise das respostas preenchidas pelos desenvolvedores.

6.3.1 Preparação do protótipo

Todas as informações que compuseram o banco de dados do protótipo foram extraídas de aplicações e bancos de dados existentes na empresa e importadas através de *scripts*. A extração das métricas de código e registro de alterações ocorreu a partir da construção de uma ferramenta que foi integrada ao ambiente de desenvolvimento dos programadores. A construção desta ferramenta foi um fator crítico no período de avaliação, uma vez que as métricas necessárias para composição do modelo não estavam disponíveis em tabelas estruturadas ou arquivos.

Uma das características implementadas no protótipo foi a configuração de valores limites para as métricas e indicadores compostos. Baseado nesses valores limites, o protótipo realiza a

avaliação do histórico de contexto determinando o estado atual do código-fonte. Esta avaliação resulta em uma indicação de pontos de alerta no *dashboard* do protótipo. Antes do protótipo ser disponibilizado para os desenvolvedores, foi realizada uma seção de calibragem do modelo junto ao arquiteto principal de desenvolvimento da equipe. Nesta calibragem, foi possível determinar as métricas e os valores limites associados a cada uma delas baseando-se nas melhores práticas e guias existentes na organização. As métricas de *software* primárias disponibilizadas durante o período de avaliação, foram exclusivas de classes e estão descritas na Tabela 12.

Tabela 12: Métricas primárias

Métrica primária	Descrição
Complexidade ciclomática	Considera o número de fluxos que um determinado trecho de código pode executar.
Número de linhas de código	Número total de linhas de código desconsiderando linhas em branco e comentários.
Número máximo de blocos aninhados	Representa o número máximo de blocos aninhados.
Número de dependências	Número de objetos que utilizam a classe.
Número de atributos	Número de atributos da classe.
Número de métodos	Número de métodos da classe.
Índice de cobertura de teste unitário	Percentual de cobertura de testes unitário automatizado da classe.
Número de comandos executáveis	Número de comandos executáveis da classe, não considerando definições de variáveis, assinaturas de métodos e demais definições.

Fonte: Elaborado pelo autor

Além das métricas primárias descritas anteriormente, os seguintes indicadores compostos foram elaborados e disponibilizados para os desenvolvedores durante o período de avaliação do *SW-Context*:

- **Fator de complexidade:** esta composição foi definida para avaliar a capacidade do protótipo calcular composições ponderadas. A configuração desta composição pode representar o fator de complexidade da classe, entretanto esta questão não foi abordada no período de avaliação do protótipo. A configuração desta composição está descrita na Tabela 13.

Tabela 13: Configuração da composição - Ponderada

Métrica primária	Operação	Fator de cálculo	Inversamente proporcional
Número de métodos	*	0,05	Não
Número de dependências	*	0,05	Não
Complexidade ciclomática	*	0,05	Não
Número de linhas de código	*	0,02	Não
Índice de cobertura de teste unitário	*	0,01	Sim

Fonte: Elaborado pelo autor

- **Número de dependências e Índice de cobertura de teste unitário:** indicador definido para avaliar a capacidade do protótipo calcular composições simples, onde as métricas configuradas são comparadas com um valor limite. Se todas as métricas estiverem com seus valores dentro dos limites configurados, a composição é considerada satisfatória. A configuração desta composição está descrita na Tabela 14;

Tabela 14: Configuração da composição - Simples

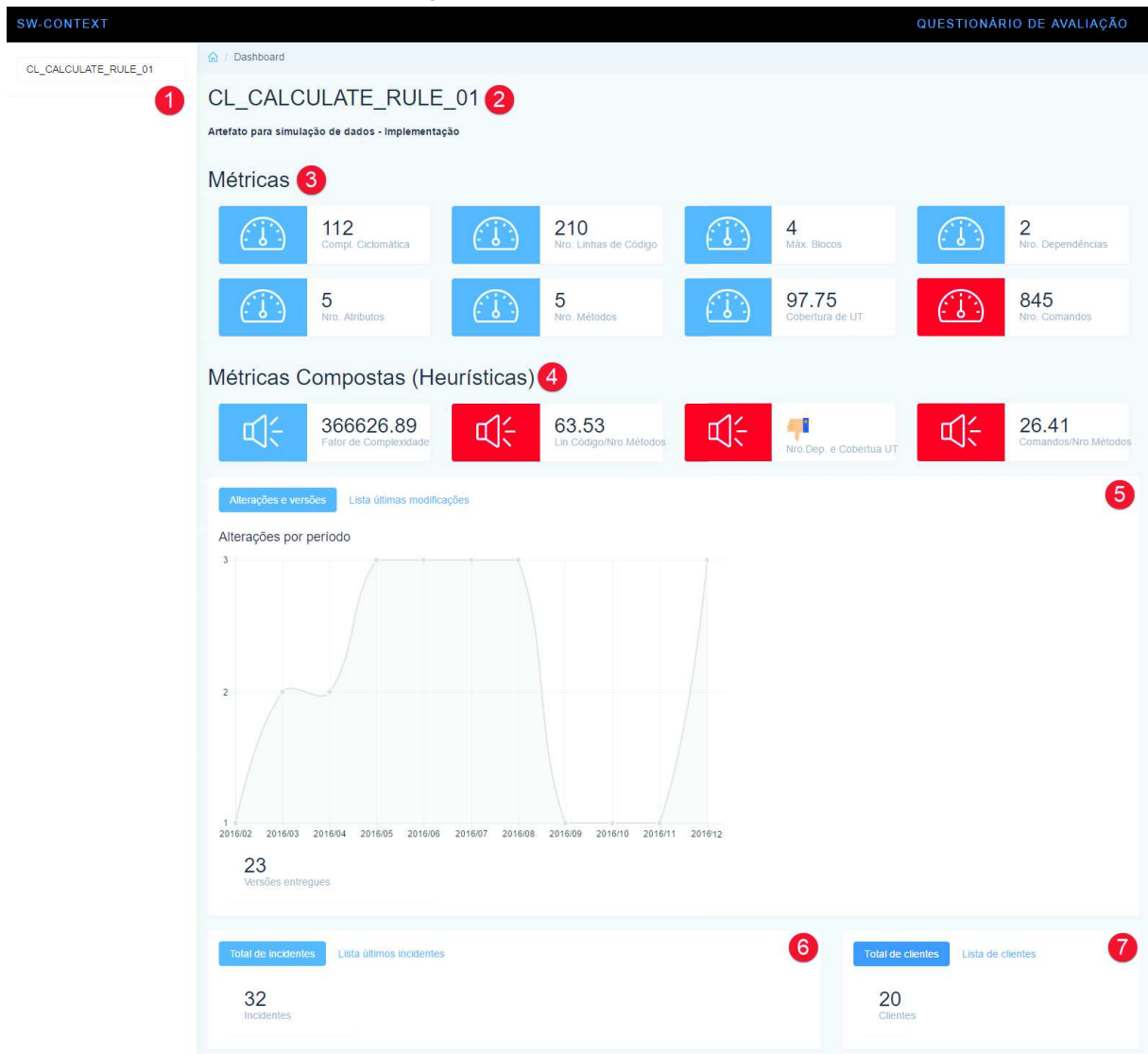
Métrica primária	Comparação	Valor limite
Número de dependências	>	10
Índice de cobertura de teste unitário	<	80

