Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Um estudo qualitativo sobre arquitetura de *software* no desenvolvimento de sistemas reais

Izabela Vanessa de Almeida Melo

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de *Software*

Dalton Dario Serey Guerrero
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

M528e Melo, Izabela Vanessa de Almeida.

Um estudo qualitativo sobre arquitetura de *software* no desenvolvimento de sistemas reais / Izabela Vanessa de Almeida Melo. – Campina Grande, 2015.

125 f.: il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2015.

"Orientação: Prof. Dalton Dario Serey Guerrero".

Referências.

1. Engenharia de Software. 2. Arquitetura de Software. 3. Visão da Indústria. I. Guerrero, Dalton Dario Serey. II. Título.

CDU 004.41(043)

"UM ESTUDO QUALITATIVO SOBRE ARQUITETURA DE SOFTWARE NO DESENVOLVIMENTO DE SISTEMAS REAIS"

IZABELA VANESSA DE ALMEIDA MELO

DISSERTAÇÃO APROVADA EM 31/08/2015

DALTON DARIO SEREY GUERRERO, D.Sc, UFCG
Orientador(a)

TIAGO LIMA MASSONI, Dr., UFCG
Examinador(a)

ROBERTO ALMEIDA BITTENCOURT, Dr., UEFS
Examinador(a)

MARCO TÚLIO DE OLIVEIRA VALENTE, Dr., UFMG
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Desde os anos 90 a comunidade científica desempenha esforços para estudar e evoluir aspectos relacionados à Arquitetura de *Software*, aumentando seu volume de publicação a partir de 1999. O aumento significativo das publicações nos últimos anos demonstra a importância e preocupação que a academia tem com relação a essa área. Porém, a partir da troca de experiência entre pesquisadores e profissionais da área, percebe-se que a indústria não parece conhecer/utilizar o que é proposto pela academia. O contexto teórico sobre arquitetura de *software*, documentação arquitetural e verificação de conformidade arquitetural já é conhecido no meio acadêmico. Porém, qual é o contexto deles dentro da indústria? Como os profissionais definem o termo "arquitetura de *software*"? Como os profissionais realizam (se realizam) a documentação arquitetural? Como realizam (se realizam) a verificação de conformidade arquitetural? O que eles pensam sobre ferramentas de apoio à verificação de conformidade arquitetural? Para responder essas questões, realizamos um estudo qualitativo dividido em 3 etapas. Primeiro, aplicamos um *survey* exploratório com o objetivo de entender o ambiente prático para ter uma noção sobre o contexto em foco. Enviamos o questionário para 149 profissionais e 4 grupos de discussão, obtendo uma taxa de resposta de 24,1%. Na segunda etapa entrevistamos 14 profissionais voluntários que responderam o *survey* exploratório (taxa de resposta de 40%). O objetivo desta etapa foi nos aprofundarmos no contexto em foco. Por fim, nossa última etapa consistiu de um *survey* confirmatório. Enviamos o questionário para os usuários do GitHub que possuem endereços de *e-mail* visíveis e tem mais de 100 seguidores (obtivemos uma taxa de resposta de 7,74%). Como resultados principais, observamos que não há uma única definição para o termo "arquitetura de *software*", dependendo de fatores desde a experiência do profissional até a empresa em que trabalham. Além disso, existem documentações arquiteturais, mas, a sua maioria não é rigorosa, formal e não é atualizada. Nem todos realizam verificação de conformidade arquitetural, e, quando realizam, normalmente é feita de forma manual. Os principais motivos para a não documentação e/ou verificação são falta de tempo ou falta de necessidade. Por fim, as ferramentas de apoio à verificação de conformidade arquitetural não são muito utilizadas e/ou conhecidas.

Abstract

Since the 90's a big effort has been applied in the academy to study and evolve aspects related to Software Architecture, increasing the amount of publications from 1999 to the present-day. The significant rise in the number of publications in the last few years shows how much importance and concern academia gives to this particular area. However, after discussing what has been experienced by researchers and professionals of this area, it is possible to notice that the industry does not know/use what is proposed in the academy. The theoretical context about software architecture, architectural documentation and architectural conformance checking is already known. However, what about the practical/industrial context? How do the professionals define "software architecture"? How do they document (if they do) the architecture? How do they perform (if they do) architectural conformance checking? What do they think about support tools for architectural conformance checking? Aiming to answer these questions, we performed a 3-step qualitative study. Firstly, we applied an exploratory survey with the goal of understanding the practical environment and having a better notion about the context of our study. We sent the survey to 149 professionals and 4 discussion groups, getting 24.1% of response rate. In the second phase we interviewed 14 volunteered professionals who responded the exploratory survey (response rate of 40%). In this phase we wanted to deeply understand the context of your study. Finally, the last phase consisted of a confirmatory survey which goal was to confirm our findings. We sent the survey to GitHub users with public e-mails and more than 100 followers, getting 7.74% of response rate. As main results we observed that there is not a single definition of "software architecture" and that it depends on several factors, as the professional's experience and the company in which he/she works. The architecture is sometimes documented but, most of the time, the documentation is incomplete, informal and outdated. In many cases, architectural conformance checking is not performed or it is performed manually. The main reasons for not documenting or verifying the architecture are lack of time and/or need. Finally, the support tools for architectural conformance checking are not often used and not well known.

Agradecimentos

Após mais de 2 anos de muito trabalho, estresse, desânimo, empolgação, enfim, uma mistura de sentimentos e pensamentos, chego ao fim de um ciclo da minha vida. E, neste ponto em que me encontro, há muito o que e a quem agradecer! Pessoas que, direta ou indiretamente, consciente ou inconscientemente, me apoiaram e ajudaram nesse processo e que merecem meus agradecimentos. Por este motivo, não serei breve, pois aprendi (com meus pais) que a gratidão é uma das melhores virtudes de um ser humano.

Primeiramente, agradeço a Deus, pela minha vida e a vida dos que me cercam. Agradeço a Ele pelas alegrias e tristezas, soluções e obstáculos, curas e doenças, paciência e estresses, coragem e desânimo, fé e dúvida. Acredito que tudo tem um sentido e uma razão, mesmo que a desconheçamos, e sempre nos faz crescer e evoluir espiritualmente.

Aos meus pais, Elizabete e Vanio, minha fonte diária de inspiração e admiração. Pela força de vontade e dedicação, contrariando uma sociedade injusta e mostrando que é possível, por meio da educação, crescer intelectualmente e vencer na vida. São meus exemplos de seres humanos, profissionais, pais, filhos, irmãos, ... A quem eu dedico todas as conquistas que tive, tenho e terei. Obrigada pelo amor, carinho, educação, apoio, amizade, e até mesmo os puxões de orelha que vocês me deram em toda minha vida.

Ao meu irmão, Marcos (meu pequeno), minha fonte diária de amor puro e gratuito. Para quem eu me esforço para dar bons exemplos e desejo que se torne um homem de bem. Quem me estimula o sentimento protetor. Quem sempre tem uma piada na manga para dar leveza em quase todas as situações.

Às minhas avós, Helena e Enilda, simplesmente por existirem. Por criarem meus pais da melhor forma que puderam. Por sempre, incondicionalmente, acompanharem meu crescimento e me "babarem" (como toda avó), seja de longe ou por perto. Junto a elas, agradeço à toda minha família (“Grande Família” e “Família Fragoso”), pelas palavras de incentivo, pelos encontros e festas que alimentam nosso amor e bem-querer. Em especial, à minha vó Lena e minha tia Bel, pela convivência diária, incentivo, apoio sempre que foi preciso e, não podendo deixar de citar, todos os mimos dedicados a mim e ao meu irmão.

Ao meu noivo, Demontiê Junior, que completa, junto com minha família, a minha fortaleza, meu porto seguro e minha base. A quem eu escolhi para estar comigo na alegria e na

tristeza, na saúde e na doença, compartilhando todos os momentos de nossas vidas. A ele agradeço por todos os sentimentos despertados em mim. Agradeço pelos planos e sonhos arquitetados e/ou concretizados juntos. Agradeço por todo apoio, orientação e ajuda, não apenas neste trabalho. Agradeço o amor, carinho, atenção, companheirismo, amizade e a paciência. Agradeço por ser (per)feito para mim.

Ao meu orientador, Dalton, pelas reuniões e conversas que clarificam a mente e estimulam a busca pelo aprendizado e desenvolvimento do trabalho. Assim como, também, por todos os momentos de compreensão, apoio e companheirismo.

Ao meu coorientador, Marco Tulio, pela proposta da ideia deste trabalho, pelo acolhimento com a coorientação, pela atenção dedicada a mim, não apenas nos meses em que estive presente em Belo Horizonte, como também, em todos os momentos durante este estudo (apesar da distância entre BH, Campina Grande e Maceió).

Ao professor Franklin Ramalho, pelo companheirismo, pela preocupação demonstrada e pelos momentos de descontração.

A Rosa e Ieda por cuidarem de mim e da minha casa, enquanto estudava (em Maceió e Campina Grande, respectivamente), e por escutarem meus desabafos, preocupações e alegrias.

À família do meu noivo, por me acolher e me adotar, instantaneamente, me fazendo sentir em casa. Em especial, aos meus sogros, Clécia e Demontiê, por terem me acolhido quando mais precisei, me passando segurança (junto com minha família e meu noivo) para continuar em Campina Grande e terminar meu curso.

A tia Rosário, Anibal, Daniel, Gabriel, Alba, Rodrigo, Pedro, Miriam, Alcionio, Angelo e Adriano pela escolha mútua de formar uma família de amigos. A eles, agradeço por serem minha base em Campina Grande, por toda a amizade verdadeira e duradoura, por compartilharmos nossas felicidades, tristezas, estresses, raivas, conquistas, enfim, os momentos vividos por nós desde a minha infância até os dias atuais.

Aos meus amigos, Marcela, Karol, Savyo, Delano, Carlúcia, Natã, Raissa, Davi, João, Laerte, Raquel, Arthur, Erickson, Luiz, Janderson e Kláudio, por estarem sempre presentes, mesmo que ausentes, nessa caminhada acadêmica, torcendo por mim e pelo meu sucesso. Pelos encontros, risadas e descontrações para desopilar do estresse do mestrado. Por todo apoio, pela paciência e compreensão em meus momentos de abuso. E, não poderia deixar de

agradecer as inúmeras caronas que me deram!!

Aos participantes e respondentes do estudo que compõe essa dissertação. Sem eles, nada disso seria possível.

À COPIN, seus funcionários e Rebeka Lemos (ex-funcionária) por desempenharem seus papéis de forma responsável, assim como, também, por sempre me atenderem com atenção e respeito.

E, por fim, a todos os cidadãos brasileiros que pagam seus impostos e, a partir deles, financiaram toda a minha formação acadêmica, desde a graduação até este mestrado.

Conteúdo

Lista de Símbolos

ADL - *Architecture Description Language*

API - *Application Programming Interface*

CMMI - *Capability Maturity Model - Integration*

GT - *Grounded Theory*

IDE - *Integrated Development Environment*

IEEE - *Instituto de Engenheiros Eletricistas e Eletrônicos*

LDM - *Lattix Inc's dependency manager*

SAVE - *Software Architecture Visualization and Evaluation*

SEI - *Software Engineering Institute*

TI - *Tecnologia da Informação*

UFCG - *Universidade Federal de Campina Grande*

UFMG - *Universidade Federal de Minas Gerais*

UML - *Unified Modeling Language*

URL - *Uniform Resource Locator*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

A Engenharia de *Software* é a área da Computação que estuda o desenvolvimento, a manutenção e a gestão de *software* de alta qualidade [42]. O constante avanço tecnológico do mundo atual faz com que esta área esteja sempre em atualização, trazendo novos conceitos, ferramentas e linguagens para o ambiente de desenvolvimento.

Segundo Parnas [29], todo sistema "envelhece"com o tempo. Os sistemas computacionais que não acompanham o surgimento de novas tecnologias são considerados ultrapassados. Para evitar esse envelhecimento e acompanhar o desenvolvimento tecnológico, é preciso realizar mudanças contínuas no *software*. Mudanças no sistema podem trazer problemas se não forem bem executadas. Segundo Parnas [29], normalmente as mudanças são realizadas por pessoas que não entendem os conceitos envolvidos no sistema quando ele foi construído [29], acarretando, assim, na inserção de defeitos ou em mudanças inapropriadas. Unindo-se a este fato, é bastante comum que a documentação de um sistema (quando existente) esteja desatualizada, incompleta e/ou inconsistente [34], dificultando, assim, a manutenção e evolução do *software*.

Sem uma documentação (ou com documentação precária) e com profissionais que não entendem os conceitos do sistema [29; 34], a probabilidade de haver a inserção de modificações incorretas no sistema pode aumentar. Isso pode ocorrer pois todo o conhecimento do sistema está, praticamente, no código fonte e na cabeça dos membros da equipe de desenvolvimento. As mudanças inseridas no *software* de forma incorreta causam modificação/de-

terioração em sua estrutura e introdução de *bugs*, dificultando o entendimento e manutenção do sistema. Isso gera um acúmulo de problemas com o *software*, possibilitando a inserção de mais modificações incorretas.

É difícil trabalhar em sistemas com muitos erros e com grande acoplamento. Os módulos podem não ser entendidos facilmente e as alterações e ampliações de uma funcionalidade podem não ser controladas [37]. Por isso, é necessário desempenhar atividades que monitorem e controlem essa evolução do sistema para que o produto final esteja de acordo com suas especificações. Uma das atividades a se realizar nesse sentido é a concepção e documentação da Arquitetura do *Software*, que é uma atividade de registro do conhecimento sobre o sistema em desenvolvimento.

Existem diversas definições na literatura para o termo "Arquitetura de *Software*" [26]. Quando Clements *et al.* lançaram seu livro "Documenting Software Architectures: views and beyond" [8], em 2002, havia, no *site* do SEI *(Software Engineering Institute)*, mais de 90 (noventa) definições coletadas para o termo em questão. Para Clements *et al.* [9; 8], Arquitetura de *Software* é *"a cola conceitual que mantém todas as fases de um projeto em conjunto, para todos os tipos de cargos da equipe"*. Já para Perry & Wolf [30], *"é um conjunto de elementos arquiteturais (ou de design) que possuem um formato particular. Os elementos arquiteturais podem ser de processamento (elementos que fornecem a transformação para elementos de dados), de dados (elementos que contém as informações) ou de ligação (cola que mantém as diferentes peças da arquitetura unidas)"*. Continuando nas variações de definições para Arquitetura de *Software*, Garlan & Perry [17] definem como *"a estrutura dos componentes de um programa, seus relacionamentos e os princípios e diretrizes que regem sua criação e evolução ao longo do tempo"*. Já Bass *et al.* [1] definem como *"a(s) estrutura(s) do sistema que compreende(m) os componentes do software, suas propriedades exteriormente visíveis e seus relacionamentos entre si"*.

A definição que utilizamos em nossas pesquisas é a de Jansen & Bosch [19], em que arquitetura é um conjunto de decisões/regras arquiteturais que estabelecem relações entre os componentes de uma aplicação. Para exemplificar uma regra arquitetural, considere um sistema em que o arquiteto utilizou camadas para separar a apresentação, da lógica de negócios e os objetos de acesso aos dados. Dizer que "a camada de apresentação não pode depender diretamente dos objetos de acesso aos dados" é criar uma regra arquitetural.

Segundo Knodel *et al.*, a arquitetura de *software* é um dos artefatos mais importantes no ciclo de vida de um sistema [22; 23]. Ela interfere nos objetivos de negócios, objetivos funcionais e na qualidade do sistema. Por esse motivo, é muito importante desempenhar esforço em sua concepção, documentação e verificação de conformidade.

Assim como é difícil manter a arquitetura documentada e atualizada, também é um desafio a ser enfrentado (durante a evolução e manutenção do *software*) manter a conformidade entre documentação e código fonte. Verificação de Conformidade Arquitetural é a atividade através da qual se verifica se a implementação do sistema está de acordo com arquitetura planejada pelos desenvolvedores/arquitetos [9]. Garantir a conformidade arquitetural de um sistema é importante para permitir reuso, compreensão do sistema, possuir documentação arquitetural atualizada, ter controle da evolução do sistema e permitir a discussão entre os membros da equipe sobre a estrutura do sistema [22].

Quando não existe conformidade arquitetural, ou seja, quando algum trecho de código do sistema não obedece a uma regra arquitetural, dizemos que há violações arquiteturais no código. Com isso, a arquitetura implementada se torna incompatível/inconsistente com a arquitetura planejada. São fatores que geram violações arquiteturais: a falta de documentação arquitetural (ou a não atualização dela), requisitos conflitantes resolvidos durante a implementação, alterações tardias durante a implementação e restrições técnicas de desenvolvimento [23].

Com a evolução do sistema, as violações arquiteturais tendem a não ser corrigidas devido a diversos fatores, como, por exemplo, falta de tempo e pressão para entrega do sistema. Assim, é gerado um acúmulo de violações arquiteturais (que quase nunca são corrigidas) e, no decorrer do tempo, essa inconsistência arquitetural só tende a aumentar, tornando-se um ciclo recursivo e acumulativo. Ao acúmulo de violações arquiteturais no código fonte chamamos de Erosão Arquitetural [30], um problema considerado grave e que acarreta em deterioração de toda a estrutura do sistema. É para evitar esse acúmulo de violações no sistema e garantir a adequação do código fonte (ou da implementação) à arquitetura planejada que existe a Verificação de Conformidade Arquitetural [23].

Existem várias formas para documentar uma arquitetura e realizar a Verificação de Conformidade Arquitetural. Uma delas, é realizar essas atividades de forma manual, utilizando documentos, linguagem natural e verificando o código manualmente. Porém, em sistemas

grandes e complexos, concordamos com Postma [34], que considera quase impossível que um ser humano consiga entender todos os detalhes de uma implementação para realizar, manualmente, uma Verificação de Conformidade Arquitetural de forma completa e rigorosa.

Diversas linguagens (ADLs) e ferramentas foram desenvolvidas pelos pesquisadores com o objetivo de auxiliar as atividades de documentação e verificação de conformidade arquitetural. Por exemplo, Eichberg *et al.* [16] desenvolveram a linguagem lógica de domínio específico chamada LogEn (*Logical Ensembles*) e Terra *et al.* [47] desenvolveram a linguagem chamada DCL (*Dependency Constraint Language*). Murphy *et al.* [28] criaram o conceito de modelo de reflexão, que mostram os pontos onde os modelos planejado e implementado concordam e discordam. Para gerar o modelo implementado, nessa abordagem, é necessária uma ferramenta, como, por exemplo, SAVE [23; 24].

Terra *et al.* [47] também desenvolveram uma ferramenta, chamada *dclcheck*, que é um *plugin* para o Eclipse que realiza a verificação arquitetural utilizando a linguagem criada por eles. Brunet *et al.* [5] desenvolveram uma API, chamada DesignWizard, que permite escrever testes de *design* para implementações em Java, usando JUnit. Teste de *design* [5] é um teste que verifica se uma implementação obedece a uma dada regra de projeto expresso como um algoritmo.

Desde os anos 90 a academia desempenha esforços para estudar e evoluir aspectos relacionados à Arquitetura de *Software*, aumentando seu volume de publicação a partir de 1999 (apesar de haver estudos datados de 1992) [4]. O aumento significativo das publicações nos últimos anos [4] demonstra a importância e a preocupação que a academia tem com relação a essa área. Estudos sobre linguagens para documentação arquitetural [47; 5; 16; 25], técnicas para visualização de arquitetura de *software* [39], abordagens para reconstrução de arquitetura de *software* [15] e até sobre o ensino da área nos cursos de graduação [35] estão sendo desenvolvidas constantemente no meio acadêmico.

Visto isso, é notável o esforço que a academia vem desempenhando na área de Arquitetura de *Software* visando melhorar o desenvolvimento de sistemas computacionais. Porém, a partir da troca de experiência entre pesquisadores e profissionais da área e da leitura de alguns artigos citados no Capítulo 2, percebemos que a indústria (quem produz os sistemas computacionais utilizados pela sociedade) não parece conhecer/utilizar o que é proposto pela

academia. Por exemplo, o artigo "UML in practice" [31] corrobora nossa percepção, uma vez que uma linguagem bastante difundida no meio acadêmico mostrou-se pouco utilizada no meio industrial.

Provavelmente, um dos motivos para esse fato pode ser a falta de comunicação e relacionamento entre o trabalho desempenhado na academia e o trabalho desempenhado na indústria. Além disso, pode haver uma diferença na compreensão, definição e utilização de arquitetura de *software* e suas ferramentas (qualquer que seja sua finalidade) entre a academia e a indústria.

Por fim, consideramos importante que a academia e a indústria caminhem juntas com o objetivo de evoluir a área de Arquitetura de *Software* [42]. Da mesma forma que não adianta a academia propor abordagens, linguagens e ferramentas sem observar as necessidades e o ambiente industrial (o que acarretaria em pesquisas pouco relevantes na prática), também é pouco eficiente a indústria desenvolver sistemas sem pesquisar e evoluir seus processos, uma vez que o mundo tecnológico está em constante mudança.

1.2 O estudo

Neste trabalho, apresentamos um estudo sobre a percepção dos conceitos de arquitetura de *software* por profissionais da indústria. Entender como os profissionais da área tratam este assunto pode contribuir com o fornecimento de diretrizes para os estudos que estão sendo desenvolvidos pelos pesquisadores.

Nesse sentido, diversas questões podem ser feitas como, por exemplos: A indústria conhece as diversas ferramentas e linguagens que são propostas pela academia? A indústria as utiliza? Elas são úteis na prática? Como os profissionais da indústria pensam e agem sobre a Arquitetura de *Software*?

As atividades de arquitetura não nos parecem ser priorizadas na indústria. Notamos que em algumas empresas do Brasil é difícil encontrar o cargo de arquiteto de *software*. Isto nos permite inferir que a construção, a documentação e a verificação, nessas empresas, depende da própria empresa e dos processos de desenvolvimento que elas utilizam.

Acreditamos, também, a partir da troca de experiência com outros profissionais, que as linguagens e ferramentas acadêmicas não parecem ser muito difundidas e utilizadas no meio

industrial. Além disso, suspeitamos que a utilidade de cada linguagem/ferramenta depende do objetivo do profissional em seu processo de desenvolvimento.

As afirmações anteriores, contudo, apesar de intuitivas, não são embasadas por resultados de estudos e dados concretos. O estudo que apresentamos nesta dissertação procura esclarecer as questões e os sentimentos de dúvidas que cercam o assunto. Para isso, o estudo foi formalizado e organizado para responder as quatro seguintes questões de pesquisa:

RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

RQ2: Qual a visão da indústria com relação à documentação arquitetural?

RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

As questões de pesquisa foram divididas em quatro tópicos: definição de arquitetura de *software*, documentação arquitetural, verificação de conformidade arquitetural e ferramentas de apoio à verificação de conformidade arquitetural. Com esses 4 tópicos pretendemos abranger o máximo de conhecimento envolvido na área com relação a como o tema é tratado na prática de desenvolvimento de sistemas (desde a definição até a utilização de ferramentas).

1.2.1 Metodologia

Nossa metodologia foi baseada no artigo de Singer *et al.* [41] e dividimos nosso estudo em três etapas: *survey* exploratório, entrevistas e *survey* confirmatório. Com essas três etapas temos um estudo quali-quantitativo (ou misto), ou seja, uma combinação de métodos qualitativos e quantitativos. Este método misto aumenta a confiabilidade nos resultados obtidos, uma vez que capturamos as informações de formas diferentes (Triangulação Metodológica dos Dados [10]). Porém, apesar de ser misto, temos como foco principal de nossa pesquisa a etapa de entrevistas (etapa qualitativa), pois, a partir dela foram descobertas as principais conclusões. Os *surveys* exploratório e confirmatório são etapas secundárias uma vez que tiveram como objetivo explorar o ambiente para realizar as entrevistas e confirmar as conclusões das entrevistas, respectivamente. Por este motivo, decidimos simplificar o título desta dissertação e focar na etapa qualitativa.

Na primeira etapa realizamos um *survey* exploratório com o objetivo de explorar o tema, em alto nível, buscando uma noção básica sobre as possíveis respostas para as questões de

pesquisa. Nesta etapa, enviamos o questionário para 149 profissionais que exercem o papel de arquitetos de *software* ou *core developers* de seus projetos (dos quais 47 foram indicados pelos próprios respondentes do *survey*). Obtivemos uma taxa de resposta de 24,1% (36 profissionais respondentes).

Na segunda etapa realizamos entrevistas, com perguntas geradas a partir da análise da etapa anterior, com o objetivo de aprofundar o conhecimento e entendimento sobre as principais questões de pesquisas. Foram feitas, em média, 13 perguntas. Alguns respondentes da etapa anterior se voluntariaram, a partir de nossa chamada, para participar desta etapa. Dos 36 profissionais que responderam ao *survey* explorátorio, 14 se voluntariaram e participaram desta segunda etapa, e, portanto, obtivemos uma taxa de resposta de, aproximadamente, 40%.

Na terceira etapa realizamos um *survey* confirmatório com o objetivo de confirmar as conclusões de nosso estudo e trazer mais confiança para elas. Para tanto, enviamos nosso questionário para 4352 profissionais do GitHub que possuíam mais de 100 seguidores. Obtivemos 337 respostas, o que nos dá uma taxa de resposta de 7,74%.

1.2.2 Resultados

Assim como na academia existem diversas definições para o termo "Arquitetura de *Software*", na indústria também não existe um consenso para a definição do termo. Consideramos este achado natural, uma vez que havendo diversas definições acadêmicas, a definição prática pode depender de diversos fatores como o conhecimento de cada profissional, de sua origem acadêmica e como aprenderam a teoria.

Com nosso estudo, constatamos que a forma de documentação arquitetural varia muito entre empresas e profissionais. Na maioria dos projetos, a documentação arquitetural é informal, incompleta e/ou desatualizada. Normalmente, documentos escritos em linguagem natural são utilizados como forma de documentar a arquitetura. Sabendo que, na academia, diariamente são pesquisadas e propostas novas linguagens formais e maneiras de organizar uma documentação arquitetural, nos questionamos o motivo pelo qual a indústria prefere documentos em linguagem natural ao invés das propostas acadêmicas.

Os dados também nos permitem afirmar que a atividade de verificação de conformidade arquitetural, quando realizada, é feita, na maioria das vezes, de forma manual. As ferra-

mentas de apoio à verificação de conformidade arquitetural são pouco conhecidas e pouco utilizadas na indústria. Isto reflete nas pesquisas realizadas no meio acadêmico pois, mostra aos pesquisadores que a indústria confia no método manual e, portanto, isso deveria ser levado em consideração ao desenvolver as ferramentas de apoio.

Por fim, nossos dados nos permitem constatar que a falta de um profissional responsável pela arquitetura de *software*, sua documentação e verificação é um dos fatores da falta de valorização para com essas atividades, uma vez que, na equipe, não há quem se sinta na obrigação de realizá-las.

1.3 Contribuições

Nossa principal contribuição é um estudo empírico para compreender melhor algumas reflexões e percepções sobre Arquitetura de *Software* dentro da indústria. Com nosso estudo, temos, brevemente, a percepção de alguns profissionais da indústria com relação à definição de arquitetura, documentação arquitetural, verificação de conformidade arquitetural e às ferramentas de apoio a essa atividade. Compreendendo melhor o contexto prático sobre arquitetura de *software*, a academia poderia voltar suas pesquisas para questões que sejam mais utilizadas nos processos de desenvolvimento de sistemas reais.

Além disso, temos uma reflexão sobre a relação entre indústria e academia. Por exemplo, qual o papel de cada uma na evolução e melhoria dos processos de desenvolvimento de sistemas reais? A indústria deveria prestar mais atenção no que é produzido na academia? Como a academia pode introduzir suas produções científicas na indústria?

Por fim, com nosso estudo, algumas questões de pesquisa ficam em aberto para serem pesquisadas como trabalhos futuros como, por exemplo: Por que muitos profissionais não conhecem o termo verificação de conformidade arquitetural? Quais são os diversos termos utilizados nessa área? Existem sinônimos que podem facilitar a comunicação entre os profissionais? Há alguma relação entre conhecimento teórico da equipe sobre arquitetura e a realização de atividades voltadas para a arquitetura do sistema?

Tivemos, ainda, 2 artigos publicados: "Verificação de Conformidade Arquitetural com Testes de Design - Um Estudo de Caso", publicado no VEM (I Workshop Brasileiro de

Visualização, Evolução e Manutenção de Software), em 2013 [1], e "Uma Ferramenta para Verificação de Conformidade Visando Diferentes Percepções de Arquiteturas de Software", publicado na Sessão de Ferramentas, em 2014 [2] (ambos realizados no CBSoft - Congresso Brasileiro de Software: Teoria e Prática). Esses artigos são relacionados à pesquisas anteriores e não possuem relação com este estudo que desenvolvemos atualmente.

1.4 Estrutura da dissertação

No Capítulo 2, fundamentamos sobre Pesquisa Qualitativa, Triangulação de Dados, Método Bola de Neve, Teoria Fundamentada (Grounded Theory) e Análise Temática. Além disso, apresentamos uma revisão bibliográfica sobre estudos qualitativos realizados na Computação, bem como ferramentas de apoio à verificação de conformidade arquitetural desenvolvidas na academia.

Nos Capítulos 3, 4, 5 e 6, apresentamos o estudo e suas 3 etapas: *Survey* Exploratório, Entrevistas e *Survey* Confirmatório. Nesses capítulos, apresentamos o planejamento, execução, resultados, análises e ameaças à validade de cada etapa. No Capítulo 7, apresentamos as Considerações Finais sobre nosso estudo, discutindo os resultados, contribuições dessa dissertação, ameaças à validade de forma geral e trabalhos futuros.

Por fim, temos 4 Apêndices: roteiro do *Survey* Exploratório, roteiro das Entrevistas, códigos da análise das entrevistas e roteiro do *Survey* Confirmatório.

[1] http://wiki.dcc.ufba.br/pub/Vem2013/ProgramacaoPt/118775_artigo.pdf. Último acesso em 22/07/2015.

[2] http://www.ic.ufal.br/evento/cbsoft2014/anais/ferramentas_v1_p.pdf. Último acesso em 22/07/2015.

Capítulo 2

Fundamentação Teórica

2.1 Pesquisa Qualitativa

Desde muito tempo, as descobertas científicas tem sido o objetivo central da ciência, porém, a forma como elas são concebidas têm variado de acordo com a natureza dos materiais estudados [44]. Nem sempre as pesquisas realizadas na academia possuem um teor estatístico, quantitativo. Às vezes, para estudar, entender, analisar comportamentos, pessoas, atividades, sentimentos, opiniões, rotinas, etc. é necessário utilizar mais que números e estatística. É necessário colher os dados e interpretá-los, dando uma conotação qualitativa para a pesquisa [40].

Problemas envolvendo opiniões, descrições de pessoas ou locais, conversações, métodos de trabalho, observações do pesquisador sobre o ambiente/cenário, etc. necessitam de métodos de pesquisa diferentes dos comumente utilizados em nossa área. Normalmente, são utilizados métodos quantitativos para pesquisar na computação, realizando experimentos, desenvolvendo ferramentas, analisando código e dados [36]. Mais especificamente, na Engenharia de Software, realizar pesquisas qualitativas é algo relativamente novo (final da década de 90) [38].

> Como colocado por Kaplan e Maxwell (1994), complementando que "o objetivo do entendimento de um fenômeno sob o ponto de vista dos participantes e seu contexto particular social e institucional são, em muito, perdidos quando dados textuais são quantificados" [40].
>
> O primeiro problema de publicar os resultados de uma pesquisa qualitativa é defender o mérito dos métodos qualitativos, se a publicação é em uma área da CC que valoriza a pesquisa quantitativa [49].

Porém, pesquisas demonstram que torna-se, cada vez mais necessária a aproximação da academia com a indústria para desenvolver pesquisas e sistemas que sejam mais utilizáveis e eficientes [42]. Para tanto, é importante que a academia entenda os comportamentos, métodos utilizados, ambientes de trabalho, pensamentos de profissionais, etc. do ramo industrial. São nesses tipos de pesquisa que o método qualitativo se torna cada vez mais essencial e importante.

Visto isso, podemos afirmar que a pesquisa qualitativa é todo tipo de pesquisa que produz resultados sem necessariamente utilizar procedimentos estatísticos ou outros meios de quantificação dos dados. Alguns dados podem ser quantificados, como números relacionados aos respondentes que possam clarificar a amostra utilizada, porém, a maioria da análise é resultado da descrição e da interpretação do pesquisador sobre o que vê, escuta e/ou entende. Os dados podem ser entrevistas, observações, assim como documentos, filmes ou vídeos [44; 12].

> Faz parte dela a obtenção de dados descritivos mediante contato direto e interativo com a situação de estudo pelo pesquisador que, frequentemente, procura entender os fenômenos segundo a perspectiva dos agentes atuantes e, a partir deste entendimento, dar sua interpretação (Neves, 1996) (...) Kaplan e Duchon (1988) colocam, como as principais características dos métodos qualitativos, "a imersão do pesquisador no contexto e a perspectiva interpretativa de condução da pesquisa" [40].

Uma pesquisa qualitativa deve ser conduzida com ética, de forma rigorosa e sistemática, utilizando uma estratégia, porém sem perder a flexibilidade. A pesquisa deve ser realizada, idealmente, face a face entre pesquisador e participantes [12]. O pesquisador deve ser autocrítico para que não interfira nos dados e resultados, além de explicar todas as conclusões a que chegou [40]. Em todo o processo, o pesquisador foca em aprender o que os participantes entendem ou acham da questão em análise [12].

Uma pesquisa qualitativa possui 3 (três) componentes: os dados, que podem ser coletados a partir de várias fontes como entrevistas, observações, documentos, vídeos, etc.; os procedimentos utilizados para analisar e interpretar os dados (normalmente é utilizada a codificação); e, por fim, os relatórios escritos e verbais produzidos a partir da análise dos dados [12; 44].

A coleta de dados inclui a escolha dos participantes que farão o pesquisador entender melhor o problema em estudo, a indicação do(s) tipo(s) de dado(s) que será(ão) coletado(s), e as justificativas para cada decisão tomada. Por sua vez, a análise dos dados e a interpre-

tação incluem a organização e a preparação dos dados: observá-los, inicialmente, para ter uma noção geral; codificação; descrição dos participantes do estudo; e, por conseguinte, a interpretação dos dados explicando quais lições foram aprendidas com o estudo [12].

É importante que o pesquisador se preocupe com a validade de seu estudo qualitativo, ou seja, realize uma checagem sobre a acurácia dos dados encontrados. Buscando ter uma boa validade do estudo, algumas medidas podem ser tomadas como, por exemplo, realizar a triangulação dos dados, ou seja, capturar os mesmos dados de diferentes formas para que as análises sejam cruzadas e validadas. Outra medida que pode ser tomada é mostrar os resultados para os participantes para que eles digam se acham corretos ou não. Utilizar uma descrição rica para expor os resultados e esclarecer os vieses envolvidos também é uma medida para buscar a validade do estudo. Apresentar toda a pesquisa para um auditor externo avaliar também é uma medida para se buscar validade [12].

Além da validade do estudo, o pesquisador também deve se preocupar com a confiança qualitativa. Para isso, podem ser documentados todos os procedimentos e suas etapas, fazer a checagem das transcrições realizadas, certificar-se de que não há mudança de significado dos códigos no processo de codificação (para tanto, pode-se escrever notas explicativas sobre cada código), e/ou realizar uma verificação cruzada de códigos escritos por diferentes pesquisadores [12].

Ainda, o pesquisador deve se manter imparcial, entretanto, deve ter uma atitude sensível para encontrar as conclusões nos dados. Nesse sentido, é importante utilizar procedimentos para tentar manter a distância entre a análise e os dados: coletar respostas de vários entrevistados; coletar as respostas de diferentes formas; perguntar às pessoas se suas conclusões e interpretações estão corretas [44].

É relevante destacar que uma forma de encontrar entrevistados/respondentes para o estudo é utilizar o método Bola de Neve (os pesquisadores solicitam dos respondentes indicações para novos e possíveis respondentes) [20].

Por ser um tipo de pesquisa ainda não muito utilizada em nossa área, é necessário realizá-la de forma bastante rigorosa e cuidadosa. Para tanto, é preciso definir bem o objeto de estudo, descrevendo o cenário e o que se pretende estudar, analisar, interpretar ou verificar. É importante que o pesquisador tente manter uma posição objetiva para não contaminar o cenário com seus valores, opiniões ou preconceitos. O pesquisador deve se preocupar em

controlar a qualidade e o processo de obtenção dos dados. Além disso, ele deve buscar uma argumentação lógica, bem estruturada e sem contradições [40].

É importante que o pesquisador se preocupe com a forma de relatar o estudo, as análises e as conclusões. O documento deve ser bem descritivo, buscando uma narrativa cronológica, todos os detalhes, as experiências, as características dos participantes, as conclusões, os procedimentos e as análises. É interessante utilizar aspas para apresentar citações com as palavras dos participantes, entrelaçando as interpretações com elas. Apresentar tabelas e gráficos com os resultados podem clarificar as análises e conclusões concebidas pelo pesquisador. Também é interessante apresentar comparações entre a teoria e o estudo realizado [12].

2.1.1 Análise Temática e Teoria Fundamentada

Existem vários tipos de metodologias qualitativas: Etnografia, Fenomenologia, Teoria Fundamentada, Estudo de Caso, Pesquisa-ação [40]. Nossa pesquisa se constitui numa Análise Temática para analisar e interpretar os resultados (apesar de o artigo [41] em que nos baseamos utilizar a Teoria Fundamentada). Análise Temática é um método analítico flexível que tem o objetivo de identificar, analisar e relatar padrões nos dados qualitativos [2]. Apesar de ter sido proposta na década de 70, só vem alcançando seu reconhecimento no mundo acadêmico recentemente [3].

Este método é adequado para diferentes interesses e contextos de pesquisas pois pode ser utilizado para trabalhar com várias questões de pesquisas sobre experiências e/ou entendimentos das pessoas sobre fenômenos, pode analisar diferentes tipos de dados, pode trabalhar com conjuntos de dados grandes ou pequenos e, por fim, o método pode ser aplicado para produzir análises orientadas aos dados ou por teorias [3].