Fonte: Elaborado pelo autor

- **Linhas de Código por Método:** calcula a proporção de linhas de código por método da classe através de uma operação matemática de divisão;
- **Número de comandos por Número de métodos:** calcula a proporção de número de comandos executáveis por método da classe através de uma operação matemática de divisão;

Adicionalmente aos valores das métricas de código e alterações realizadas, foram extraídos os incidentes relacionados aos artefatos de código, bem como a identificação dos clientes que realizaram a abertura destes incidentes. Essas informações estão armazenadas em uma ferramenta específica de gerenciamento de incidentes, de forma que montar a relação de artefato *versus* incidente se tornou um desafio semelhante à construção da ferramenta de extração de dados anteriormente citada. Um fator crítico de sucesso para a composição deste relacionamento foi a existência de um identificador único que fica armazenado em cada versão do artefato e aos seus incidentes, mesmo em bancos de dados e aplicações distintas. De posse destas ferramentas construídas, foi possível disponibilizar o protótipo para avaliação com as seguintes informações (Figura 38):

- Histórico de métricas primárias - Área 3;
- Histórico de indicadores compostos - Área 4;
- Lista de alterações do artefato contendo data, identificação e descrição - Área 5;
- Identificação dos desenvolvedores que realizaram a alteração - Área 5;
- Lista de incidentes relacionados aos artefatos - Área 6;
- Identificação dos clientes que criaram os incidentes - Área 7.

Figura 38: *Dashboard* utilizado

Fonte: Elaborado pelo autor

6.3.2 Questionário

O questionário elaborado para avaliação do modelo possui questões e sentenças que visam caracterizar a experiência profissional e formação dos participantes, classificar a atividade de manutenção que em que o protótipo foi utilizado (Tabela 15) e, principalmente, avaliar o modelo. As perguntas de avaliação do *SW-Context* foram elaboradas com base nos conceitos do modelo de aceitação de tecnologia (TAM - *Technology Acceptance Model*) proposto por DAVIS (1989) (Tabela 16). O modelo TAM considera os seguintes itens como principais influências para a aceitação de uma nova tecnologia:

- **Facilidade de uso percebida:** grau em que uma pessoa acredita que a tecnologia poderia diminuir os seus esforços;

- **Utilidade percebida:** grau em que uma pessoa acredita que a tecnologia poderia melhorar o desempenho no desenvolvimento de suas atividades.

As opções de respostas para as perguntas que avaliaram a utilidade e facilidade de uso do protótipo seguiram o padrão da escala Likert (LIKERT, 1932) de cinco pontos, variando entre 1 (discordo totalmente) até 5 (concordo totalmente). Adicionalmente, o questionário foi composto com mais 2 perguntas que avaliaram a percepção geral dos participantes em relação ao protótipo utilizado. Estas questões estão descritas na Tabela 16. Após a utilização do protótipo, os 12 participantes responderam às perguntas descritas nas Tabelas 15 e 16 a fim de avaliar sua experiência com a ferramenta.

Tabela 15: Questionário - perguntas de caracterização

Categoria	Número da questão	Pergunta
Caracterização do Participante	1	Qual o seu grau de escolaridade?
	2	Qual a situação do seu grau de escolaridade?
	3	Qual a sua experiência em desenvolvimento de software?
	4	A quanto tempo você trabalha com desenvolvimento de <i>software</i> na empresa?
Classificação da Atividade	5	Qual foi a atividade de manutenção realizada?

Fonte: Elaborado pelo autor

Tabela 16: Questionário - perguntas sobre utilidade e facilidade de uso

Categoria	Número da questão	Pergunta
Utilidade do Sistema	6	As informações contextuais caracterizadas pelas métricas de código auxiliaram o desenvolvedor a ter uma visão mais ampla do objeto em questão?
	7	A consolidação das métricas de código em uma visão gráfica contribuiu para o entendimento da situação atual da classe/artefato em termos de manutenibilidade?
	8	A composição de mais de uma métrica em um novo indicador permitiu a avaliação de critérios adicionais enriquecendo a consciência situacional do artefato de código em questão?
	9	A diferenciação de valores limites para códigos de acordo com a qualificação do código-fonte (<i>Framework</i> ou <i>Implementação</i>) permitiu uma avaliação mais assertiva sobre os artefatos?
	10	As informações referentes às últimas alterações de código, relacionadas com os desenvolvedores que efetuaram as alterações, possibilitaram um entendimento mais amplo sobre o porquê da situação atual do projeto em relação aos valores das métricas?
	11	O conjunto total de informações e relações disponibilizadas pelo modelo <i>SW-Context</i> contribuíram para o entendimento do impacto das alterações?
Facilidade de Uso	12	O serviço que alerta as métricas, que possuem valor excedente ao limite configurado, contribuiu para alguma tomada de decisão em relação a refactoring, bem como a adequação do objeto às melhores práticas?
	13	A identificação dos desenvolvedores que realizaram as últimas alterações facilitou o esclarecimento de dúvidas sobre o objeto em alteração?
	14	O conjunto total de informações e relações disponibilizadas pelo modelo <i>SW-Context</i> facilitaram a análise das alterações realizadas?
	15	A utilização do modelo <i>SW-Context</i> , no meu dia-a-dia, facilitaria a minha compreensão em relação aos artefatos de código?
Percepção Geral	16	A disponibilização das mesmas informações contextuais, apresentadas em um nível mais abstrato (pacote, projeto ou módulo), dariam uma informação mais ampla em relação ao estado atual do <i>software</i> como um todo?
	17	Que informações/funcionalidades poderiam ser adicionadas ao modelo, a fim de prevenir a degradação da arquitetura dos artefatos?

Fonte: Elaborado pelo autor

6.3.3 Resultados do questionário

Em relação aos questionamentos que caracterizaram a formação acadêmica e profissional dos participantes verificou-se um público bem diverso que possui, minimamente, graduação em andamento. Do total de participantes, 67% possui ou está cursando graduação, 25% possui o nível de pós-graduação ou está com o seu curso em andamento e finalmente 8% dos participantes possui o título de mestre ou está igualmente cursando o mesmo. 82% dos participantes possui seu curso de formação concluído.

A Figura 39 mostra os resultados em relação à experiência dos participantes em desenvolvimento de *software*. Nenhum dos participantes possui menos de 2 anos de experiência em desenvolvimento. 67% dos desenvolvedores que avaliaram o modelo possuem mais de 7 anos de experiência. 17% dos participantes possui entre 5 e 7 anos de experiência de desenvolvimento de *software*. Por fim, 16% dos participantes possui entre 2 e 5 anos de experiência.

Em relação à experiência dos participantes na empresa, tem-se uma distribuição mais hete-

Figura 39: Resultado da questão 3



Fonte: Elaborado pelo autor

rogênea. 34% dos participantes possui entre 5 e 7 anos de empresa. Com mais de 7 anos de experiência na empresa, tem-se 33% dos participantes. Ou seja, 67% dos desenvolvedores que avaliaram o modelo possui pelo menos 5 anos de trabalho na empresa. 25% dos desenvolvedores possui entre 2 e 5 anos e finalmente 8% dos participantes possui menos de um ano de serviços prestados. Estes resultados podem ser verificados na Figura 40.

Figura 40: Resultado da questão 4



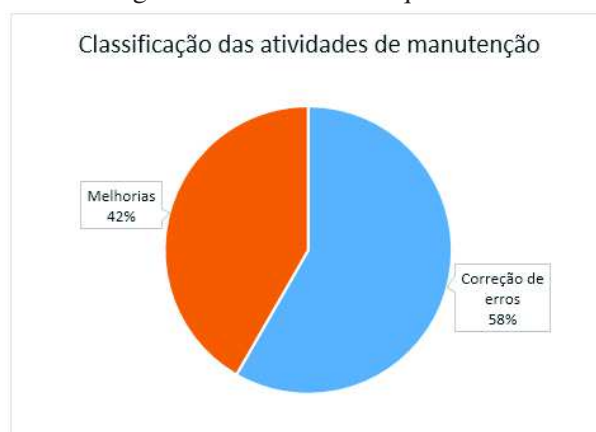
Fonte: Elaborado pelo autor

O principal ponto que pode ser percebido nas respostas das questões 1 a 4, é o tempo de experiência em desenvolvimento de *software* dos participantes da avaliação. Conforme citado anteriormente, nenhum dos desenvolvedores possui menos de 2 anos de envolvimento com desenvolvimento de aplicações. Além deste fato, mais da metade dos participantes possuem pelo menos 5 anos de experiência em projetos de desenvolvimento de sistemas na empresa. Considera-se que esses fatores contribuem para uma avaliação qualificada do trabalho em questão.

Durante o período de avaliação, os desenvolvedores utilizaram o protótipo desenvolvido em atividades específicas de manutenção. Como pode ser verificado na Figura 41, 42% das atividades que foram realizadas foram caracterizadas como melhorias, ou seja, implementação de

novas funcionalidades e comportamentos oriundos de projetos novos. O restante das atividades, 58%, foram classificadas como correções de incidentes, originadas de incidentes de clientes externos e/ou internos.

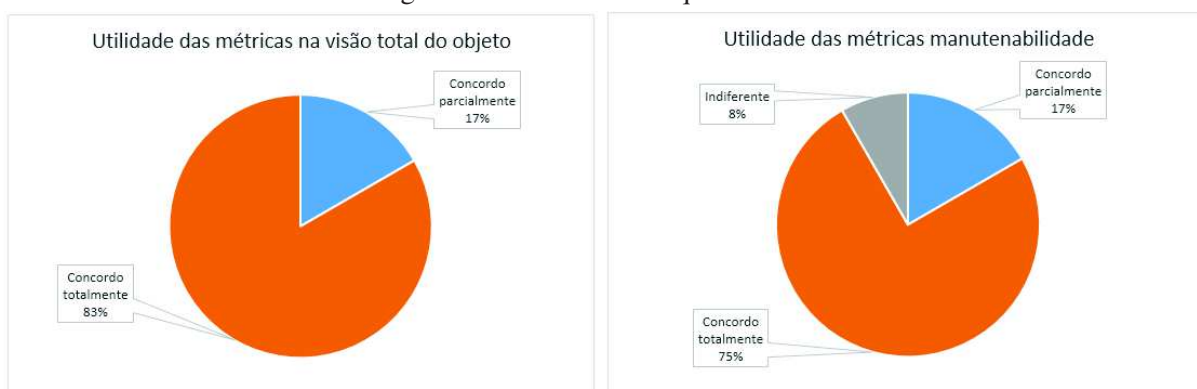
Figura 41: Resultado da questão 5



Fonte: Elaborado pelo autor

As questões que envolveram a utilidade do sistema, bem como a facilidade de uso percebida apresentaram percentuais com resultados positivos. As questões 6 e 7 abordaram a utilidade das métricas em relação à situação geral do artefato em questão, bem como os aspectos de manutenibilidade. Todos os participantes afirmaram que as informações representadas através de métricas foram úteis para o entendimento geral da classe em que estavam trabalhando, sendo 83% concordando totalmente com a afirmação e 17% concordando parcialmente. Em relação aos aspectos de manutenibilidade, 75% dos desenvolvedores entenderam que as informações disponibilizadas foram úteis para o entendimento, concordando totalmente. 17% dos participantes concordaram parcialmente e 8% afirmaram serem indiferentes a esta afirmação. A Figura 42 mostra os resultados das questões 6 e 7.