Análise temática tem 6 (seis) fases e é um processo recursivo, sendo permitido passar para a próxima fase sem ter concluído completamente e corretamente a fase anterior [3]. São elas [2]:

- Familiarização dos dados (Fase 1): esta é uma fase comum a todas as formas de análise qualitativa. O pesquisador deve mergulhar em seus dados, ler e reler, ver e rever, ouvir e re-ouvir todos os tipos de dados envolvidos na pesquisa. Nesta etapa tem-se as

observações analíticas iniciais.

- Codificação (Fase 2): também é uma fase comum em muitas abordagens de análise qualitativa. Nesta etapa, os dados são etiquetados com códigos expressivos e relevantes para as questões de pesquisa. Um código nomeia algum fenômeno observado pelo pesquisador que tem algum significado para sua pesquisa [11]. É um processo analítico que permite uma leitura semântica dos dados.

- Procurando por temas (Fase 3): um tema é um padrão coerente e significativo nos dados, que seja relevante para a questão de pesquisa. Procurar por temas é como codificar seus códigos para identificar similaridade entre os dados. É uma atividade ativa pois os temas não estão embutidos nos dados esperando serem descobertos, mas sim, são construídos pelo pesquisador.

- Revisão dos temas (Fase 4): nesta fase o pesquisador deve revisar os temas e tentar contar uma história convincente e completa sobre os dados. Deve começar a encontrar e definir as relações entre os temas. Pode ser necessário dividir um tema em dois ou o contrário (unir dois temas em um). Ou, ainda, pode ser preciso descartar todos os temas e recomeçar a etapa anterior.

- Definir e nomear temas (Fase 5): o pesquisador deve conduzir e escrever uma análise detalhada de cada tema mostrando o que o tema fala sobre a questão de pesquisa. O pesquisador deve identificar a essência de cada tema e dar um nome conciso e informativo para cada um.

- Escrita de relatório (Fase 6): nesta fase o pesquisador deve escrever sua análise. Deve escrever uma narrativa analítica e mostrar uma história coerente extraída dos dados.

A Análise Temática tem como vantagens a flexibilidade; é um método relativamente fácil e rápido para aprender e aplicar; acessível a pesquisadores com pouca experiência em pesquisas qualitativas; e pode gerar percepções imprevistas [3].

Já a Teoria Fundamentada (ou Grounded Theory) é uma metodologia indutiva e foi utilizada pelo artigo no qual baseamos nossos estudos [41]. Ela foi proposta, inicialmente, por Glaser e Strauss, em 1967, no livro *"The discovery of Grounded Theory: strategies for*

qualitative research" [18; 32]. Porém, no decorrer dos anos, diferentes versões e vertentes foram produzidas ao redor dela [7].

Independente disso, a Teoria Fundamentada tem o objetivo de conceber teorias sobre o assunto estudado a partir das análises dos dados coletados. Teoria é um conjunto integrado de proposições que explicam a variação da ocorrência de um fenômeno [11; 44]. O pesquisador que utiliza a metodologia não possui, inicialmente, uma teoria a ser testada pela análise dos dados. Pelo contrário, ele possui dados coletados, analisa-os para, então, a partir de suas interpretações, derivar suas teorias sobre seu objeto de estudo [32]. Este processo de criação das teorias se dá a partir de uma contínua interação entre a análise e a coleta de dados [45; 32]. Na etapa de análise dos dados, o pesquisador deve tentar entender como e por que determinado fenômeno ou situação acontece em seu contexto estudado [18; 32].

O mais importante nesta metodologia é que a coleta e análise dos dados devem ser realizadas continuamente até ocorrer a saturação dos dados. Ocorre saturação dos dados quando o pesquisador não consegue encontrar novos códigos relevantes em sua análise ou quando eles começam a se repetir [32].

Com isso, a teoria é o final de um estudo. São as conclusões de uma pesquisa realizada pelo pesquisador, ao invés de ser o início (hipótese a ser testada), como normalmente é realizado em outras metodologias de pesquisa. A teoria é, então, descoberta, desenvolvida e verificada por meio de uma sistemática coleta e análise de dados [32; 11]. As teorias fundamentadas, por derivarem dos dados coletados, oferecem uma visão do ambiente observado/estudado, aumenta a compreensão do objeto de estudo e fornece um guia significativo para ação no meio [44].

Uma das limitações da Teoria Fundamentada é que não existem critérios bem definidos e rígidos para garantir uma saturação dos dados. A decisão de considerar os dados saturados ou não depende do pesquisador responsável pelo estudo [32].

Apesar da finalidade do método Teoria Fundamentada ser a criação de teorias, sua utilização não precisa ser restrita apenas a quem tem esse interesse/objetivo. Pesquisadores podem utilizar apenas alguns procedimentos da Teoria Fundamentada para atingir seus objetivos, ou seja, cada pesquisador pode adaptar o método de acordo com seu interesse [11; 44].

Escolhemos, para o nosso estudo, a Análise Temática pois ela é mais flexível, oferecendo

uma forma mais acessível de análise, especialmente para aqueles que estão no início de uma carreira de investigação qualitativa [2]. Além disso, mesmo que seja possível adequar a metodologia da Teoria Fundamentada, ainda assim, ela é voltada e guiada para desenvolver teorias (que não é nosso foco).

2.2 Estudos qualitativos em Engenharia de *Software*

Os estudos qualitativos na Computação vem crescendo e tomando seu espaço na academia. O primeiro abordado nesta sessão é o artigo escrito por Singer *et al.*, *"Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter"* [41], no qual baseamos a modelagem do nosso estudo. O Twitter é o serviço de microblogging mais popular do mundo, possuindo mais de 500 milhões de usuários postando mais de 500 milhões de tweets por dia. Nele, os usuários podem escrever uma publicação com até 140 caracteres.

É sabido que a engenharia de *software* muda rapidamente. São linguagens, tecnologias, dispositivos e ferramentas que surgem constantemente. Os desenvolvedores precisam se atualizar sobre essas mudanças. Com este objetivo, várias mídias são utilizadas: *e-mails*, IRC, *blogs*. Porém, algumas mídias ainda não são bem compreendidas, como é o caso do Twitter. Segundo os autores, as pesquisas apontam que desenvolvedores utilizam o Twitter em seus trabalhos, porém, ninguém havia pesquisado ainda sobre como e porque se dá essa utilização. Também não havia sido pesquisado sobre porque alguns não utilizam este microblogging no trabalho.

Ainda segundo os autores, aprender como e porque os desenvolvedores utilizam (ou não) o Twitter pode ajudar a compreender os desafios e necessidades dos desenvolvedores em seu processo de aprendizagem, comunicação e colaboração. Para tanto, foi necessário realizar uma pesquisa empírica.

O estudo foi baseado na metodologia de Teoria Fundamentada (*Grounded Theory*) e foi dividido em 3 etapas: *survey* exploratório, entrevista e *survey* de validação. Os *surveys* foram enviados para usuários do GitHub que possuíam *e-mail* público. O *survey* explorátorio (enviado para 1.160 usuários) teve 271 respondentes e tinha como objetivo adquirir uma visão geral sobre as razões para ler e postar no Twitter, assim como os benefícios e desafios dos usuários. Também foi capturada uma visão geral sobre como os usuários do Twitter

fazem para descobrir e seguir outros usuários, além de questionar o motivo para a não adoção por parte de alguns profissionais.

Como resultado do *survey* exploratório, os autores perceberam que o Twitter pode fornecer valor para alguns desenvolvedores de *software*. Além disso, alguns desenvolvedores enfrentam vários desafios com o uso do *microblogging*, porém, eles possuem estratégias para contornar as situações. Já outros, não podem ou não querem utilizar o Twitter.

Na segunda etapa (entrevistas), foi solicitado que os respondentes do *survey* exploratório se voluntariassem. As respostas do *survey* exploratório foram analisadas e utilizadas para construir o roteiro de entrevista com o objetivo de aprofundar o conhecimento adquirido na etapa anterior. Foram entrevistados 27 profissionais (quando atingiu o ponto de saturação da metodologia utilizada).

Durante a análise do *survey* exploratório e das entrevistas foram criados *memos* sobre temas recorrentes e conceitos emergentes nos dados. Foi utilizado codificação axial, interagindo através das respostas da pesquisa exploratória e das transcrições das entrevistas. O conjunto final de temas formou a terceira fase (*survey* validatório) que obteve 1200 respondentes (o questionário foi enviado para 10.000 usuários).

No final do estudo, os autores possuíam alguns resultados finais:

- Os desenvolvedores seguem, no Twitter, outros desenvolvedores, projetos e líderes de pensamento. Isso permite que eles fiquem cientes de tecnologias e práticas atuais e acessem diversas opiniões. Além disso, alguns utilizam o Twitter para promover suas atividades próprias, conteúdo e projetos. Isso os ajuda a disseminar o conhecimento e aumentar a adoção de tecnologias.

- Os desenvolvedores usam o Twitter para fazer e responder perguntas. Eles seguem especialistas, o que lhes permite participar de conversas que lhes proporcionam conhecimentos que, normalmente, seriam de difícil acesso. Além disso, eles vêem o aprendizado como um investimento em suas carreiras e gostam da aprendizagem acidental que o Twitter facilita.

- Alguns desenvolvedores gerenciam suas "imagens" no Twitter. Eles descobrem desenvolvedores interessantes para manter um relacionamento e comunidades são construídas em torno de um interesse em comum. O Twitter pode ajudar a construir uma

confiança no meio profissional e gerar oportunidades de trabalho.

- Os desenvolvedores tentam manter uma rede relevante seguindo desenvolvedores relevantes e deixando de seguir as pessoas que mudaram de interesse. O volume de conteúdo é alto, então eles utilizam ferramentas que permitem filtrar os *tweets*, salvam *links* que lhes interessam para ler depois e possuem uma rotina de leitura dos *tweets* (um horário determinado).

- Os motivos para não utilização do Twitter são o excesso de informação, a restrição de 140 caracteres, o fraco suporte para conversações e a falta de necessidade.

Os *insights* fornecidos pelos não adeptos e usuários pouco frequentes são valiosos para a compreensão das limitações do Twitter em Engenharia de *Software*. Um grande número de profissionais não entende como o Twitter pode ajudar em seu trabalho e existe a preocupação do Twitter ser uma distração. Por outro lado, os adeptos mostraram que o Twitter pode ter sua vantagem se for usado com estratégias para filtrar conteúdo e reduzir as distrações.

Por fim, como limitações dos estudos são apresentados que não é possível generalizar os resultados. Como foi utilizado *Grounded Theory*, as entrevistas pararam quando houve saturação. Outros *insights* poderiam surgir com outras entrevistas. Além disso, a maioria dos respondentes eram de países ocidentais e participantes da comunidade GitHub. Talvez, com outras culturas o pensamento seja diferente. E o estudo foi focado no Twitter, não sendo possível generalizar os resultados para outras plataformas de *microbloggins*.

Segundo o artigo “UML in Practice” de Marian Petre [31], UML foi, por muito tempo, descrita, por alguns, como "a língua franca da engenharia de software”. Porém, as evidências da indústria não pareciam apoiar tal afirmação. Com isso, surgiu a questão: como UML é exatamente utilizada na indústria (se for utilizada)? A maioria dos estudos até então se concentravam em compreender UML, métricas e qualidade do modelo. Porém, nenhuma delas consideraram a utilização de UML na indústria.

Este artigo é resultado de um estudo empírico realizado durante 2 anos com mais de 50 desenvolvedores de *software*. O estudo foi uma série de entrevistas com os desenvolvedores com perguntas relacionadas sobre o uso de UML em suas empresas que trabalhavam naquele momento ou anteriormente. Cada participante era de empresa diferente e, com isso, a amostra ficou ampla e representativa. As entrevistas foram analisadas a partir de categorização

das transcrições e, estas, foram revisadas por dois profissionais experientes com o objetivo de validar o estudo.

Como resultados, Marian Petre obteve que a maioria não usa UML, poucos usam apenas para satisfazer políticas de gestão ou de clientes, alguns utilizam para gerar códigos automáticos, alguns utilizam de forma pessoal e informal e nenhum deles utilizam de forma rigorosa e completa. Com isso, a autora conclui que UML não é utilizado de forma universal, e, muitas vezes, é utilizado de forma seletiva e informal.

No artigo "How Do Professionals Perceive Legacy Systems and Software Modernization?" de Khadka *et al.* [20], os autores afirmam que pesquisas sobre modernização de sistemas legados tradicionalmente focam em problemas técnicos. Além disso, tomam como princípio que sistemas legados são obsoletos, mesmo que eles sejam cruciais para operações organizacionais. Porém, não estava claro se os profissionais da indústria também tinham essa percepção (de que sistemas legados são obsoletos). Para tanto, foi realizado um estudo exploratório com o objetivo de descobrir novas perspectivas e *insights* sobre sistemas legados na indústria.

O estudo utilizou duas técnicas, misturando pesquisa qualitativa e quantitativa (triangulação de dados) para trazer mais confiança nos resultados. Foram, então, realizadas entrevistas semi-estruturadas, utilizando Teoria Fundamentada, para analisar as transcrições e um *survey* estruturado para validar as respostas das entrevistas.

Na entrevista, 26 profissionais de diferentes tamanhos de empresas, de diferentes cargos, com diferentes tempos de experiência, foram questionados, por um período entre 1 e 2 horas, sobre perguntas variadas (o que faz um sistema ser legado, quais são os principais fatores que levam à modernização desses sistemas, quais desafios são enfrentados durante esse processo de modernização). Os entrevistados foram selecionados de acordo com 2 critérios: ter experiência com sistemas legados e ter experiência com a modernização deles. A abordagem bola de neve foi utilizada para conseguir mais entrevistados.

Os resultados das entrevistas foram validados a partir de um *survey* estruturado aplicado a 198 profissionais. Os respondentes desta etapa indicaram se concordavam ou não com os resultados. Esta etapa foi divulgada em listas de discussões, redes sociais como Twitter, LinkedIn, Facebook, assim como a partir de referências pessoais. Os 198 respondentes eram de mais de 30 países diferentes. 22 das 198 respostas foram excluídas da amostra pois não

possuíam experiência com sistemas legados.

Como resultados desta pesquisa, obtiveram que os profissionais valorizam seus sistemas legados e os desafios enfrentados não são apenas técnicos, havendo também de negócios e aspectos organizacionais. A maioria dos respondentes afirmaram que um sistema legado é um sistema central da empresa e antigo (sistemas que não se encaixam em estratégias futuras da empresa), porém não são obsoletos como se afirma na academia. São sistemas críticos onde qualquer falha afeta os negócios.

A maioria dos profissionais que responderam o estudo confiam em seus sistemas legados. Porém, apesar de confiáveis, são inflexíveis para as mudanças necessárias devido às mudanças de negócios, possuem alto custo de manutenção, falta de mão de obra capacitada e a documentação não é atualizada. Por fim, os desafios enfrentados para a modernização dos sistemas legados foram pontualidade (terminar no período correto é difícil), devido a falta de recursos e mudanças dos planos iniciais; migração de dados; arquiteturas complexas; falta de conhecimento sobre o sistema; identificar a lógica de negócios; resistências à mudanças; e explicar porque é preciso modernizar para o setor de gestão da empresa.

Segundo o artigo "A Field Study of Refactoring Challenges and Benefits" de Kim *et al.* [21], acredita-se que a refatoração melhora a qualidade do *software* e a produtividade do desenvolvedor, tornando-a mais fácil de manter e entender o sistema. No entanto, poucos estudos empíricos avaliam, quantitativamente, os benefícios de refatoração ou investigam a percepção dos colaboradores em relação a esses benefícios. Com isso, foi realizado um estudo de campo (na Microsoft) sobre a definição de refatoração, os benefícios e os desafios enfrentados por uma grande organização de desenvolvimento de *software* e foi investigado se há um benefício visível de refatoração em um grande sistema.

Para tanto, foi realizado um *survey*, entrevistas semi-estruturadas com engenheiros de *software* profissionais e uma análise quantitativa do histórico de versões. O *survey* tinha o objetivo de investigar sobre a definição de refatoração na prática, assim como a percepção dos benefícios da refatoração. O questionário foi enviado para 1290 engenheiros da Microsoft que haviam realizado comentários, no repositório, com a palavra-chave *"refactor"* (e suas variações) nos últimos 2 anos em 5 produtos (Windows Phone, Exchange, Windows, OCS e Office). Dos 1290 engenheiros, 328 responderam o questionário. A partir desse *survey*, os pesquisadores tomaram conhecimento sobre o processo de refatoração que o Windows

passou. Como o Windows é um dos maiores e mais antigos sistemas da Microsoft (e uma equipe designou esforço para refatorá-lo), o estudo de caso foi concentrado no Windows.

As entrevistas semi-estruturadas foram realizadas com 6 membros-chave da equipe de refatoração do Windows, tendo o objetivo de capturar a percepção deles com relação ao tema. Foi analisado, também, dois fatores (dependências e defeitos) no histórico de versões com o objetivo de avaliar os benefícios de refatoração. O fator "dependência" foi escolhido de acordo com o objetivo da equipe de refatoração, que era diminuir o número de dependências indesejáveis entre os módulos. Já o fator "defeito" foi escolhido pois os participantes do estudo afirmaram que há risco em realizar refatoração pois pode adicionar defeitos no sistema.

Como resultados, o artigo apresenta que a definição de refatoração, na prática, é mais amplo do que "transformações do programa que preservam comportamento". Os desenvolvedores percebem que a atividade envolve custos e risco para o projeto, além de ser necessário um suporte ferramental, além dos apresentados nas IDEs. O estudo de caso com o Windows mostrou como a refatoração do sistema foi desenvolvido em uma empresa grande. Por fim, a análise quantitativa mostrou que houve redução no número de dependências e defeitos.

No artigo "Five years of Software Architecture Checking: A Case Study of Eclipse" [6], Brunet *et al.* afirmam que desenvolvedores, arquitetos e interessados no tema costumam investir esforço para manter uma boa arquitetura do sistema. Porém, no decorrer do desenvolvimento, o código fonte vai se desviando da arquitetura documentada. Para isso, diversas ferramentas são desenvolvidas com o objeto de manter a concordância entre arquitetura e código fonte. Embora existam muitas ferramentas, pouco se sabe sobre sua utilização na prática. O Eclipse armazenou os relatórios dos últimos 5 anos de sua ferramenta de verificação (criada pelos próprios desenvolvedores). Brunet *et al.* basearam seus estudos em 3 questões de pesquisa: que tipos de regras são escritas? Que tipos de violações arquiteturais ocorrem? Como os desenvolvedores lidam com essas violações?

Foram analisados os relatórios gerados pela ferramenta de verificação arquitetural do Eclipse (de 2008 a 2013) sobre os 16 principais *plugins*. Além disso, foram entrevistados 7 desenvolvedores do Eclipse com o objetivo de entender como eles lidam com os resultados da verificação de conformidade arquitetural. Como resultados, obtiveram que dentre as 838 regras arquiteturais descritas, a maioria refere-se a aspectos hierárquicos (extensão,

implementação, reescrita). 226 regras são relacionadas à chamadas a métodos e 38 de instanciação.

Muitas violações ocorrem com relação a permissão de acesso e muitas delas estão concentradas em algumas entidades da arquitetura. Por fim, os desenvolvedores consideram algumas violações muito caras para serem corrigidas e outras consideram que são exceções às regras. Normalmente, as regras são atualizadas para lidarem com as exceções que surgem, porém, não são todas as exceções que acarretam em mudanças na regras. Isto ocorre pois modificar uma regra significa editar os arquivos de configuração, o que os fazem adiar ou negligenciar essa atividade. Ainda que muitas violações permaneçam sem correção, os desenvolvedores tomam medidas para eliminar as dependências indesejadas naquelas que consideram relevantes para o projeto.

No artigo "Mature Architecting - A Survey about the Reasoning Process of Professional Architects" [48], Heesch *et al.* afirmam que arquitetar é um processo de tomar decisões e, embora existam muitas abordagens e ferramentas para auxiliar as várias atividades de arquitetura, nenhuma delas orientam sobre a racionalidade da tomada de decisão. Como motivo, os autores apontam o baixo conhecimento sobre a forma como os arquitetos de *software* tomam suas decisões. Para tanto, os autores realizaram um *survey* com 53 arquitetos de *software* com o objetivo de entender como eles raciocinam em seus projetos.

O estudo foi baseado em 3 questões de pesquisa: como os arquitetos de *software* definem o escopo e priorizam o problema durante a sua análise? Como os arquitetos propõem soluções? Como os arquitetos escolhem a solução dentre as propostas? Os participantes do estudo foram profissionais que tenham trabalhado em desenvolvimento de sistemas nos últimos 5 anos e tenham arquitetado nos últimos 2 anos. A maioria dos participantes trabalham em grandes companhias.

Como resultados, obtiveram que é necessário ter uma profunda compreensão dos requisitos e do problema para arquitetar de forma bem sucedida. Os requisitos devem ser priorizados e os mais difíceis devem ser realizados em primeiro lugar. Os requisitos devem ser documentados pois são parte importante da lógica por trás das decisões arquiteturais. Quando o tempo e orçamento forem muito limitados, é possível considerar menos opções de *design*, principalmente se o arquiteto já tenha a utilizado em outros projetos. O arquiteto deve procurar simplicidade ao tomar suas decisões. Por fim, as soluções candidatas para a

arquitetura devem ser validadas. A arquitetura deve ser regularmente avaliada para garantir coerência entre as decisões e para descobrir restrições ocultas. E a documentação completa das decisões arquiteturais podem reduzir o esforço para sua avaliação.

No artigo "What Industry Needs from Architectural Languages: A Survey" [25], Malavolta *et al.* afirmam que, normalmente somos confrontados com pesquisas, definições, linguagens e ferramentas relacionados a certos temas mas que, muitas vezes, existe uma lacuna entre a teoria e a prática. Este mesmo contexto se aplica à arquitetura de *software* e às linguagens que são desenvolvidas para esta área. Dezenas de linguagens foram desenvolvidas nos últimos anos, mas não está claro se elas conseguem satisfazer as necessidades da indústria.

O objetivo deste estudo foi entender os profissionais com relação aos pontos fortes, as limitações e as necessidades relacionadas às linguagens arquiteturais. Para tanto, foram selecionados profissionais da indústria que tenham trabalhado com diferentes tipos de linguagens arquiteturais e foi utilizado o método bola de neve para conseguir mais respondentes. Participaram, deste estudo, 48 profissionais de 40 empresas de TI em 15 países diferentes. Desses 48, 23 preencheram o *survey* e 25 foram entrevistados com base no mesmo.

O *survey* continha 51 perguntas, principalmente com respostas abertas e tinham o objetivo de responder as seguintes questões de pesquisa: quais são as necessidades de descrição dos profissionais? Quais recursos das linguagens arquiteturais existentes são úteis (ou não) para os profissionais? Para realizar o estudo, inicialmente foi feito um levantamento sistemático das linguagens arquiteturais existentes. Depois, foi planejado o público alvo do estudo, além das perguntas que constariam no *survey*. Por fim, foi realizada a análise das respostas a partir de codificação dos dados.

Como resultados principais do estudo, Malavolta *et al.* observaram que as linguagens arquiteturais acadêmicas não suprem as principais necessidades da indústria. Diferentemente dos profissionais da indústria, os profissionais da academia compartilham a crença da necessidade de linguagem formal e de domínio específico. Dentre as funcionalidades úteis para a indústria estão sintaxe gráfica, ferramentas de suporte e suporte para múltiplas visões da arquitetura. Enquanto isso, a engenharia reversa está entre as funções não úteis para a indústria. Por fim, perceberam que entender o que os arquitetos de *software* precisam pode ajudar a direcionar os pesquisadores da área e que as linguagens normalmente utilizadas na

indústria são as construídas por ela mesma (não são oriundas da academia).

2.3 Ferramentas de suporte à Verificação de Conformidade Arquitetural

Segundo Sjoberg *et al.* [42], em todos os campos da Engenharia de *Software* deve-se realizar pesquisas empíricas para entender o quão importante são as diferentes tecnologias existentes que auxiliam as atividades da área. Com esse tipo de conhecimento é possível criar novas tecnologias mais úteis, assim como deve auxiliar a indústria a tomar suas decisões.

No âmbito do desenvolvimento e evolução de *software*, há várias pesquisas realizadas com o objetivo de encontrar formas simples e práticas de verificar a arquitetura. Em sua maioria, a automatização de alguma etapa no processo de verificação é uma questão de grande importância, visto que realizar uma verificação arquitetural de forma manual, muitas vezes, se torna uma atividade muito complexa, principalmente se o sistema é de larga escala [34].

Sangal *et al.* [37] desenvolveram uma ferramenta de análise estática que tem o objetivo de fornecer aos arquitetos uma forma de visualização gráfica do sistema, além de mostrar possíveis deficiências da arquitetura implementada. A ferramenta foi chamada de LDM e é baseada no conceito de DSM (*Dependency Structure Matrix* ou Matriz de Dependência Estrutural), permitindo representar e gerenciar dependências interclasses ou interpacotes em sistemas orientados a objetos.

Uma DSM é uma matriz quadrada cujas colunas e linhas são classes/pacotes (depende do objetivo do usuário) e o valor de cada célula da matriz indica se há dependência ou não entre os componentes presentes na linha e coluna da célula em questão. Por exemplo, considerando um sistema que possui as classes A, B e C (Figura 2.1).

	A	B	C
A		X	X
B			
C		X	

Figura 2.1: DSM de um sistema imaginário.

A partir da Figura 2.1 é possível ter o conhecimento de que a classe B possui dependência

com as classes A e C. Enquanto isso, a classe C possui dependência com a classe A. Se for o objetivo do arquiteto de *software* saber o nível de acoplamento entre as classes (baseado em números, em quantidade de dependências entre as classes), então, cada célula, ao invés de possuir um "X", teria um número indicando quantos acessos foram realizados de uma classe para outra.

Para realizar a verificação arquitetural, LDM dispõe de uma linguagem baseada em regras de *design*. Basicamente, usa-se *"can-use"* e *"cannot-use"* para indicar dependências que podem ou não, respectivamente, acontecer entre dois componentes. Após extrair as dependências automaticamente do código fonte, os arquitetos de *software* utilizam a ferramenta para escrever as regras de *design* e realizar a checagem de conformidade. As dependências inconsistentes com a arquitetura planejada são indicadas ao usuário por um triângulo vermelho.

Terra *et al.* [47] realizaram uma avaliação da ferramenta onde mostram que "matrizes de dependências representam um instrumento simples e, ao mesmo tempo, poderoso para visualizar e raciocinar sobre a arquitetura de um *software*". Porém, há limitações na expressividade da linguagem disponibilizada pela ferramenta LDM.

É possível notar, após leituras sobre a ferramenta, que a automatização presente em LDM para realizar a verificação arquitetural é pequena. É preciso escrever regras de *design*, gerar a DSM com as dependências, mandar realizar a verificação arquitetural para, então, analisar as violações presentes. Esse processo não se torna prático no dia-a-dia do desenvolvimento de um *software* pois, a cada modificação realizada no código fonte, se quiser realizar a verificação, é necessário realizar os passos novamente na ferramenta.

Murphy *et al.* [28], tendo o conhecimento de que os modelos de alto nível gerados por engenharia reversa são diferentes dos modelos criados pelos engenheiros/arquitetos de *software*, desenvolveram os modelos de reflexão. Com essa pesquisa é possível que os desenvolvedores de software se baseiem em modelos de alto nível para obter conhecimento sobre o *software*, realizar reengenharia e/ou fazer checagem de conformidade arquitetural.

A abordagem proposta por Murphy *et al.* consiste na definição, pelo engenheiro de *software*, de um modelo de alto nível de interesse (arquitetura planejada), extração do modelo fonte a partir do código fonte (utilizando uma ferramenta como, por exemplo, SAVE [23; 24]) e definição de um mapeamento declarativo entre os dois modelos gerados. A ferramenta

utilizada calcula, então, o modelo de reflexão que mostra os pontos onde os dois modelos (planejado e implementado) concordam e discordam. O engenheiro analisa e interpreta o modelo de reflexão e, se necessário, modifica o modelo de entrada (planejado) para fazer um novo modelo de reflexão. Esta é uma atividade iterativa, como é possível ver na Figura 2.2.

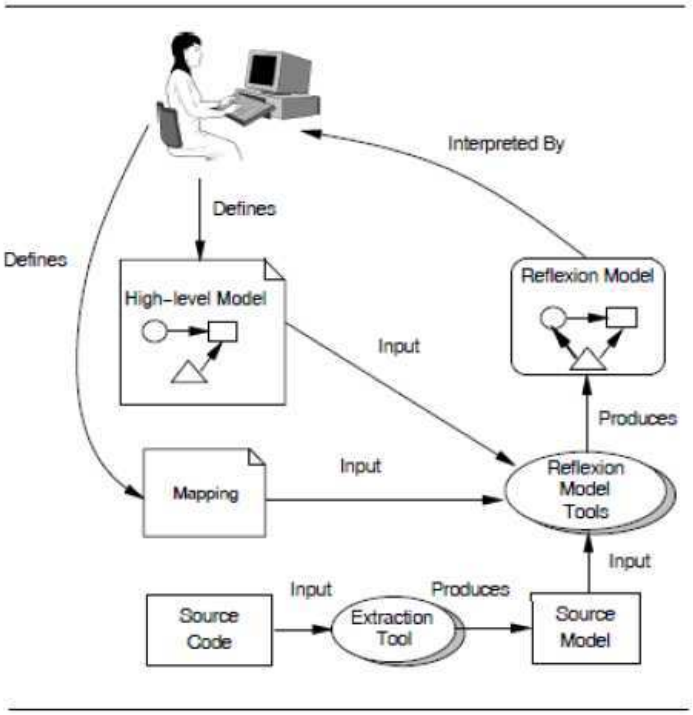

Figura 2.2: Abordagem utilizada com modelos de reflexão [28]

Apesar de permitir uma visualização de alto nível da arquitetura do *software* (tanto a planejada quanto a implementada), essa abordagem não é expressiva o suficiente, pois algumas dependências, que podem ser estabelecidas em sistemas orientados a objetos, podem ficar de fora da representação [47]. Além disso, o processo ainda é muito manual, exigindo do engenheiro/arquiteto a manutenção do modelo de entrada e do mapeamento entre os modelos. Por esse motivo, Terra *et al.* acreditam que, por tomar um certo tempo para realização do processo, é provável que os arquitetos apliquem a ferramenta apenas quando tiverem versões estáveis do sistema.

Eichberg *et al.* [16] desenvolveram a linguagem lógica de domínio específico chamada LogEn (Logical Ensembles) com o objetivo de permitir a checagem de conformidade arquitetural contínua. A checagem contínua é fundamental para o entendimento do código, reuso e manutenibilidade. Ela permite que problemas sejam consertados, praticamente, assim que

inseridos no código (pré-requisito para prevenir contra a erosão arquitetural). Para garantir a continuidade, o processo de verificação, utilizando LogEn, é integrado ao processo de compilação do Eclipse.

Com LogEn é possível expressar dependências estruturais entre grupos lógicos (chamados de *ensembles*) de elementos do código fonte. Ricardo Terra mostra em sua dissertação de mestrado [46] um exemplo de restrição escrita em LogEn (Figura 2.3). A restrição é que "somente classes Factory podem criar objetos da classe Product e de suas subclasses".

```
% Factory é um ensemble que inclui toda a classe org.foo.Factory
part_of(E, "Factory"):-
   type(ClassId, 'org.foo.Factory'),
   E= ClassId|method(E, _, ClassId, _, _)|field(E, _, ClassId, _).

% ProductCreation é um ensemble que inclui todos os construtores
% de todas as subclasses de org.foo.Product
part_of(E, "ProductCreation"):-
   type(ClassId,'org.foo.Product'),
   inherits(SubClassId, ClassId),
   method(E, SubclassId, '<init>', _, _).

% FactoryViolation é uma restrição que especifica que somente ao ensemble
% Factory é permitido criar instâncias de Product e de suas subclasses
violation(S, T, "FactoryViolation"):-
   part_of(T, "ProductCreation"),
   tnot(part_of(S, "Factory")).
```

Figura 2.3: Restrição LogEn [46]

Apesar de permitir uma checagem contínua e automática, LogEn não possui uma sintaxe tão simples (como é possível ver no exemplo dado na Figura 2.3).

Terra *et al.* [47] desenvolveram a linguagem chamada DCL (*Dependency Constraint Language*) - uma linguagem declarativa para representar a arquitetura de um *software*. Neste trabalho, arquitetura de *software* é um conjunto de decisões críticas para o sucesso do sistema e estabelecem a interação entre seus componentes.

Além da linguagem, desenvolveram uma ferramenta, chamada *dclcheck*, que é um *plugin* para o Eclipse que realiza a verificação arquitetural. Ao solicitar que a verificação arquitetural seja realizada, é mostrado, no próprio Eclipse, as violações que ocorrem no sistema (Figura 2.4).

DCL utiliza técnicas estáticas (evita sobrecarga na execução do sistema em análise), detecta violações arquiteturais assim que são implementadas no código fonte, é uma linguagem simples (declarativa) e expressiva. Porém, além de não capturar informações dinâmicas, as regras arquiteturais escritas em DCL não podem ser agregadas ao conjunto de testes execu-

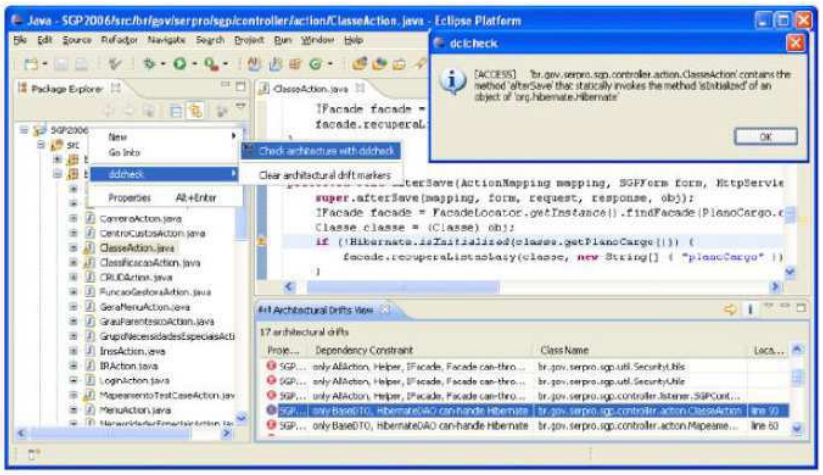

Figura 2.4: Dclcheck mostrando violações arquiteturais num sistema de estudo [47]

táveis. Ou seja, apesar de detectar as violações arquiteturais com êxito, os desenvolvedores podem optar por considerá-las ou não em suas implementações, já que não existe uma forma de "impedir".

Como é possível notar, existem várias abordagens acadêmicas para realizar a verificação arquitetural, mas nem sempre são usadas, muitas vezes porque a linguagem utilizada na verificação é diferente da linguagem usada na aplicação [5]. Nesse contexto, Brunet *et al.* desenvolveram uma API, chamada DesignWizard, que permite escrever testes de *design* para implementações em Java, usando JUnit.

Teste de *design* é um teste que verifica se uma implementação obedece a uma dada regra de projeto expresso como um algoritmo. A diferença entre um teste de *design* e um teste funcional é que aquele verifica como o *software* foi construído enquanto que o funcional verifica se o *software* faz o que deve ser feito. Testes de *design* são automatizados e podem ser escritos na mesma linguagem utilizada para desenvolver o sistema.

Com DesignWizard, os desenvolvedores podem ter um melhor entendimento do sistema, já que utiliza a mesma linguagem de desenvolvimento. Além disso, a documentação da arquitetura passa a ser executável, facilitando a manutenção da conformidade arquitetural. Como pode ser agregado ao conjunto de testes funcionais, testes de *design* são bastante úteis para garantir que as decisões arquiteturais sejam seguidas (sem precisar analisar manualmente). Porém, não é possível especificar regras que dependem de aspectos dinâmicos com DesignWizard. Além disso, os arquitetos de *software* necessitam de mais tempo para

entender a API e escrever as regras arquiteturais antes de serem repassadas para os desenvolvedores.

No âmbito das transformações de artefatos, Pires *et al.* [33] propuseram uma técnica para transformar automaticamente diagramas de classe em UML para testes de *design*. Com isso, é possível especificar a arquitetura em UML e transformá-la em testes de *design*, automatizando, assim, a verificação arquitetural. Porém, apesar de UML ser utilizada no ambiente coorporativo, a linguagem não é muito utilizada para descrição e verificação arquitetural [31].

Após essa revisão bibliográfica fica evidente que existem muitas ferramentas com o objetivo de auxiliar a atividade de Verificação de Conformidade Arquitetural. Porém, surgem as dúvidas: essas ferramentas são utilizadas? Elas são úteis? O que a indústria pensa sobre elas?

Capítulo 3

Um estudo qualitativo sobre arquitetura de *software* no desenvolvimento de sistemas reais

Como vimos nos capítulos anteriores, existe um interesse fecundo na academia em relação à pesquisas relacionadas à Arquitetura de *Software* e propostas de novas técnicas, ADLs e ferramentas que auxiliam atividades nesse sentido. O aumento significativo das publicações nos últimos anos [4] demonstra a importância e a preocupação que a academia tem com relação à essa área.

Uma de nossas inferências é de que a indústria não parece conhecer/utilizar o que é proposto pela academia. Assim, realizamos um estudo quali-quantitativo (com foco principal na etapa qualitativa) com o objetivo de investigar e entender a perspectiva da indústria e dos profissionais em desenvolvimento de sistemas reais com relação ao tema Arquitetura de *Software*.

Nosso estudo foi concebido a partir de curiosidades e reflexões que surgiram após nossas leituras e diálogos. Dessa forma, temos as seguintes indagações: A indústria conhece e utiliza as ferramentas, técnicas e/ou ADLs propostas pela academia? Como a indústria define o termo "arquitetura de *software*"? Com relação à documentação arquitetural, será que, na prática, a arquitetura é documentada? Se sim, qual é a estrutura e notação utilizada pelos profissionais?

Ainda nesse contexto, outras questões orientam nossa pesquisa: será que existe, na in-

dústria, a preocupação de realizar atividades para verificação de conformidade arquitetural? Se sim, como essas atividades são realizadas? Quais são os motivos alegados por aqueles profissionais que não documentam a arquitetura, não realizam verificação arquitetural e nem utilizam ferramentas?

Esses e outros questionamentos foram organizados e resumidos em quatro questões de pesquisa.