Figura 42: Resultado das questões 6 e 7

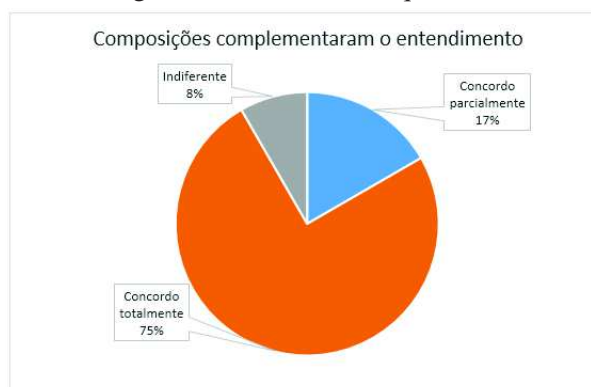


Fonte: Elaborado pelo autor

A utilidade das métricas compostas também foi avaliada, através da questão 8 do questionário.

nário. Conforme mostra a Figura 43, 75% dos desenvolvedores concordaram totalmente que as composições de métricas foram úteis, complementando o entendimento sobre a classe em questão. 17% dos participantes concordaram parcialmente e 8% foram indiferentes. Verifica-se que os resultados da questão 8 obtiveram exatamente os mesmos percentuais da questão 7. Os desenvolvedores demonstraram a mesma opinião, tanto em relação à utilidade das métricas primárias quanto em relação à utilidade das métricas compostas.

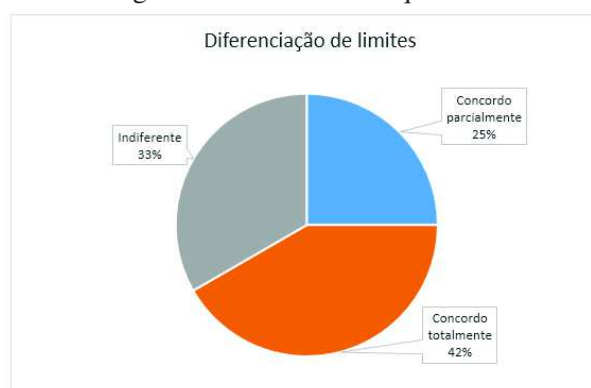
Figura 43: Resultado da questão 8



Fonte: Elaborado pelo autor

O protótipo implementou a configuração de valores limites para as métricas primárias e compostas. Nesta configuração, é possível diferenciar os valores limites de acordo com o tipo de classe, permitindo uma parametrização qualificada para os artefatos de implementação e *framework*. A Figura 44 mostra os resultados da questão 9, onde 42% dos desenvolvedores concorda totalmente com a utilidade desta característica do modelo. 25% respondeu concordar parcialmente com a utilidade desta possibilidade, e por fim, 33% se mostrou indiferente à diferenciação dos valores limites.

Figura 44: Resultado da questão 9

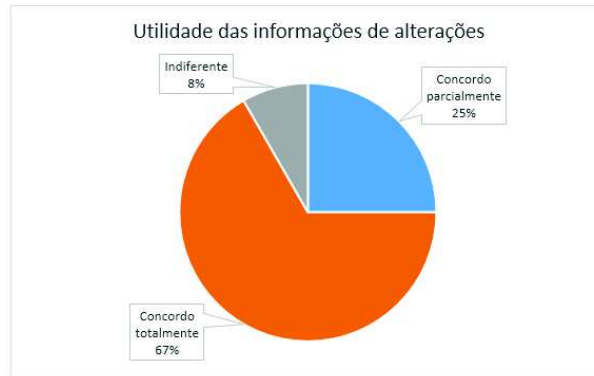


Fonte: Elaborado pelo autor

Com relação à utilidade das informações das últimas alterações realizadas no código combinadas com os dados dos desenvolvedores que realizaram as alterações, 67% dos participantes

responderam concordar totalmente enquanto 25% concordaram parcialmente. Apenas 8% dos desenvolvedores se mostrou indiferente com a utilidade destas informações. A Figura 45 mostra os percentuais resultantes da questão 10.

Figura 45: Resultado da questão 10

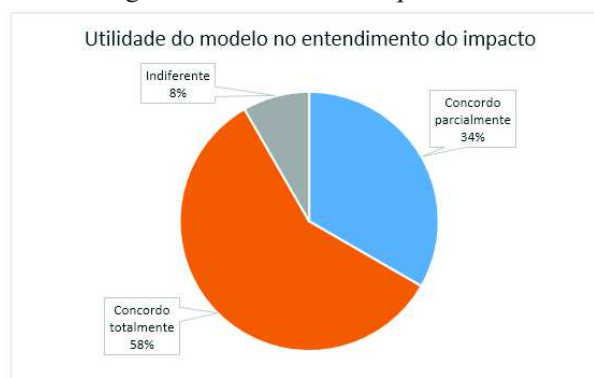


Fonte: Elaborado pelo autor

A questão 11 abordou a utilidade geral do modelo para o entendimento do impacto das mudanças que estavam sendo realizadas. O percentuais obtidos nas respostas foram os mesmos da questão 10, onde, 67% dos desenvolvedores concordaram totalmente e 25% concordaram parcialmente. 8% dos participantes demonstraram indiferença em relação à utilidade do modelo para o entendimento do impacto das alterações. Os resultados anteriormente descritos podem ser verificados na Figura 46.

De forma geral, nenhuma questão que abordou a utilidade de uso do modelo apresentou respostas de discordância, o maior percentual obtido de respostas onde os desenvolvedores se mostraram indiferentes foi na questão 9. O percentual obtido foi de 33% dos participantes. Acredita-se que este resultado deve-se ao fato dos desenvolvedores não terem alterado artefatos de diferentes classificações (implementação/*framework*) durante o período de avaliação, que durou 30 dias.

Figura 46: Resultado da questão 11



Fonte: Elaborado pelo autor

As questões 12 a 15 abordaram a facilidade de uso percebida pelos desenvolvedores, de forma que seus esforços sejam reduzidos. A questão 12 abordou o quanto o serviço de alertas de métricas que estão com valores excedentes aos limites configurados facilitou a tomada de decisão para adequação do artefato às melhores práticas estabelecidas pela empresa. Apenas 17% dos desenvolvedores se mostrou indiferente nos questionamentos, 67% concordou totalmente e, por fim, 16% concordou parcialmente com a facilidade de uso do serviço de alertas. A Figura 47 mostra os resultados obtidos na questão 12.

Figura 47: Resultado da questão 12



Fonte: Elaborado pelo autor

A questão 13 tratou da facilidade de uso percebida em relação ao esclarecimento de dúvidas sobre o objeto em alteração a partir da identificação dos desenvolvedores que realizaram alterações anteriores. Ou seja, o quanto essas informações aceleraram a busca de um possível especialista no assunto que estava sendo tratado na atividade de manutenção. A Figura 48 mostra os resultados obtidos na questão 13 onde, 75% dos desenvolvedores concordaram totalmente e 8% concordaram parcialmente. Por fim, 17% dos desenvolvedores se mostraram indiferentes.

Figura 48: Resultado da questão 13

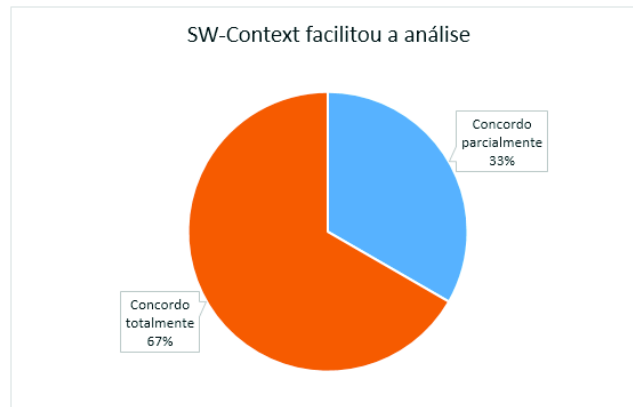


Fonte: Elaborado pelo autor

A facilidade de uso percebida pelos desenvolvedores em relação a todas as informações dis-

ponibilizadas pelo protótipo do *SW-Context* foi abordada na questão 14. Os percentuais obtidos nas respostas destas questões foram positivos. Todos os participantes afirmaram que o conjunto total das informações disponibilizadas facilitou a análise das alterações realizadas. A Figura 49 mostra os resultados, onde 67% concordaram totalmente e 33% concordaram parcialmente com a facilidade de uso percebida ao utilizar o conjunto de todas as informações.

Figura 49: Resultado da questão 14



Fonte: Elaborado pelo autor

A questão 15 abordou o quanto a utilização do modelo *SW-Context* que, de forma constante, facilitaria a compreensão sobre os artefatos de código nas atividades diárias dos desenvolvedores. Da mesma forma que a questão 14, os resultados obtidos podem ser considerados satisfatórios em relação ao uso contínuo da ferramenta. Na Figura 50 podem ser verificados estes resultados de modo que, 58% dos participantes concordaram totalmente e 42% concordaram parcialmente. Ou seja, 100% dos participantes concordam de alguma forma que a utilização constante o modelo *SW-Context* facilitaria a compreensão dos deles sobre os artefatos de código que fazem parte da sua atividade diária.

Figura 50: Resultado da questão 15

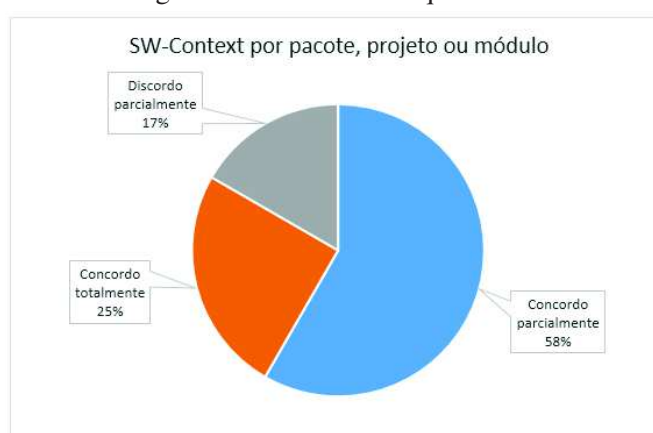


Fonte: Elaborado pelo autor

As informações contextuais disponibilizadas pelo protótipo são relacionadas diretamente

aos artefatos de código ao nível de classes. A questão 16 abordou a percepção dos desenvolvedores em relação a disponibilização das mesmas informações em um nível mais abstrato, por exemplo, projeto, módulo ou pacote. Esta questão foi a única que apresentou um percentual de respostas da opção discordo parcialmente. 17% dos participantes discordaram de que as informações em níveis mais abstratos trariam um maior entendimento sobre o *software*/aplicação que está sendo alterado. 58% concordaram parcialmente com esta possibilidade e 25% concordaram totalmente. Uma das possibilidades deste percentual de respostas com discordância pode ser que, uma visão mais abstrata, seja mais útil para gestores e/ou arquitetos do que para os desenvolvedores que necessitam alterar artefatos individualmente.

Figura 51: Resultado da questão 16



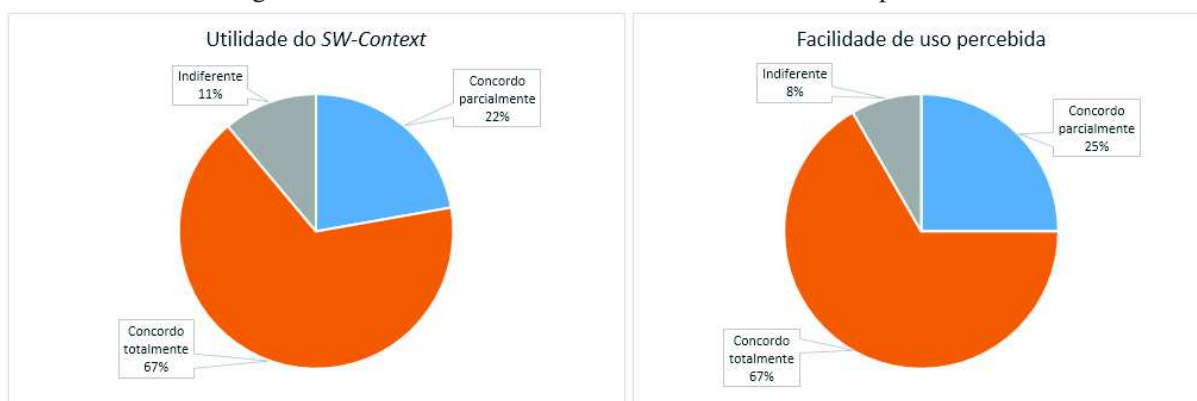
Fonte: Elaborado pelo autor

A questão 17 perguntou, de forma aberta, aos desenvolvedores que informações/funcionalidades poderiam ser adicionadas ao modelo a fim de prever a degradação da arquitetura dos artefatos. Esta questão era opcional e nem todos os participantes responderam. Entretanto, as respostas obtidas podem ser consideradas uma boa contribuição para trabalhos futuros. A seguir são descritas algumas das respostas obtidas:

- Inclusão da métrica de dependências dinâmicas, onde os artefatos são referenciados através de códigos gerados em tempo de execução ou registros em banco de dados;
- Possibilitar ao desenvolvedor documentar a criação ou alteração do artefato;
- Indicação de artefatos críticos, que quando alterados, enviam alertas imediatamente ainda que as métricas estejam de acordo com os limites configurados;
- Ampliação da funcionalidade de pesquisa possibilitando filtros por período de alteração e desenvolvedor que alterou;
- Inclusão de métricas que indiquem o nível de acoplamento com outros componentes/pacotes;
- Indicar o nível de reimplementação de classes e interfaces.

A Figura 52 mostra os resultados agrupados de todas as questões que abordaram facilidade de uso percebida e a utilidade do *SW-Context*. De uma forma geral, a avaliação do modelo obteve respostas com percentuais positivos em relação aos aspectos endereçados pelo modelo de aceitação de tecnologia TAM. Como pode ser verificado, 67% dos desenvolvedores concordaram totalmente com ambos aspectos de aceitação de tecnologia em relação ao modelo *SW-Context*. Pode-se considerar que a consolidação das informações contextuais em um local único, correlacionadas e a disponibilização gráfica das informações, através do *dashboard*, atingiu o objetivo de melhorar a consciência situacional dos desenvolvedores nas atividades de manutenção. As respostas obtidas nas questões relacionadas à percepção geral do *SW-Context* indicam o foco correto do modelo em centrar as informações contextuais nos artefatos. Aliadas a este foco, estão incluídas as indicações de informações e funcionalidades que poderiam ser adicionadas ao modelo.

Figura 52: Utilidade do *SW-Context* e facilidade de uso percebida



Fonte: Elaborado pelo autor

Um dos desafios deste estudo foi avaliar o maior número de funcionalidades implementadas no protótipo no período de 30 dias. O planejamento inicial era contar com uma equipe de 8 pessoas durante o período de 30 dias. Entretanto, não havia uma previsão exata de atividades de manutenção seriam demandadas, bem como, o período da demanda. Desta forma, o risco de não se obter um número considerável de utilizações/avaliações do modelo era alto. Para mitigar este risco, o escopo da avaliação foi ampliado em termos de número de artefatos importados no banco de dados do protótipo e potenciais avaliadores. Ao final do período de avaliação, o banco de dados do protótipo possuía 1176 artefatos importados e 22 desenvolvedores habilitados a utilizar a ferramenta nas suas atividades diárias. Destes, 12 realizaram atividades de manutenção em artefatos considerados pelo modelo.

6.4 Considerações sobre o capítulo

Este capítulo abordou em duas seções a metodologia de avaliação do modelo bem como os resultados das avaliações realizadas. Na Seção 6.2, foi descrito a avaliação inicial, realizada

através de um experimento controlado. A Seção 6.3 descreveu a avaliação final, executada na indústria através de um estudo de caso. O objetivo desta avaliação final foi verificar a aplicabilidade do modelo em um cenário real, onde as requisições de mudança ocorrem a partir das necessidades de clientes, sem data determinada, e os times de desenvolvimento precisam entregar essas demandas, através de novas funcionalidades e/ou correções, lidando com as restrições inerentes à atividade de desenvolvimento de *software* no mundo corporativo.

7 CONSIDERAÇÕES FINAIS

Esta dissertação apresentou *SW-Context* que é um modelo focado na definição de informações de contexto para *software*, armazenamento dessas informações em histórico de contextos e disponibilização qualitativa da informação, a fim de auxiliar o dia-a-dia dos desenvolvedores de *software*. O modelo permite a configuração de valores limites de métricas de *software*, que são utilizados na análise do histórico de contextos a fim de prevenir a degradação da arquitetura de aplicações. As informações armazenadas e calculadas pelo modelo são disponibilizadas por meio de *web services*, e consumidas por telas que interpretam as informações e as representam graficamente.

O Capítulo 2 introduziu os principais conceitos sobre contexto e histórico de contextos, sua relação com a engenharia de *software* e por fim sobre *software analytics*. No Capítulo 3 foram apresentados trabalhos relacionados ao *SW-Context*, bem como uma relação de critérios de comparação. Após a comparação entre os trabalhos relacionados, o Capítulo 4 descreveu o modelo *SW-Context*, com seus requerimentos, a visão geral e a arquitetura.

No Capítulo 5 foram apresentados os aspectos de implementação do modelo *SW-Context*, com uma visão detalhada sobre os componentes de *software*, classes e serviços implementados. O Capítulo 6 abordou as duas avaliações realizadas em relação do modelo *SW-Context*, sendo a primeira um experimento controlado e a segunda um estudo de caso em uma empresa de desenvolvimento de *software*. A avaliação realizada através do estudo de caso utilizou uma metodologia que possibilitou verificar a facilidade de uso e também a utilidade do sistema para os desenvolvedores. As perguntas foram respondidas após o uso do protótipo e também foram aceitas sugestões.

Por fim, pode-se concluir que os objetivos de definição de um modelo de contexto para *software*, sua representação qualitativa e utilidade foram alcançados de acordo com o que foi proposto desde o início. Os resultados das avaliações indicam que a utilização de informações contextuais de *software* em uma forma qualitativa aprimorou a consciência situacional dos desenvolvedores auxiliando-os em suas atividades de alteração de código. Ou seja, o *SW-Context* teve uma boa aceitação por parte dos desenvolvedores que atuam diariamente em atividades de manutenção e que trabalhos futuros poderiam agregar mais informações relevantes e propiciariam conclusões ainda melhores tanto para o tema quanto para a pesquisa.