3.1 Questões de pesquisa

Dividimos as questões de pesquisa em quatro tópicos: definição de arquitetura de *software*, documentação arquitetural, verificação de conformidade arquitetural e ferramentas de apoio à verificação de conformidade arquitetural. Com esses tópicos pretendemos abranger o máximo de conhecimento envolvido na área com relação a como o tema é tratado na prática de desenvolvimento de sistemas reais (desde a definição até a utilização de ferramentas). Com isso, concebemos quatro principais questões de pesquisa:

RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

RQ2: Qual a visão da indústria com relação à documentação arquitetural?

RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

3.2 Metodologia

Escolhemos realizar uma pesquisa quali-quantitativa (ou mista), com um foco maior na etapa qualitativa, pois nosso objetivo principal é entender o ambiente de desenvolvimento de sistemas reais com relação à arquitetura de *software*, sua documentação e verificação e à utilização de ferramentas de apoio.

A nossa metodologia tem como principal influência/referência o trabalho de Singer *et al.* [41], denominado *"Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter"*. Este artigo utiliza o método *Grounded Theory* (GT) ou Teoria Fundamentada, para realizar seu estudo. Porém, nosso objetivo não é criar novas teorias (como

GT propõe), e sim entender o contexto do nosso ambiente de estudo. Com isso, decidimos utilizar a Análise Temática (exposto no Capítulo 2) como método de análise para a etapa qualitativa, por ser um método mais flexível ao nosso objetivo.

Dividimos, então, nosso estudo em três etapas: *Survey* Exploratório (etapa mista), Entrevistas (etapa qualitativa) e *Survey* Confirmatório (etapa quantitativa). Cada etapa possui uma metodologia, análise e resultados particulares. Assim, optamos por descrevê-los em detalhes nos capítulos 4, 5 e 6. É importante frisar que, com as três etapas, realizamos, então, uma triangulação metodológica dos dados [10], uma vez que capturamos os dados e as informações sobre nosso tema de três formas diferentes para responder nossas questões.

Visando explorar, em alto nível, com perguntas genéricas, o ambiente de desenvolvimento de sistemas no contexto do nosso estudo, a primeira etapa deste trabalho foi um *survey* exploratório. Com esta etapa, objetivamos ter uma noção básica sobre definição, documentação e verificação arquitetural e suas ferramentas de apoio na prática. O *survey* foi enviado para 149 (cento e quarenta e nove) profissionais da área que podem entender sobre Arquitetura de *Software* (utilizando o método Bola de Neve), além de 4 (quatro) grupos de discussão (openstack, eclipse-dev e duas listas de arquitetos de *software* no LinkedIn). A análise desta etapa foi utilizada para a criação do roteiro de perguntas da etapa de entrevistas.

Com a etapa de entrevistas tivemos o objetivo de aprofundar os resultados obtidos no *survey* exploratório, possibilitando-nos um melhor entendimento sobre o nosso objeto de estudo e desenvolver as respostas para as questões de pesquisa. Entrevistamos os profissionais que responderam o *survey* exploratório e se voluntariaram para tal etapa. Nesta etapa, fizemos perguntas mais objetivas e direcionadas às respostas anteriores dos participantes. Transcrevemos todas as entrevistas e as codificamos utilizando o método de Análise Temática.

Por fim, com a última etapa, o *survey* confirmatório, nosso objetivo foi verificar nossas principais conclusões. Com ele, quantificamos a concordância de outros profissionais da área com relação às nossas constatações das etapas anteriores. Enviamos o *survey* para os usuários do GitHub que possuem mais de 100 seguidores. Utilizamos este critério por considerar que quem possui esta quantidade de seguidores é um profissional respeitado na rede social de desenvolvedores.

Nos próximos capítulos, apresentamos as três etapas com suas respectivas metodologias, resultados e análises.

Capítulo 4

Etapa 1: *Survey* Exploratório

A primeira etapa de nosso estudo consistiu de um *survey* exploratório cujo objetivo principal é a exploração do ambiente de desenvolvimento de sistemas reais no contexto de arquitetura de *software*, verificação de conformidade arquitetural, as ferramentas de apoio que a indústria tem conhecimento e quais as suas limitações. Esta etapa nos permitiu ter uma noção, em alto nível, de como o contexto de arquitetura está inserido no desenvolvimento de sistemas reais. Neste capítulo, apresentaremos o planejamento, execução, ameaças à validade, resultados e as análises desta etapa.

4.1 Planejamento

Nos baseamos nas quatro questões de pesquisa (expostas no capítulo anterior) para elaborar as perguntas deste *survey*. Fizemos perguntas relacionadas a vários aspectos desde o perfil do participante, passando por questões sobre arquitetura, documentação, verificação e ferramentas de apoio, até perguntas para encontrar novos respondentes para nosso questionário. O roteiro de nosso *survey* exploratório pode ser visto por completo no Apêndice A.

A amostra de candidatos a respondentes do *survey* foi construída a partir de uma lista de arquitetos (quando existia este papel) e *core developers* de empresas/projetos presentes em nossa rede de contatos. Nesta etapa, tivemos dificuldade de encontrar profissionais que exercessem o papel de arquiteto de *software* (nosso foco de candidatos, inicialmente). Também fizeram parte dos candidatos a respondentes os *core developers* do Fraunhofer, Oracle, Arch-Linux e Linux Kernel. Além disso, enviamos o *survey* para duas listas no LinkedIn voltadas

para arquitetos de *software* e as listas de desenvolvimento do OpenStack e do Eclipse. Para complementar a lista de candidatos, utilizamos o método bola de neve, solicitando sugestões, dos próprios respondentes, de profissionais que pudessem responder nossa pesquisa.

4.2 Execução

Enviamos todos os *e-mails* com o *link* para o *survey* exploratório para 149 profissionais. 102 profissionais foram escolhidos a partir dos critérios elencados na subseção anterior e 47 foram indicados no modelo bola de neve. Dos 149 profissionais, 36 responderam o *survey*, obtendo uma porcentagem de 24,1% de respostas.

Além disso, encaminhamos o *survey* para 4 listas (openstack, eclipse-dev e duas listas de arquitetos de *software* no LinkedIn). Destas, obtivemos 22 respostas. Portanto, no total, tivemos 58 respondentes. Além disso, 2 pessoas (que não responderam o *survey*) nos enviaram *e-mails* com algumas informações sobre arquitetura de *software* e checagem de conformidade arquitetural em seu ambiente de desenvolvimento. O período de recebimento de respostas foi entre 11/09/2014 a 06/10/2014.

4.2.1 Dados profissionais dos respondentes

Os respondentes de nosso *survey* tem, em média, 12,34 anos de experiência em Tecnologia da Informação, onde o mais experiente tem 30 anos e o menos experiente tem 7 meses. Dos 58 respondentes, 57 afirmaram ter algum nível de experiência em arquitetura de *software*. As respostas do participante que afirmou não possuir experiência nesta área não foram tão destoantes das demais, e, portanto, consideramos que não interfere em nossos resultados. Por esse motivo, optamos por não retirá-la de nossa análise.

Contabilizamos que os papéis desempenhados pelos respondentes variam entre desenvolvedor, arquiteto de *software*, *designer* de *software*, engenheiro de *software*, gerente de projeto, gerente de TI, analista de sistema, engenheiro de teste, administrador de rede, coordenador de desenvolvimento, líder de equipe e *partner* (Figura 4.1).

Contabilizamos, também, que as linguagens nas quais os profissionais tem experiência variam entre Java, C, C#, C++, Python, PHP, Ruby, JavaScript, Bash, Perl, Haskell, Visual Basic, PLSQL, .Net, Assembler, shell, yacc, lex, make, ADA, Objective C, Assembly, Action

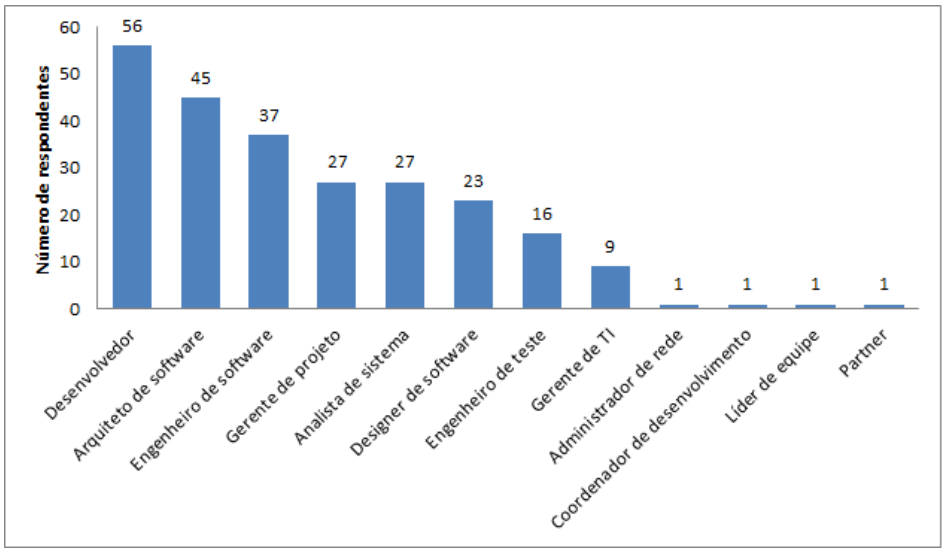

Figura 4.1: Papéis desempenhados pelos respondentes.

Script, prolog, lisp, Pascal, Scala, TCL, R, Delphi, Go (Figura 4.2).

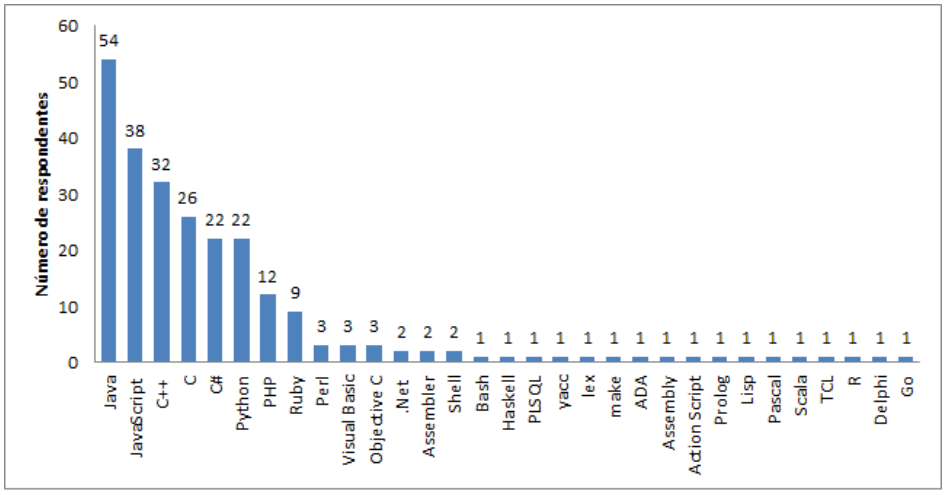

Figura 4.2: Linguagens de programação nas quais os respondentes possuem experiência.

4.3 Resultados e Análises

Nesta seção, apresentamos nossos resultados e análises decompondo-os de acordo com as questões de pesquisa, visando facilitar a leitura dos resultados.

4.3.1 RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

Inicialmente, em nosso *survey* (cujo roteiro está no Apêndice A), disponibilizamos a definição de arquitetura de *software*, que utilizamos para este estudo, e questionamos a relevância de nossa definição no cotidiano profissional dos respondentes. Como apresentamos na Figura 4.3, 52 respondentes (89,65%) afirmaram que a definição dada por nós varia de "relevante" (nível 4) para "totalmente relevante" (nível 6) no seu cotidiano profissional.

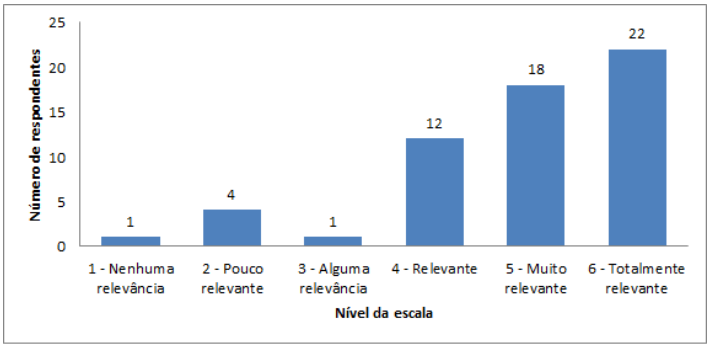

Figura 4.3: Nível de relevância da definição apresentada pelos pesquisadores desse estudo.

Isto nos dá um indício de que a maioria dos profissionais que responderam nosso questionário considera que componentes e seus relacionamentos são importantes e fazem parte de uma arquitetura de *software* (níveis 4, 5 e 6). Porém, ao questionarmos sobre outros aspectos que não estavam contemplados em nossa definição (mas que eles consideravam importantes estarem presentes) apenas cinco respondentes (8,6%) se apresentaram satisfeitos com nossa definição e não acrescentaram nenhum aspecto extra.

Observamos os respondentes (é o caso do Respondente 58, por exemplo) elencarem requisitos não-funcionais como um aspecto importante da arquitetura de um sistema. Aspectos como segurança, desempenho, escalabilidade, eficiência, complexidade e qualidade (tanto da arquitetura quanto do *software*) foram citados durante nosso questionário.

> Pensar em arquitetura apenas como componentes e relacionamentos não tem sido suficiente nos meus designs. Mesmo esse sendo o escopo da definição de arquitetura assumido para essa pesquisa, particularmente penso em componentes e dependências como uma parte relativamente fácil de manter sob controle. O que me toma muito mais tempo (talvez por ser mais importante, ou talvez por apenas custar mais pensar sobre isso) é como vou acomodar no meu design soluções elegantes para aspectos não funcionais. Ultimamente os principais requisitos não funcionais que tomam o meu tempo são desempenho, escalabilidade e segurança (Respondente 58).

Também foram citados durante nosso questionário os aspectos de negócio, incluindo modelo de dados; os aspectos comportamentais do *software*; os aspectos de risco para o *software*, como, por exemplo, gargalos computacionais, sessões críticas e perigo em caso de falha de algum componente; os aspectos de *deployment*; os aspectos sobre padrões de *software*; os detalhes de integração entre sistemas; as tecnologias, como, por exemplo, plataformas e detalhes relacionados às APIs utilizadas no desenvolvimento do sistema; alguns aspectos formais, como a norma de descrição arquitetural (AD - Architectural Description) IEEE 42010; e, por fim, o ambiente operacional de desenvolvimento e implantação do *software* (como é o caso do Respondente 9).

> Temos aplicações perfeitas do ponto de vista 'arquitetural' mas que não consegue se encaixar nos ambientes existentes. A arquitetura de um 'leão' é perfeita para as savanas africanas, mas não teria muito sucesso na floresta amazônica ou no polo norte (Respondente 9).

Capturamos, ainda, observações interessantes de profissionais que possuíam definições diferentes da que apresentamos no questionário (Respondentes 11 e 24), além de sugestões de melhoria para nossa definição (Respondente 56).

> No meu cotidiano, vários outros aspectos são trabalhados como parte da arquitetura e em um nível bem mais alto do que esse descrito aqui. A arquitetura precisa tratar de problema(s) a ser(em) resolvido(s) ou necessidade(s) sanada(s) pelo produto (isso define o que), diferencial em relacão aos concorrentes e como o produto endereça essas necessidades (isso define o como). É essencial que a arquitetura consiga fazer essa ponte entre executivos e técnicos para que todos trabalhem em sintonia. A arquitetura precisa treinar os técnicos no o que e porque e conseguir justificar o projeto para executivos detalhando beneficios e vantagens. Esse é o primeiro trabalho e parte da arquitetura. Depois disso, a arquitetura trata de definições de "use cases"para só então definir componentes e sua interacão. Talvez na sua visão, essas definições venham em uma etapa anterior ou seja nomeada de outra forma mas é parte essencial da definição do produto e parte integrante da arquitetura do produto (pelo menos, no meu cotidiano) (Respondente 11).

> The architecture is much more a way to visualize how responsibility/functionality is split between different parts of a system. Usually a formal definition kills this. I have often worked from a SW architecture drawn on a whiteboard - which was much better than any UML/whatever formal syntax (Respondente 24).

> I would also recommend to not rely on some programming language's notion of component (e.g. class, package etc.) because in practice, you will want to express stronger architectural rules than what the language allows (Respondente 56).

A partir disso, temos o indício de que a definição de arquitetura de *software* não pode se restringir a componentes e seus relacionamentos (aspectos estruturais do código) pois, de 58 respondentes, apenas 5 (8,6%) afirmaram não haver nenhum acréscimo à nossa definição.

Por outro lado, o alto índice de relevância de nossa definição (89,65%) nos faz refletir que componentes e seus relacionamentos podem ser considerados aspectos centrais da definição da maioria dos respondentes. Assim, temos indícios de que, semelhante à academia [8], existem diversas definições para arquitetura de *software* utilizadas na indústria, com variados aspectos. Suspeitamos que a definição utilizada, assim como os aspectos que compõem a arquitetura, na prática, depende de profissional para profissional e de ambiente para ambiente.

4.3.2 RQ2: Qual a visão da indústria com relação à documentação arquitetural?

Com relação à documentação arquitetural, quando questionamos sobre os sistemas que os respondentes desenvolviam, 25 dos 58 respondentes (43,10%) afirmaram que a arquitetura do sistema é bem documentada, enquanto que 33 (56,90%) responderam que não é bem documentada.

Dentre as justificativas para a arquitetura não ser bem documentada estavam: falta de tempo, pressão para entregas do *software*, priorização de outras atividades (como implementação e testes), custo alto, tamanho da equipe, falta de experiência na equipe quanto à documentação, cultura da equipe (considera-se uma atividade não importante, não é exigência dos clientes, não acreditam em documentação), utilização de metodologia ágil para desenvolvimento do *software*, complexidade do sistema e o sistema é opensource. Corroboramos nossas afirmações com citações de alguns respondentes.

> Todas as desculpas são válidas.... Pressão no desenvolvimento, entendimento por muitos de que isto não é importante, mudanças ao longo do tempo... Mas quando temos que tomar a decisão de alterar um componente isso pesa muito, quase sempre tomamos a decisão baseada na experiência de alguém, mas não porque temos certeza que componente X não precisa mais ser usado (Respondente 8).

> The system was created using agile methodologies. I believe this is one of the reasons for the lack of good documentation. Agile Practitioners usually avoid too much documentation, often resulting in 'no' documentation at all. The major problem is to document the natural evolution of the system. This specific system is 10 years old now and still being used, although I not work with it anymore (Respondente 30).

> Because we do not believe in documentation, it's too much overhead. It's also a very structured approach and imposes some distance between the architect and developers, and we always avoid that distance; it's better to work within evey team, on a day-to-day basis, and let architectural concepts be part of everyday work (Respondente 38).

> Pela falta de um processo e um framework que permita registrar, integrar e atualizar decisões da micro arquitetura para a visão da macro arquitetura. Por falta de cultura também. Pela sensação de que a informação nunca vai ser consumida, pela pressão do negócio. Se tivermos que optar entre investir em documentação ou testes, preferimos testes sempre (Respondente 52).