7.1 Contribuições

A principal contribuição do modelo *SW-Context*, em relação aos estudos pesquisados, é a definição conceitual das informações de contexto para *software* e sua utilização, através de visualizações qualitativas, a fim de contribuir para a melhora da consciência situacional dos desenvolvedores.

Outra contribuição de destaque do *SW-Context*, é a característica de sensibilidade ao con-

texto. O modelo permite a representação das decisões de projetos e melhores práticas de programação em métricas de *software*, bem como a avaliação das possíveis modificações dos valores destas métricas em relação aos valores configurados. Essa capacidade do modelo permite aos desenvolvedores de *software*, o acompanhamento da arquitetura das aplicações, bem como, se beneficiar dos serviços proativos e se antecipar a possíveis degradações de arquitetura em tempo real.

A análise de históricos de contexto utilizando a abordagem *Software Analytics* também caracteriza-se como uma contribuição relevante. A utilização desta abordagem permite a obtenção de percepções e dados, que podem estar implícitos no histórico de contexto armazenado, através da correlação das informações de diferentes dimensões e categorias. Foi possível realizar análises qualitativas através de correlações das informações de contexto e vincular as mesmas às modificações contextuais consideradas relevantes conforme parâmetros configurados.

As interfaces de entrada de dados do modelo permitem o armazenamento de informações contextuais de aplicações de diferentes tecnologias, caracterizando-se como genérico neste aspecto. Ou seja, diferentes times de desenvolvimento poderão acompanhar a evolução contextual das suas aplicações independente da linguagem de programação das mesmas. Para tanto, basta que estas aplicações consigam disponibilizar as informações desejadas em relação às métricas de código e suas alterações. Já as interfaces de saída de dados, proveem um formato padrão de exposição de serviços permitindo o consumo de dados por diferentes interfaces de usuário, além da camada de visualização embarcada no próprio modelo. Desta forma, times de desenvolvimento poderão usufruir dos serviços e ter as informações contextuais integradas em diferentes dispositivos e ferramentas.

7.2 Trabalhos futuros

Como trabalhos futuros percebe-se algumas necessidades oriundas da avaliação realizadas através do estudo de caso. A ampliação das possibilidades de pesquisa de artefatos foi uma funcionalidade citada nas respostas das avaliação. O protótipo atual contempla uma pesquisa simples pelo nome do artefato, entretanto, pesquisar por período de alterações e desenvolvedores que as realizaram seriam opções que agregariam valor ao modelo. Outra necessidade elencada na avaliação foi a possibilidade de documentação dos motivos da alteração pelos desenvolvedores. Esta característica está presente no trabalho relacionado de LEANO; KASI; SARMA (2014), onde a informação obtida por desenvolvedores experientes ou especialistas é considerada uma excelente fonte de identificação de contextos de tarefas.

A agregação de novas métricas ao modelo também tornou-se evidente nas respostas obtidas na avaliação. Entretanto, já existem algumas ferramentas de mercado que possuem o enfoque específico na geração de métricas de *software*. A extensão do modelo para integração à ferra-

mentas de mercado, como o SonarQube¹, poderia ampliar as possibilidades de uso do modelo se adequando às diferentes linguagens de programação existentes.

Alguns dos trabalhos relacionados como HARON; SYED-MOHAMAD (2015) e ANTUNES; CORREIA; GOMES (2011) apresentam suas informações relacionadas aos artefatos através de ferramentas acopladas às IDEs de desenvolvimento. Como o modelo *SW-Context* disponibiliza as informações contextuais através de serviços, uma possível expansão do modelo poderia ser a criação de *plugins* de visualização das informações contextuais que possam ser integrados nas mais populares ferramentas de desenvolvimento.

A ampliação da funcionalidade de composição de métricas ponderadas também poderia ser considerada como um trabalho futuro. Esta funcionalidade considera fatores de cálculo que determinam pesos para cada uma das métricas primárias, entretanto, as atribuições do fator de cálculo e sua variação são realizadas de forma manual. Uma possibilidade seria a aplicação de algumas das técnicas de *Machine Learning* na calibragem do fator de cálculo das composições ponderadas.

Por fim, outra expansão que o modelo poderia ter é o armazenamento de informações contextuais em níveis mais altos de abstração. Atualmente o modelo está preparado para armazenar as informações contextuais no nível de artefato de código, porém, ao considerar o módulo, projeto ou aplicação o qual o artefato pertence poderia propiciar uma visão mais ampla das aplicações de *software*. Com esta expansão, seria possível realizar comparações entre projetos de acordo com as métricas armazenadas e tomar decisões gerenciais em relação aos times de desenvolvimento. Esta característica agregaria gerentes de projetos e produtos como público alvo do modelo.

¹<https://www.sonarqube.org/>

REFERÊNCIAS

- AMELLER, D.; AYALA, C.; CABOT, J.; FRANCH, X. How do Software Architects Consider Non-functional Requirements : an exploratory study., [S.l.], p. 41–50, 2012.
- ANTUNES, B.; CORREIA, F.; GOMES, P. Context Capture in Software Development. **Arxiv preprint arXiv11014101**, [S.l.], 2011.
- ANTUNES, B.; GOMES, P. Context-Based Retrieval in Software Development. **Proc. of the Doctoral Symposium on Artificial Intelligence (SDIA 2009) of the 14th Portuguese Conference on Artificial Intelligence (EPIA 2009)**, [S.l.], p. 1–10, 2009.
- BARBOSA, J. L. V.; MARTINS, C.; FRANCO, L. K.; BARBOSA, D. N. F. TrailTrade: a model for trail-aware commerce support. **Computers in Industry**, [S.l.], v. 80, p. 43–53, 2016.
- BAYSAL, O.; HOLMES, R.; GODFREY, M. W. Developer Dashboards: the need for qualitative analytics. **IEEE Software**, [S.l.], v. 30, n. 4, p. 46–52, 2013.
- BUSE, R. P. L.; ZIMMERMANN, T. Information needs for software development analytics. **Proceedings - International Conference on Software Engineering**, [S.l.], p. 987–996, 2012.
- BUSE, R. R. P. L.; ZIMMERMANN, T. Analytics for software development. **Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010**, [S.l.], p. 77–80, 2010.
- DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS quarterly**, [S.l.], p. 319–340, 1989.
- DEITEL, H. M.; DEITEL, P. J. **Java: como programar**. [S.l.]: PRENTICE HALL BRASIL, 2010.
- DEVORE, J. L.; FARNUM, N. **Applied statistics for engineers and scientists**. [S.l.]: Duxbury Press, 1999. (Statistics Series).
- DEY, A.; ABOWD, G.; SALBER, D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. **Human-Computer Interaction**, [S.l.], v. 16, n. 2, p. 97–166, 2001.
- DRIVER, C.; CLARKE, S. Context-aware trails [mobile computing]. **Computer**, [S.l.], v. 37, n. 8, p. 97–99, August 2004.
- DRIVER, C.; CLARKE, S. An application framework for mobile, context-aware trails. **Pervasive Mob. Comput.**, Amsterdam, The Netherlands, The Netherlands, v. 4, n. 5, p. 719–736, Oct. 2008.
- ENDSLEY, M. R. Toward a theory of situation awareness in dynamic systems: situation awareness. **Human factors**, [S.l.], v. 37, n. 1, p. 32–64, 1995.
- HARON, N. H.; SYED-MOHAMAD, S. M. Test and Defect Coverage Analytics Model for the assessment of software test adequacy. In: MALAYSIAN SOFTWARE ENGINEERING CONFERENCE (MYSEC), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p. 13–18.