Consideramos que os problemas citados pelos respondentes para justificar a não documentação arquitetural também são enfrentados por profissionais que documentam a arquitetura, pois são problemas naturais e passíveis de surgimento em qualquer projeto em desenvolvimento. Isso nos leva à reflexão: o que diferencia um profissional que documenta a arquitetura de um profissional que não documenta? Seria uma questão de prioridade ou compreensão?

Contabilizamos, ainda, que 22 dos 25 que afirmaram possuir uma arquitetura bem documentada afirmaram, também, que houve evolução na arquitetura do sistema (mudanças nas regras arquiteturais). E 15, dos mesmos 25, também afirmaram que a equipe atualiza a documentação (Figura 4.4) com uma frequência de "frequentemente" (nível 4) a "sempre" (nível 6).

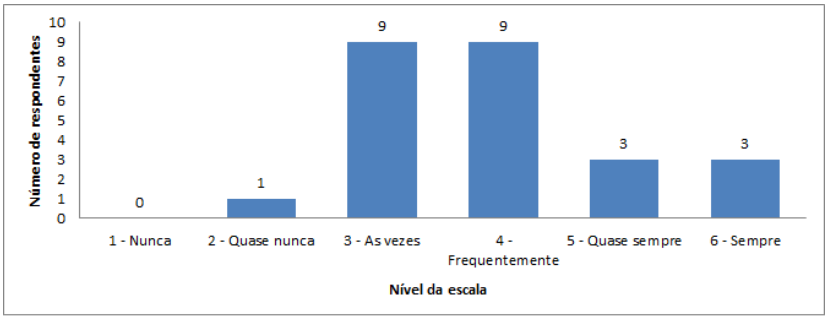

Figura 4.4: Frequência de atualização da documentação arquitetural.

A partir da Figura 4.4, podemos observar que apenas 15 dos 58 respondentes (25,8%) documentam a arquitetura e atualizam suas documentações com uma certa frequência (níveis 4, 5 e 6). Isto nos dá indícios de que a atividade de documentar a arquitetura não parece possuir prioridade no processo de desenvolvimento de sistemas reais. Sendo assim, refletimos se os profissionais da indústria consideram a atividade de documentação arquitetural uma atividade importante.

Além disso, os tipos de documentos utilizados para registrar a arquitetura do *software*, segundo os respondentes, variam entre arquivos doc./docx.; ADLs; UML; *e-mails*; fotos

em quadros; diagramas de processos; *wiki*; ferramenta própria da empresa; *power points*; doxygen; SysML; Simulink; Proteus; diversas ferramentas de *deployment* para plataformas embarcadas; *websites*; *books*; modelos Archimate; metadados formais (plugin.xml etc) (Figura 4.5).

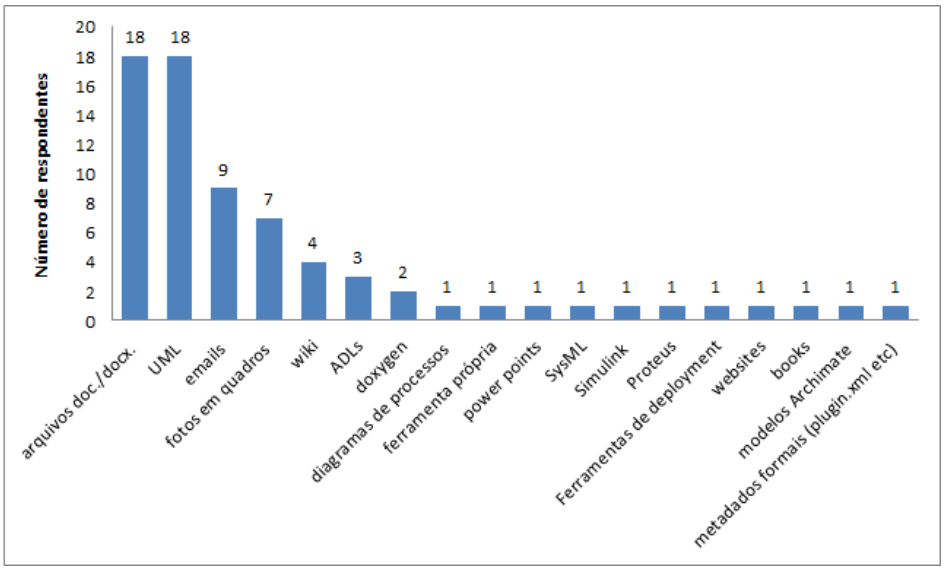

Figura 4.5: Tipos de documentos utilizados para registrar a arquitetura.

Isto nos leva a questionar se, com tantos tipos de documentos sendo utilizados, há uma formalização na documentação. Existe alguma notação padrão que seja utilizada? Também, com essas respostas, podemos ver que 72% dos que afirmaram ter boa documentação arquitetural utilizam doc./docx. ou UML para documentar. Ou seja, podemos suspeitar que a linguagem utilizada para a documentação arquitetural pode ser, normalmente, informal e não permite realizar verificação de conformidade arquitetural de forma automática.

Realizamos, ainda, outras perguntas sobre o conhecimento da equipe e discussão sobre arquitetura de *software*. Para analisar melhor as respostas, dividimos os respondentes em dois grupos: os que possuem sistemas com arquiteturas bem documentadas (BD) e os que possuem sistemas com arquiteturas não documentadas (ND).

Perguntamos: *"Numa escala de 1 a 6, onde 1 significa 'a maioria dos desenvolvedores desconhece as regras' e 6 significa 'todos os desenvolvedores têm pleno conhecimento das regras', qual o nível de conhecimento da equipe sobre as regras arquiteturais desse sistema em questão?"*. Como é possível ver na Figura 4.6, a maioria das pessoas (não importa se tem um sistema com arquitetura bem definida ou não) afirmaram que a equipe possui um nível

de conhecimento entre 4 e 6 acerca da arquitetura do *software*. Para deixar claro, os níveis variavam da seguinte forma: 6 - todos conhecem; 5 - quase todos conhecem; 4 - a maioria conhece; 3 - metade conhece; 2 - menos da metade conhece; 1 - todos desconhecem.

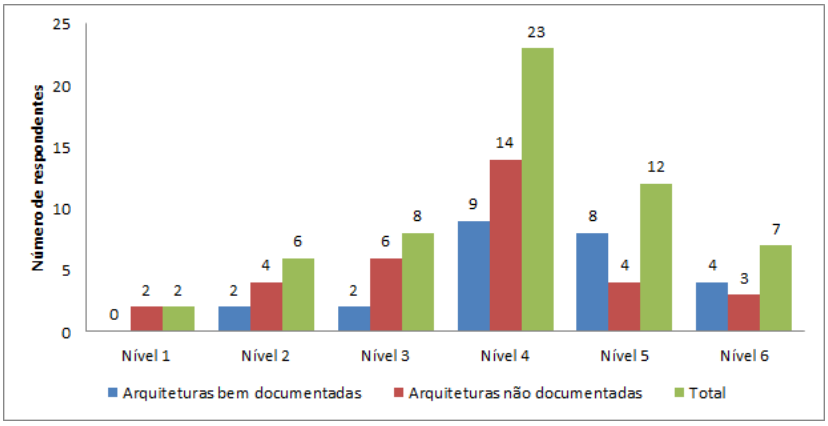

Figura 4.6: Nível de conhecimento da equipe sobre a arquitetura do *software*.

Observamos que 84% dos profissionais que possuem uma boa documentação arquitetural afirmam que sua equipe possui um nível de conhecimento bom (nível 4 - 6) sobre a arquitetura. Enquanto isso, 63,6% dos profissionais que não possuem a arquitetura bem documentada afirmam que sua equipe possui um nível de conhecimento bom (nível 4 - 6) da arquitetura do sistema. Isso nos dá indícios de que os profissionais confiam que sua equipe é conhecedora da arquitetura do sistema, independentemente de haver documentação arquitetural ou não. Executamos o teste exato de Fisher para analisar se as respostas dependiam de ter ou não uma arquitetura bem documentada. Com um *p-value* de 0.3267 não foi possível observar estatisticamente, para nossa amostra, se a diferença era significativa. Apenas um estudo comparativo mais aprofundado nos permitiria afirmar e entender melhor tal questão.

Perguntamos, também, quantas pessoas participam da tomada de decisões arquiteturais e, como é possível ver na Figura 4.7, a tomada de decisão (não importa se tem um sistema com arquitetura bem definida ou não) ficou dividida, quase que igualmente, entre as opções: todos os desenvolvedores, apenas um grupo pequeno e apenas os *seniores*. Isso nos dá indícios de que a quantidade de pessoas envolvidas na tomada de decisão não interfere muito na arquitetura ser bem documentada ou não. Executamos, novamente, o teste exato de Fisher com o mesmo objetivo dito anteriormente. Com um *p-value* de 0.8079 também não foi possível observar estatisticamente, para nossa amostra, se a diferença era significativa. Como

dito anteriormente, apenas uma pesquisa mais minuciosa sobre este ponto poderia afirmar e entender isto.

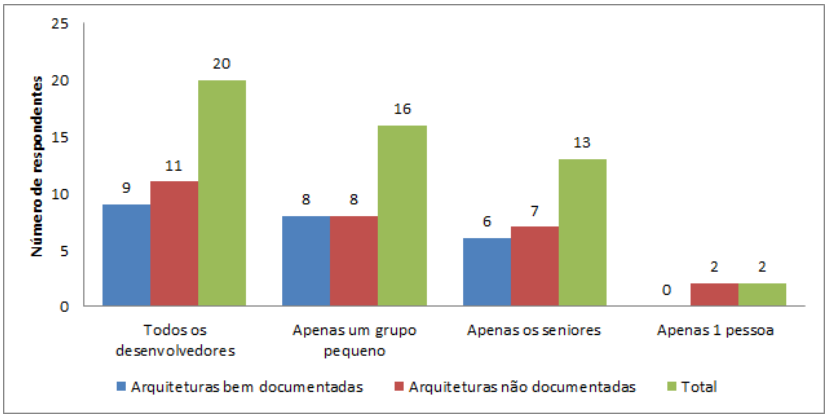

Figura 4.7: Quem é o responsável pela tomada de decisões.

Também foram respostas para esta questão:

- toda a equipe;
- 3 pessoas;
- depende do tipo de metodologia utilizada no projeto;
- a pessoa que desenvolver, arquiteta sua parte;
- todos participam da reunião, mas os *seniores* geralmente são os que tomam as decisões;
- um grupo de desenvolvedores, mas não um grupo pequeno.

A partir disso, podemos suspeitar que não há um padrão, entre as empresas e equipes de desenvolvimento, da quantidade de profissionais envolvidos na tomada de decisão.

Questionamos, ainda, com que frequência a equipe discute sobre arquitetura. Disponibilizamos 6 níveis de frequência para responder esta questão: nunca (nível 1), quase nunca (nível 2), as vezes (nível 3), frequentemente (nível 4), quase sempre (nível 5) e sempre (nível 6).

Segundo a Figura 4.8, a maioria das pessoas (não importa se tem um sistema com arquitetura bem definida ou não) afirmaram que a equipe discute a arquitetura do *software* entre "frequentemente" (nível 4) e "sempre" (nível 6).

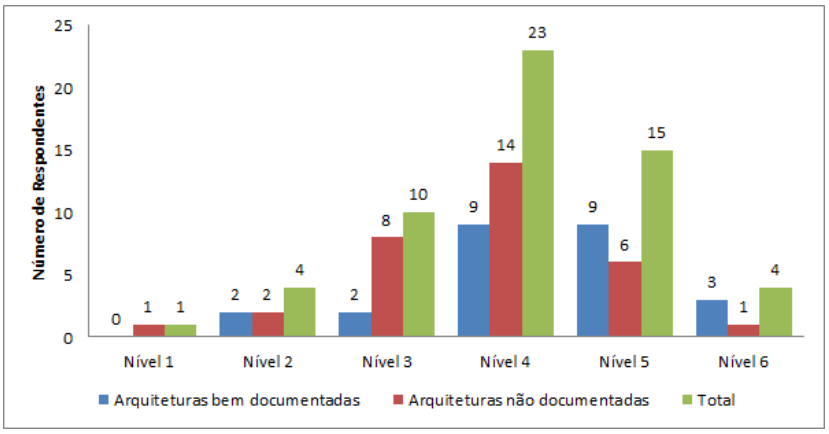

Figura 4.8: Frequência de discussão sobre arquitetura de *software*.

Observamos que 84% dos desenvolvedores que consideram ter uma documentação bem definida em seus projetos, afirmam discutir a arquitetura frequentemente, quase sempre ou sempre. Para os que consideram não possuir boa documentação em seus projetos, o número de desenvolvedores que afirmam realizar discussões arquiteturais com a mesma frequência cai para 63,63%. Por um lado, essa diferença numérica pode permitir a interpretação de que equipes que se preocupam mais em documentar bem a arquitetura têm uma tendência maior a discutir a arquitetura mais frequentemente. Por outro lado, a diferença pode ser relativamente pequena (considerando-se o tamanho da amostra), permitindo-nos inferir que não existe forte correlação entre possuir ou não uma boa documentação arquitetural e a frequência de discussões da equipe sobre o tema. Executamos o teste exato de Fisher para analisar se as respostas dependiam de ter ou não uma arquitetura bem documentada. Com um *p-value* de 0.2434 não foi possível observar estatisticamente, para nossa amostra, se a diferença era significativa. Imaginamos, portanto, que apenas estudos mais minuciosos podem explicar melhor se essa diferença é significativa ou não.

Além disso, quisemos saber por quais meios se discute sobre arquitetura nos projetos dos respondentes. Segundo a Figura 4.9, a discussão sobre arquitetura, no grupo que possui uma documentação bem definida, ocorre mais em *e-mails* e conversas presenciais. Já no grupo que não tem uma documentação bem definida as discussões possuem uma tendência maior de ocorrerem em conversas presenciais (30 dos 33 respondentes afirmaram discutir a arquitetura pessoalmente). Podemos suspeitar, então, que é bastante comum discutir sobre a arquitetura em conversas presenciais da equipe (não importando se a equipe documenta bem

ou não a arquitetura). O teste exato de Fisher foi executado, novamente, e com um *p-value* de 0.5372 não foi possível observar estatisticamente, para nossa amostra, se a diferença era significativa.

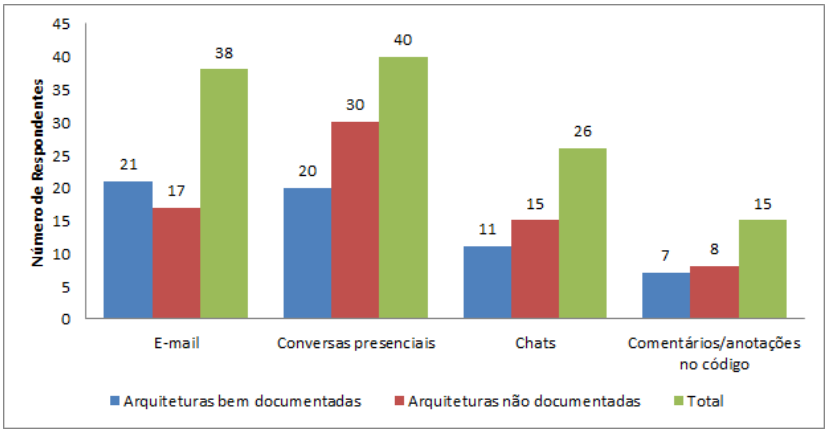

Figura 4.9: Onde se discute sobre arquitetura de *software*.

Também foram respostas que recebemos para esta pergunta: reuniões semanais e bi-semanais; na ferramenta CASE; em revisão de código; *standups*; *bug reports* (Bugzilla); *site* colaborativo; *wiki*; fóruns internos; telefonemas; documentos de *design*.

Para finalizar a etapa de perguntas sobre documentação arquitetural, questionamos o que um membro novo na equipe precisa fazer para conhecer as principais regras arquiteturais do sistema em desenvolvimento. Como respostas tivemos: a leitura de *wiki*, documento arquitetural, código fonte, comentários de código e diagramas UML; a realização de treinamentos para o novo membro conhecer o código e as tecnologias utilizadas; a conversa com membros mais experientes da equipe; a utilização de *pair programming*; a revisão de código; e, por fim, alguns afirmaram que o novo membro aprenderia sozinho, com a prática do dia-a-dia.

É interessante citar que algumas das pessoas que afirmaram não ter uma boa documentação sobre a arquitetura do sistema afirmaram, também, que um novo desenvolvedor deve (além de outras ações como conversar com pessoas experientes, ver o código fonte e programar em par) ler os documentos existentes sobre o sistema para entender a arquitetura do sistema. Ou seja, temos indícios de que mesmo os que afirmam não possuir uma boa documentação, ainda assim, tem algum tipo de documentação (mesmo que não seja completa e/ou atualizada).

A seguir, alguns comentários feitos pelos respondentes sobre esta pergunta.

> É duro falar ... mas só com o tempo ... Apesar de termos alguns documentos, poucos gestores cobram que um novo membro da equipe leia todos estes documentos antes de iniciar o trabalho (Respondente 8).
>
> By reading code. Sad, but true. Also, by asking around (Respondente 28).
>
> Reading bits and bobs of documentation even from 'unoffical' sources such ad mailing List archives or blog posts. Chatting with people over IRC. Internal training at their companies. Studying the source code (Respondente 55).

4.3.3 RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

Com relação à verificação de conformidade arquitetural, 27 dos 58 respondentes (46,6%) afirmaram haver alguma atividade com essa finalidade, enquanto 31 (53,4%) afirmaram não haver nenhuma atividade nesse sentido. Vale salientar que nem todos os respondentes apresentaram uma relação de um para um entre documentar a arquitetura e realizar alguma atividade de verificação de conformidade arquitetural. Percebemos que alguns afirmaram possuir a arquitetura bem documentada mas não realizavam verificação arquitetural (36% dos que possuem arquitetura bem documentada), enquanto outros afirmaram que não tinham uma documentação bem definida mas realizavam alguma atividade de verificação arquitetural (33% dos que não possuem arquitetura bem documentada).

As principais justificativas dadas para a não realização de atividades de verificação de conformidade arquitetural foram que a arquitetura não estava definida, é uma tarefa custosa, há pressão para entrega do produto, falta de tempo, falta de priorização da atividade, o projeto é pequeno, a arquitetura está em constante mudança, a etapa não estava no planejamento, existe um fraco suporte ferramental e as violações arquiteturais ainda não são tão danosas ao sistema. Todas as justificativas podem ser vistas na Figura 4.10.

Unindo essas justificativas com o fato de tantos profissionais não realizarem atividades de verificação de conformidade arquitetural (mais de 50%), ficamos com o questionamento de se, na prática, os profissionais consideram-na uma atividade importante.

As citações que expomos abaixo (Respondentes 3, 51 e 58) nos dão a impressão de que verificar a arquitetura só será prioridade e terá a devida atenção por todos no desenvolvimento de um sistema real se os problemas causados pelas violações arquiteturais forem danosos.

> Verificar a arquitetura é uma tarefa custosa. Muitas vezes são poucas pessoas que saberiam analisar essas inconformidades com precisão (arquitetos). E atualmente

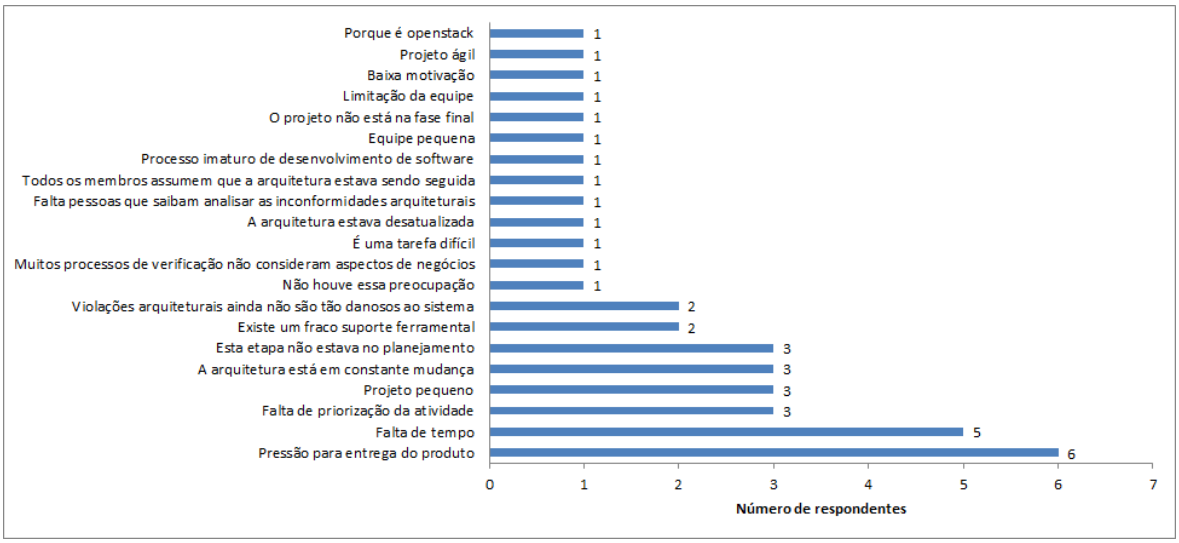

Figura 4.10: Justificativas dadas pelos respondentes para a não realização de atividades de verificação de conformidade arquitetural.

> na minha empresa, vale mais entregar o software do que garantir que há integridade arquitetural (Respondente 3).
>
> A menos que um problema surja por causa de violações arquiteturais, todos assumem que a arquitetura está sendo seguida (Respondente 51).
>
> Uma das causas é a falta de ferramentas automáticas que façam essas verificações e se integrem bem com o arcabouço de desenvolvimento que já utilizamos. Isso reduziria o custo de se fazer essa verificação, que é o segundo problema. A terceira causa é que quebras de regras arquiteturais não são (ainda) um problema regular e/ou danoso o suficiente para demandar uma verificação rotineira (Respondente 58).

Já para os que afirmaram realizar alguma atividade voltada para verificação de conformidade arquitetural perguntamos sobre a periodicidade da realização dessas atividades. Com a Figura 4.11 podemos observar que a maioria (55,5%) dos que afirmam checar a arquitetura, possui uma certa frequência de realização da atividade (diariamente ou 1 vez por semana). Porém, isso corresponde a apenas 25,8% do total de respondentes desse *survey*, o que nos mostra que é baixa a taxa de profissionais que se dedicam a esta atividade com frequência. Isto nos leva a refletir sobre qual frequência é mais adequada para este tipo de atividade, porém, apenas um estudo mais detalhado é capaz de responder essa questão.

Também foram respostas para a periodicidade da realização de atividades nesse sentido: após projetar os casos de uso e após concluir a codificação; 1 vez no projeto inteiro (com foco na segurança) e ao término de cada interação para validar conformidade com use cases e funcionalidades; a cada novo código; não há periodicidade.

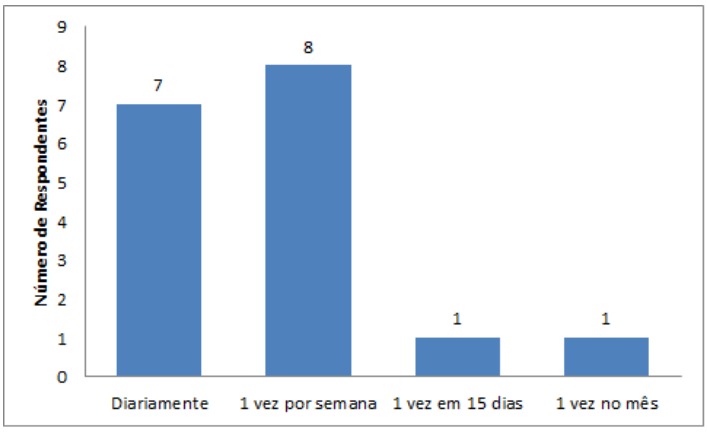

Figura 4.11: Periodicidade de atividades de verificação de conformidade arquitetural.

Questionamos, ainda, se os respondentes já tiveram problemas no *software* devido a violações arquiteturais no código. Como mostramos na Figura 4.12, os respondentes se dividem em responder entre sim e não (independente se há ou não atividades de verificação de conformidade arquitetural).

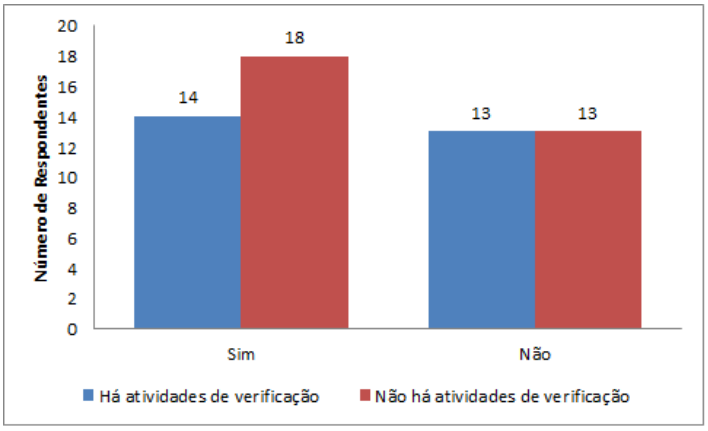

Figura 4.12: Já teve problemas devido a violações arquiteturais?

Com isso, temos indícios de que o surgimento ou não de problemas causados pelas violações arquiteturais não influenciam os profissionais a realizarem ou não atividades de verificação de conformidade arquitetural. O teste exato de Fisher foi executado para analisar esta questão. Com um *p-value* de 0.7919 não foi possível observar estatisticamente, para nossa amostra, se a diferença era significativa. Portanto, é necessário um estudo mais aprofundado, levando em consideração, por exemplo, a gravidade dos problemas que surgem para os dois grupos, para que possamos fazer afirmações mais completas.

Dentre os problemas causados pelas violações arquiteturais foram citados pelos respondentes a dificuldade de entender, manter, testar e evoluir o sistema, o grande acoplamento no código fonte, o surgimento de *bugs*, a dificuldade de identificar e solucionar os *bugs*, a baixa performance do sistema, o aumento da complexidade do código, e a má disseminação do conhecimento da arquitetura perante os membros da equipe.

4.3.4 RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

Por fim, com relação as ferramentas de apoio à verificação de conformidade arquitetural, 13 (48,14%) dos 27 respondentes que afirmaram ter alguma atividade desse nível, afirmaram utilizar alguma ferramenta de apoio. As ferramentas citadas pelos que as utilizam foram Fraunhofer SAVE, Sonar, SA4J, JDepend, Findbugs, Design Checker, DesignWizard, Eclipse API tooling, Visual Studio e Review Board.

Os 13 respondentes que utilizam ferramentas afirmaram ter um nível de satisfação entre "satisfeito" (nível 4) e "muito satisfeito" (nível 6) com elas. Porém, citaram como limitações os falsos positivos, a falta de documentação para o usuário da ferramenta, a dificuldade de integração com IDEs e a impossibilidade de integrar a ferramenta no fluxo de trabalho de integração contínua.

Para os 14 (51,86%) que afirmaram não utilizar nenhuma ferramenta de apoio para realizar atividades de verificação de conformidade arquitetural, ficam os questionamentos: como eles fazem tal atividade? As ferramentas existentes não são de grande utilidade para a indústria? Existe resistência por parte da indústria em utilizar ferramentas que automatizem processos relacionados a arquitetura do sistema?

Além disso, 9 respondentes dentre os 14 acima citados, afirmaram não conhecer nenhuma ferramenta. Para estes, ficam as perguntas: porque não conhecem? Seria falta de interesse? Seria falta de divulgação? As empresas não estão se preocupando com isso? Mais uma vez, apenas estudos aprofundados são capazes de responder essas perguntas.

Dentre as ferramentas conhecidas por quem não as utiliza foram citadas, Sonar e Visual Studio. São justificativas para a não utilização dessas ferramentas de apoio: a falta de necessidade; a empresa não definir uma ferramenta para utilização; não ser considerada essa

opção, pela equipe, nem ter sido realizada uma avaliação de prós e contras; auto-confiança por parte dos arquitetos (fazem a verificação manualmente); a falta de crença na possibilidade de automatizar essa atividade; a falta de uma ferramenta adequada; e, por fim, as ferramentas colocam o *software* como centro principal e os respondentes consideram que existem outras coisas, além do código, que compõem a arquitetura e que as ferramentas não capturam.

> Como falei, as verificações são mais informais e a arquitetura de componentes e regras arquiteturais não são, necessariamente, documentadas de forma a permitir o uso de ferramentas. Em resumo: não se considerou essa opção o suficiente e nem se avaliou os benefícios e contrapartidas. Acho que não vimos necessidade ainda, e por isso, não procuramos mais informações. Na verdade, os problemas de violação de arquitetura enfrentados até aqui não justificam o uso ou não foram sérios o suficiente para sentirmos essa necessidade. Acho que os controles em vigor foram suficientes até aqui (Respondente 10).

> The developers and architects are VERY self-confident, and very knowledgeable. They do it "by hand"(Respondente 28).

> Todas [as ferramentas] que foram levantadas pela nossa equipe coloca o software no centro de tudo. Nos nossos grandes sistemas o software compõe cerca de 40% do ativo que chamamos de solução (Respondente 52).

4.4 Considerações finais

Ao final desta primeira etapa de nosso estudo, temos uma noção, mesmo que breve e não aprofundada, sobre a visão da indústria com relação às nossas 4 questões de pesquisa.

Primeiramente, a nossa dificuldade de encontrar um profissional com o cargo de arquiteto de *software* dentro das empresas que procuramos nos chamou a atenção. Isto nos faz refletir, inicialmente, sobre quem é o responsável em desenvolver as atividades relacionadas à arquitetura de *software* nas empresas e como elas são realizadas (se forem realizadas).

Apenas com esta etapa, notamos que existem diferentes visões sobre o que é a arquitetura de um *software* (apesar da maioria concordar que os componentes e seus relacionamentos fazem parte de uma arquitetura). Na academia, isto reflete que, se é objetivo dela desenvolver pesquisas, nesta área, que sejam utilizadas na indústria, é necessário compreender esta abrangência de definições e aspectos e/ou definir um escopo e os tipos de empresas as quais suas pesquisas se dirigem.

Notamos também que a atividade de documentar a arquitetura possui baixa prioridade (por diferentes motivos) entre a maioria dos respondentes. Porém, mesmo que a arquitetura

não seja bem documentada, ainda assim, percebemos que existe algum tipo de documentação (informal, incompleta e/ou desatualizada). Além disso, documentos do tipo doc./docx. são bastante utilizados para registrar a arquitetura, assim como, também, a linguagem UML.

A atividade de verificação de conformidade arquitetural também nos pareceu ter baixa prioridade entre a maioria dos respondentes. Com esta etapa, tivemos a impressão de que essas atividades só serão prioridades nos processos de desenvolvimento de *software* quando existirem grandes problemas causados pelas violações arquiteturais.

Dentre os que realizam alguma atividade nesse sentido, mais da metade a realiza de forma manual. Um dos motivos é que as ferramentas de apoio a essas atividades são pouco difundidas e/ou conhecidas. Além disso, fazemos uma relação com a forma mais popular dos profissionais documentarem a arquitetura. O objetivo da maioria dessas ferramentas é automatizar o processo de verificação. Conjecturamos que quando se documenta a arquitetura com linguagem natural ou UML torna-se difícil a automatização dessa atividade utilizando alguma ferramenta (o que justifica, também, a taxa de utilização).

4.5 Ameaças à validade

Como ameaças à validade desta etapa temos, inicialmente, a ameaça interna com relação à escolha dos candidatos a responder este questionário. Nosso objetivo inicial era fixar nossos respondentes em profissionais com o cargo de arquiteto de *software*. Porém, quando procuramos profissionais com este cargo nos deparamos bastante com a frase "não temos este cargo aqui". A partir disso, tentamos separar profissionais da forma mais diversificada possível, procurando os *core developers* de sistemas e empresas conhecidas.

Temos, ainda, a ameaça de constructo tanto na elaboração do questionário quanto na análise das respostas. Na elaboração temos a ameaça de que nossas perguntas podem não capturar toda a riqueza de informações do ambiente em questão (desenvolvimento de sistemas reais). Na análise temos a ameaça da expectativa do pesquisador, uma vez que nosso conhecimento prévio da área pode ter causado um viés na análise. Para tentar contornar estas ameaças, todo o questionário foi revisado por mais 2 pesquisadores. Como exemplo da ameaça a elaboração do questionário temos 2 comentários feitos por candidatos/respondentes:

> Acredito que a pesquisa seja mais voltada para sistemas de informação. É importante observar como se trabalhar com arquitetura de software para sistemas embar-

> cados, principalmente no nível crítico. Por exemplo, regulação afeta diretamente o design (Respondente 19).
>
> Lá na minha área da empresa, não há no processo a checagem se a arquitetura projetada está de acordo com o que foi projetado. A gente trabalha com a linguagem de programação ABAP, porém a maior parte do desenvolvimento é em uma ferramenta de modelagem de datawarehousing chamada BW, da empresa SAP. Tentei responder o formulário nas duas vezes que você mandou, porém não vi como responder adequadamente (Comentarista 2 - via e-mail).

Por fim, temos a ameaça externa uma vez que não é possível generalizar os resultados de nosso estudo, apesar de termos representantes de diferentes linguagens de programação, diferentes países e empresas.

Capítulo 5

Etapa 2: Entrevistas

A segunda etapa deste estudo consistiu de entrevistas com os profissionais da área de desenvolvimento de sistemas reais. Com esta etapa, temos como principal objetivo entender, de uma forma mais aprofundada que a etapa anterior, o ambiente de desenvolvimento de sistemas reais no contexto de arquitetura de *software*, documentação arquitetural, verificação de conformidade arquitetural e as ferramentas de apoio que a indústria tem conhecimento. Com esta etapa entendemos melhor como o contexto de arquitetura de *software* está inserido no desenvolvimento de sistemas reais. Neste capítulo, apresentaremos o planejamento desta etapa, assim como também sua execução, resultados, análise e ameaças à validade.

5.1 Planejamento

Planejamos, em média, treze perguntas para os entrevistados. As perguntas envolviam assuntos relacionados à experiência e opinião dos arquitetos de *software* entrevistados com relação à definição de arquitetura, documentação arquitetural, verificação de conformidade arquitetural, violações arquiteturais e ferramentas de apoio à verificação de conformidade arquitetural. Planejamos as perguntas com o objetivo de responder as quatro questões de pesquisas e alguns questionamentos que surgiram da etapa anterior. O nosso roteiro de entrevista pode ser visto no Apêndice B.

Para selecionar os participantes desta etapa, eliminamos o primeiro quartil (25%) menos experiente dos respondentes que declararam ter sido arquiteto de *software* no *survey* exploratório (primeira etapa). Após isso, enviamos 35 *e-mails* personalizados convidando para

a fase de entrevista. Destes, 20 responderam se voluntariando para participar da entrevista (taxa de 57,15%). Dos 20, 2 só poderiam participar por *e-mail* (o que não era nosso objetivo) e 4 deixaram de responder os *e-mails* depois de um certo tempo (impossibilitando o agendamento da entrevista). Portanto, entrevistamos 14 voluntários (taxa de resposta de, aproximadamente, 40%).

Para analisar as entrevistas e suas transcrições utilizamos a Análise Temática baseada em Braun & Clarke [2]. Segundo Braun & Clarke, a Análise Temática é um *"método para identificar, analisar e reportar padrões contidos nos dados"*. Esta abordagem de análise foi detalhada no Capítulo 2.

5.2 Execução

Realizamos as entrevistas entre 05/11/2014 e 28/11/2014. Elas tiveram duração média de 26 minutos (onde a entrevista mais duradoura foi de 46 minutos e a menos duradoura foi de 12 minutos e 37 segundos). Utilizamos *Hangout* ou *Skype* (à escolha dos entrevistados) para realizar as entrevistas e apenas uma delas foi feita de forma presencial. Gravamos todas as entrevistas e as transcrevemos para codificar e analisar.

Os entrevistados tinham experiência média de 7 anos como arquitetos de *software* e trabalhavam em diversos tamanhos de empresa nos países Brasil, Argentina, Uruguai, Estados Unidos da América, Alemanha, Áustria, Hungria e Reino Unido.

Dos 14 voluntários a realizarem entrevistas, 7 afirmaram, no *survey* exploratório, ter uma arquitetura bem documentada, 6 afirmaram realizar atividades de verificação de conformidade arquitetural e 3 afirmaram utilizar ferramentas de apoio a esse tipo de atividade (Tabela 5.1).

5.2.1 Codificação

Para a codificação utilizamos a ferramenta nVivo 10 [1], um *software* que permite a reunião e organização de diferentes tipos de dados, facilitando a análise do conteúdo pelo pesquisador.

Ao final da análise, chegamos a 207 códigos separados em 5 grandes temas: Arquitetura de *Software* no geral, Documentação arquitetural, Verificação de conformidade arquitetural,

[1] http://www.qsrinternational.com/products_nvivo.aspx. Último acesso em 19/08/2015.

Tabela 5.1: Resumo sobre os voluntários da entrevista com relação ao *survey* exploratório.

Entrevistado	**Possui arquitetura bem documentada**	**Realiza atividade de verificação**	**Utiliza ferramentas de apoio a verificação**
E1	x		
E2	x	x	x
E3	x		
E4		x	
E5		x	x
E6	x	x	
E7		x	
E8	x	x	x
E9			
E10	x		
E11			
E12			
E13	x		
E14			

Violações arquiteturais e Ferramentas de apoio à verificação de conformidade arquitetural. Os gráficos com a hierarquia dos códigos se encontram no Apêndice C.

Os códigos sobre os dados dos entrevistados são chamados códigos de contexto e não entram na análise temática explicitamente. Eles são apenas para contextualizar o estudo, fornecendo informações sobre os entrevistados.

Na Figura 5.1 é possível visualizar os principais temas e subtemas encontrados na análise final.

5.3 Resultados e Análises

Nesta seção, apresentamos nossos resultados e análises decompondo-os de acordo com as questões de pesquisa, visando facilitar a leitura dos resultados.