- HOLMES, R.; MURPHY, G. Using structural context to recommend source code examples. **Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.**, [S.l.], p. 117–125, 2005.
- KERSTEN, M.; MURPHY, G. C. Using task context to improve programmer productivity. **Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '06/FSE-14**, [S.l.], p. 1—11, 2006.
- LATOZA, T. D.; TOWNE, W. B.; HOEK, A. V. D. Harnessing the Crowd : decontextualizing software work. **International Workshop on Context in Software Development**, [S.l.], p. 2–3, 2014.
- LAVALLÉE, M.; ROBILLARD, P. N. Why Good Developers Write Bad Code : an observational case study of the impacts of organizational factors on software quality. **IEEE/ACM 37th IEEE International Conference on Software Engineering Why**, [S.l.], p. 677–687, 2015.
- LEANO, R.; KASI, B. K.; SARMA, A. Recommending Task Context: automation meets crowd. **International Workshop on Context in Software Development**, [S.l.], 2014.
- LI, W.; EICKHOFF, C.; VRIES, A. P. de. Want a coffee?: predicting users' trails. In: **ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL**, 35., 2012, New York, NY, USA. **Proceedings...** ACM, 2012. p. 1171–1172. (SIGIR '12).
- LIENTZ, B.; SWANSON, E. **Software maintenance management**: a study of the maintenance of computer application software in 487 data processing organizations. [S.l.]: Addison-Wesley, 1980.
- LIKERT, R. A technique for the measurement of attitudes. **Archives of psychology**, [S.l.], 1932.
- LOU, J.-G.; LIN, Q.; DING, R.; FU, Q.; ZHANG, D.; XIE, T. Software analytics for incident management of online services: an experience report. **2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)**, [S.l.], p. 475–485, 2013.
- MARTIE, L.; HOEK, A. V. D. Context in Code Search. **International Workshop on Context in Software Development**, [S.l.], 2014.
- MCCABE, T. J. A complexity measure. **ICSE '76 Proceedings of the 2nd international conference on Software engineering**, [S.l.], p. 407, 1976.
- MENZIES, T.; ZIMMERMANN, T. Software analytics: so what? **IEEE Software**, [S.l.], v. 30, n. 4, p. 31–37, 2013.
- MENZIES, T.; ZIMMERMANN, T. Software analytics: so what? **IEEE Software**, [S.l.], v. 30, n. 4, p. 31–37, 2013.
- MOSTEFAOUI, G. K.; PASQUIER-ROCHA, J.; BRÉZILLON, P. Context-Aware Computing: a guide for the pervasive computing community. **ICPS '04: Proceedings of the The IEEE/ACS International Conference on Pervasive Services**, [S.l.], p. 39–48, 2004.
- PARNIN, C.; GÖRG, C. Building usage contexts during program comprehension. **IEEE International Conference on Program Comprehension**, [S.l.], v. 2006, p. 13–22, 2006.

PETERSEN, K.; WOHLIN, C. Context in industrial software engineering research. **2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009**, [S.l.], p. 401–404, 2009.

RAJLICH, V. Software evolution: a road map. **Proceedings IEEE International Conference on Software Maintenance. ICSM 2001**, [S.l.], p. 6, 2001.

ROSA, J.; BARBOSA, J.; KICH, M.; BRITO, L. A Multi-Temporal Context-aware System for Competences Management. **International Journal of Artificial Intelligence in Education**, [S.l.], p. 1–38, 2015.

ROSA, J.; BARBOSA, J. L. V.; RIBEIRO, G. D. ORACON: an adaptive model for context prediction. **Expert Systems with Applications**, [S.l.], v. 45, p. 56–70, 2016.

SAWADSKY, N.; MURPHY, G. Fishtail: from task context to source code examples. **Proceedings of the 1st Workshop on Developing Tools As Plug-ins**, [S.l.], p. 48–51, 2011.

SILVA, J. M.; ROSA, J. H.; BARBOSA, J. L.; BARBOSA, D. N.; PALAZZO, L. A. Content distribution in trail-aware environments. **Journal of the Brazilian Computer Society**, [S.l.], v. 16, n. 3, p. 163–176, 2010.

SMITH, A. **Who Controls the Past Controls the Future - Life Annotation in Principle and Practice**. 2008. Tese (Doutorado em Ciência da Computação) — University of Southampton, 2008.

SOLINGEN, R. van; BASILI, V.; CALDIERA, G.; ROMBACH, H. D. **Goal Question Metric (GQM) Approach**. [S.l.]: John Wiley and Sons, Inc., 2002.

TILLMANN, N.; HALLEUX, J. D.; BISHOP, J.; XIE, T. Code Hunt: context-driven interactive gaming for learning programming and software engineering. **International Workshop on Context in Software Development**, [S.l.], p. 0–1, 2014.

WEISER, M. **The Computer for the 21st Century**. 1991. 94–104 p. v. 265, n. 3.

ZHANG, D.; HAN, S.; DANG, Y.; LOU, J.-G.; ZHANG, H.; XIE, T. Software Analytics in Practice. **IEEE Software**, [S.l.], v. 30, n. 5, p. 30–37, 2013.