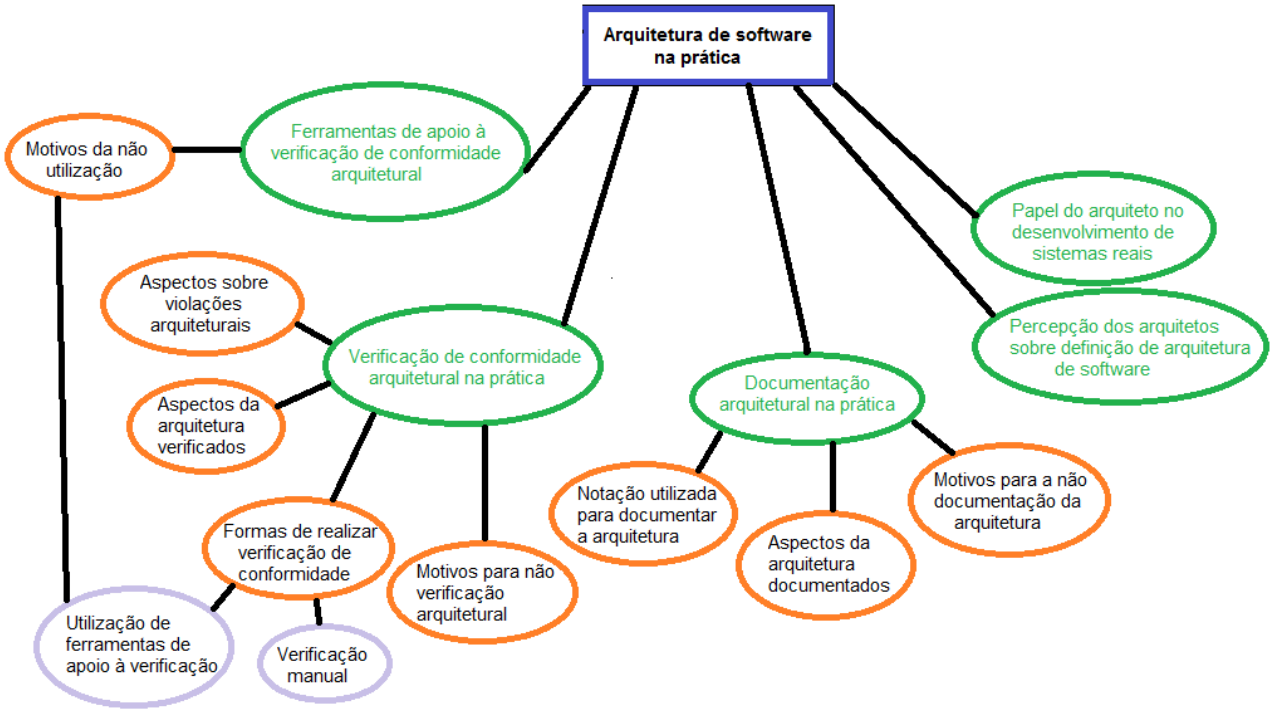

Figura 5.1: Temas e subtemas.

5.3.1 RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

Após o *survey* exploratório e as 14 entrevistas, concluímos que não há um consenso sobre a definição de arquitetura de *software* na indústria (bem como não há na academia). Os aspectos arquiteturais citados pelos entrevistados, assim como no *survey* exploratório, foram diversos: aspectos tecnológicos, de negócio, dependências entre componentes, aspectos de *hardware*. Não notamos relação entre localização, tamanho da empresa e tempo de experiência com relação aos aspectos que cada profissional considerava como parte da arquitetura. Apesar da falta de consenso, podemos afirmar que arquitetura de *software* pode ser resumida como um conjunto de decisões importantes para o sistema (seja de qual âmbito for).

> Well, for me it's a design of which technologies and patterns you will follow to accomplish or how you will solve a problem (E1).
>
> You know there is no common definition for software architecture (E4).
>
> Eu tento fugir um pouquinho das definições clássicas do SEI (Instituto de Engenharia de Software de Carnegie-Mellon), como essa dos componentes e conectores, porque eu acredito que é uma visão muito tecnicista e ela não reflete o papel que uma arquitetura tem na prática que a gente convive. [...] Eu gosto da definição

> do Eoin Woods que diz que a arquitetura é um conjunto de decisões que, se feitas incorretamente, podem levar um projeto ao cancelamento. Então eu gosto dessa definição, porque ela é simples e mostra a importância estratégica das decisões arquiteturais ao longo de um projeto. [...] a arquitetura tem que estar alinhada aos condutores de negócio e para que ela possa realizar valor de negócio. Então, sem dúvida, que ela vai desembocar em componentes, em conectores, em boas relações, boas abstrações, boas implementações de mecanismos arquiteturais. Mas isso é inútil se ela não observa o contexto (E10).

> So... Software Architecture is the high level view of the components of software systems and their interactions (E12).

Assim como no *survey* exploratório, nesta etapa também percebemos que, na prática, apesar de não haver um consenso na definição de arquitetura de *software*, a maioria dos profissionais acredita que componentes e seus relacionamentos fazem parte da definição de arquitetura de *software*.

> I mean, architecture is, plus a lot of things, practices. I mean, the simplest definition is 'how to put pieces together'. That is, if I had to say something, some definition, I believe that is the right one (E4).

> é um conjunto documentado de decisões, que servem para modelar a forma como os componentes de um sistema interagem entre si (E5).

> So... Software Architecture is the high level view of the components of software systems and their interactions (E12).

> É definir os sistemas, em termos de subcomponentes, subsistemas e estruturas de dados, e a comunicação entre os sistemas (E14).

Além disso, apesar de não notar relação entre tempo de experiência e tamanho da empresa com relação à definição dada pelos profissionais, percebemos que a definição de arquitetura de *software*, na prática de um profissional, depende do objetivo e das experiências vividas por ele, assim como também do projeto a ser desenvolvido e da cultura da empresa a qual está inserido.

> [...] It [definition] depends of what you are talking about, I mean, if you are architecting an application or a solution of a problem, just that (E1).

> My definition is that software architecture is the overall structure of a system. It doesn't matter if the system is still in design, if it's only an idea or if it's already in the field. Is overall structure of a system combined with the technical and strategic choices made for the system combined with the intention of the user, or of the client. So it consists of three things: system structure, choices, (strategic technical sometimes even commercial, organizational, whatever) and, third, intention or needs of the user (E7).

> Então, a arquitetura pra mim, ela é uma estratégia técnica que tem que ser desenvolvida de uma forma contextual e muito inserida, muito não, completamente inserida na realidade de cada projeto que um arquiteto trabalha (E10).

> Arquiteto de ... não tem exatamente, assim, muito bem definido o que um arquiteto de computação, de sistema, deve ser aqui no Brasil. Em cada empresa é uma coisa diferente. Certo? Sem dúvida, a implantação impactou bastante na arquitetura, mas não necessariamente todo arquiteto com quem eu trabalhei, ou nas empresas que eu já vi, se importa com regras de negócio (E14).

Achamos interessante citar que alguns profissionais afirmaram que as decisões sobre a tecnologia que será utilizada são importantes, mas elas não devem ser priorizadas para tomar decisões arquiteturais. Além disso, um respondente afirmou que o domínio do problema e sua solução devem ser consideradas como o centro de toda a arquitetura.

> Seria como, por exemplo, como se tu fosse escolher um carro, certo? Eu quero a eficiência de consumo de combustível, no entanto, tem que ter 4 portas, 4 lugares, isso seria uma boa especificação. Uma péssima especificação não funcional, seria, tem que ser da GM, modelo Corsa (E6).

> As vezes a gente tem uma tendência, principalmente o pessoal desenvolvedor que tem um cunho mais técnico assim, de pensar logo na tecnologia. [...] a tecnologia não pode vir em primeiro lugar. Muito embora ela seja muito importante porque a gente está falando de TI. [...] o domínio do problema é a primeira coisa a ser compreendida quando você vai desenhar a arquitetura. [...] Você tem que tentar resolver da melhor forma e [...] o entendimento do problema tem que estar no centro (E13).

5.3.2 RQ2: Qual a visão da indústria com relação à documentação arquitetural?

Com relação à documentação arquitetural, reforçamos, com essa etapa, nossa suspeita de que, na maioria dos projetos, existe uma documentação arquitetural, mesmo que seja simples, incompleta e/ou desatualizada. Talvez tenhamos nos equivocado ao questionar, no *survey* exploratório, quem possuía uma arquitetura bem documentada. Talvez devêssemos ter adicionado uma pergunta anterior ("Você possui um documento arquitetural?"), pois ter uma documentação e ter uma boa documentação são afirmações diferentes.

> Documentada sempre é, por mais displicente que seja o profissional ou a empresa, uma questão até cultural, ainda assim você vai ter um nível de documentação. Por mais rudimentar que seja a ferramenta que você esteja usando, a própria ferramenta às vezes te dá a capacidade de documentar com o próprio código, por exemplo: JavaDocs, enfim. O que eu questiono não é a existência ou não do documento é a qualidade do documento (E10).

Os motivos apresentados, pelos entrevistados, para a displicência com a documentação arquitetural foram, praticamente, os mesmos apresentados na etapa anterior: prioridade em

outras atividades, falta de tempo, falta de crença em documentação, sentimento de falta de necessidade. O motivo "falta de necessidade"foi exposto no contexto de metodologias ágeis, ou quando a equipe é pequena, ou quando o sistema é trivial, ou quando os sistemas são opensources. Além disso, percebemos, a partir das entrevistas, que documentação arquitetural é considerada uma atividade entediante e com baixa prioridade.

> E também não senti muita falta, porque primeiro era uma coisa trivial [...] tinha camadas e ba ba ba, então tinha que ser trivial porque o tempo não permitia algo mais elaborado (E3).

> We have meetings, daily meetings, "What I did yesterday. What I am doing today. What I have to do. What stops I have."But that's not every day, just, every two or three days. You know, the roles, we don't comply with the roles of agile methodology. Because pure agile methodologies have one problem, they don't consider architecture important, just refactor, they say. They focus only on functional requirements. So there's not much place for an architect in a pure agile methodology [...] the most senior developers know [the architecture], semi-seniors know half of the rules and juniors don't know almost any rule. But we work together, we work with small teams. We don't have big teams, we have small projects. We have now a lot of constructors that help, we have micro services architecture along the whole company. So now in working in a team, we are only three, and we are taking two projects, two different applications, actually that are part of a bigger stack of application. And being only three, you can talk to the other person, you can sit next to him and so, "Ok! Look at this"We can make better programing, we can just make a few drawings in a white board, just with the major guidelines. But nothing important, just drawings, actually, and no diagrams or UML, just conceptual basic drawings, maybe to clarify some points, some ideas, but only that (E4).

> Não sei, é questão talvez de política das empresas onde trabalhei ou questão de querer pegar o cliente de todo jeito e entregar em prazo curto demais. Aconteceu, eu tive muitos problemas de desenvolvimento com entregas porque eram, vamos dizer assim, atividades muito extensas para se fazer e queriam que fossem feitas em tempo muito menor, então eles saiam eliminando várias etapas, como por exemplo essa de planejar, de planejar arquitetura etc. a análise era meio que feita na hora (E11).

> The reason why I stated that in the survey is that now I'm currently working on an open source software, which it's documentation is terrible. And this is the reason why the open source software system I am involved in at the moment is a mess (E12).

> Primeiramente, sempre se tem mais trabalho a se fazer do que tempo, tempo é um recurso bem caro [...] documentar é muito chato e as pessoas não gostam disso, elas não vêem isso como algo produtivo ou algo de valor, que nem codificar, então isso sempre acaba sendo a última prioridade e competindo com aquelas coisas que precisam de mais tempo (E14).

Com as entrevistas, reforçamos nossa suspeita de que cada empresa tem sua forma e notações para documentar a arquitetura. Normalmente, são usados para documentar a arquitetura: ADLs, desenhos, diagramas, tabelas, *wikis*, UML, ArchiMate, texto em linguagem

natural, assim como podem se basear em padrões e métodos como CMMI. Contudo, percebemos que o texto em linguagem natural foi a forma mais popular para documentar a arquitetura entre os nossos entrevistados.

> When I was younger, I used to use formal specification for software architecture documentation, like UML, for instance. I found out anyway that in the industry it really depends in which component or, let's say, industry sector you are working on. Every company has a different standard (E12).

Descobrimos, ainda, que, em muitos casos, os documentos arquiteturais não devem ser muito rigorosos e extensos. Porém, não percebemos um consenso entre os entrevistados sobre o que é preciso documentar da arquitetura do sistema. Foram citados como importantes para documentar: aspectos funcionais, aspectos não-funcionais, componentes e seus relacionamentos, aspectos de *deployment*, as tecnologias utilizadas e todas as decisões com suas respectivas justificativas.

> That depends on the role. For example, requirements engineer will have to document his requirements, requirements have reason to exist. Why do we have this requirement? So I want documentation about that. I want the developers and testers to produce a testing manual. I want someone, whoever, to produce a user manual. Could be a tester, for example, a technical documentation writer. I want the lead engineer to keep very detailed trace of the technical choices made. Why did he make this technical choice? And what constraints? And what conditions? I want management to document the financial decisions being made (E7).

> In long systems I document also everything. From the ideas to the design decisions, and ideas about why I choose a certain way. It's very useful on the long run. When I check my systems, when I have to do some refactor, when I have to add new functionality (E9).

> Naturalmente eu acredito que a gente não deva ser dogmático com a documentação, então eu vou te dar novamente exemplos. Quando você olha pra escola do SEI, eles propõem documentos extremamente rigorosos e pesados, mas que são impraticáveis pra 90% da realidade que eu conheço do mercado, aqui do sudeste, Belo Horizonte, São Paulo, Brasília, Rio, onde a gente tem um contato maior com profissionais. Então as empresas não tem tempo pra escrever tanto documento e muito menos manter aquele nível de documentação. Então, novamente, eu acredito que o nível da documentação deva ser bastante contextual de acordo com o rigor do projeto, o tamanho, pra que ele possa ser aplicado. É preferível você ter algum bom documento, do que uma promessa de um documento gigante, que na prática não se realiza [...] Então né, naturalmente, requisitos arquiteturais fazem parte também do documento, isso é importante, e uma documentação, que eu vejo que é fundamental, que é o racional do arquiteto. Ou seja, "por que que eu escolhi java?", "por que que eu escolhi C#?", "por que que eu escolhi Oracle nesse projeto?", então a gente entende que mais importante do que a decisão, é a racionalização de uma decisão. Por que? Nas empresas, é muito comum que as pessoas façam julgamentos sobre decisões do passado, então alguém faz um julgamento "olha, porque que lá em 2010 né, essa pessoa aqui escolheu Oracle? Oracle não faz nenhum sentido na nossa empresa", então, é fácil ser profeta do passado, e

> isso acontece porque as pessoas não documentam o racional das decisões, então o racional arquitetural, que também é uma escola que a gente busca até de outros autores, de outros autores europeus também, com bastante ênfase nessa questão, é você descrever o processo que levou você a tomar uma certa decisão arquitetural em um projeto, então isso também vem a reboque (E10).

Achamos interessante que alguns entrevistados consideram que a documentação arquitetural deveria ser escrita numa linguagem apropriada para quem vai lê-la, ou seja, a documentação arquitetural deve ser escrita pensando no perfil de cada pessoa envolvida na produção do sistema, desde cliente até desenvolvedor. Há, ainda, os que afirmaram acreditar que a documentação arquitetural não deve ser separada do código fonte do *software*.

> I consider it important that you can somehow see the architecture, and ideally not separate from the actual code or the thing that gets executed. Because if you write documentation about it, that's going to be out of date as soon as you make the next change. Though, I would consider it important that the architecture is visible, but that it's not in separate documentation (E2).

> O que eu quero dizer é que os detalhes da arquitetura precisam estar evidentes no código. Você não deveria precisar, como desenvolvedora, de documentação para entender da estruturação de um determinado código (E6).

> Eu tenho que ter uma documentação com o ponto de vista que me remeta a pessoas com menos conhecimento na área, por exemplo: eu tenho um stakeholder, um cliente de um nível, talvez ele não entenda..., que não seja de um nível técnico certo, então eu tenho que ter uma documentação que demostre isso, que passe segurança pra meu cliente que ele entenda o que eu tô fazendo, então eu tenho que ter uma documentação, tenho que ter documentações diferentes, com níveis de linguagem diferentes (E11).

Notamos que a maioria dos entrevistados considera importante documentar a arquitetura, porém, afirmaram também que tão importante quanto documentar é a habilidade do arquiteto em transmitir as suas decisões para a equipe. A documentação arquitetural pode ajudar, segundo os entrevistados, na comunicação (principalmente em empresas grandes), no entendimento do sistema, na manutenção do sistema, evitar a perda de conhecimento sobre o *software* e guardar as decisões e suas justificativas.

> Yes, I think it's important because every time you add new developers they have to understand why you choose that architecture, what were you thinking when you choose that path (E1).

> And I do consider important to teach junior developers, to teach them all you know, to guide them (E4).

> Para diminuir a necessidade da conversa ponto a ponto da transmissão do conhecimento ponto a ponto, se uma equipe é grande você não consegue transmitir todo conhecimento que tem conversando um por um [...] também a medida que o tempo passa o conhecimento vai se dissipando em algum momento que for manter aquele sistema pode ter dificuldade de entender como ele foi modelado. Então é tão importante quanto documentar os próprios requisitos (E5).

> Most of the time when you design or build a system you have a certain lifecycle in your head. It's going to live for certain time. And in order to do that you're going to invest a certain amount of resources. Most of the time it will be money, but it can also be engineering hours, it can be hardware, it can be effort from other people, up to marketing, for example. And protecting that investments can best be done by ongoing documentation. If you don't document... I'm myself very much into open-source, but if you don't document at all the only documentation you're going to have is code. And you see that in a lot of open-source projects where there's nothing, there's no model, there're no requirements, there're no, hardly, any test results, you know, design papers, and you see many, many, open-source projects simply dying, after certain time, that could have lived for a long time. So the longer your system has to live, and in also extreme cases where systems were designed to live for forty or fifty years, the more you need documentation (E7).

> Absolutely, yes. Because when we have a documentation of an architecture, we have two things at the same time: we have the design documentation that we want to follow during the implementation and also we have to document the reasoning behind all these figures. So when we have an architecture that is the subset of ideas that we tried and we have selected some that we decided that we should implement things in this way and not another way. I think it is very important to document both the best solution that we want to implement, and the reasoning why we selected a certain solution [...] Later on, when we have to change something in the system or when we have to adapt new requirements we will be able to reuse this information to that documents (E9).

> Yes, absolutely [it is important]. Because if you don't document the architecture of your software, when you go to the other phases of software development, a lot of things tend to get lost in translation. And this implies that if you do a check point, for instance, of a software that is really big implemented, compared with the architecture that has been designed for the same software, you may end up finding out that the actual implementation is really different from the architecture that you designed. So, for this reason, is really important to document, even in a really pedant way, the architecture of a software (E12).

Ficamos surpresos com o relato do entrevistado E4 que afirmou não existir modificações na arquitetura durante o desenvolvimento de seu sistema. Em nossa opinião, seria comum haver modificações na arquitetura, uma vez que é natural a evolução dos requisitos e conceitos envolvidos em torno do sistema.

> Well, we don't change the architecture on the fly. So, we make changes just in the beginning of the project. If the architecture changes we have to do a refactor or maybe we just do a new project. I mean, we need to start a new project, do it again (E4).

Percebemos que, normalmente, a documentação arquitetural não é atualizada. Quando é preciso modificar a arquitetura há conversas com toda a equipe sobre o assunto e quando acontecem realmente as modificações, elas são comunicadas através de *e-mail*, reuniões, conversas informais, documentações ou apenas no código.

> First of all, when I feel that we need that change of the architecture, I like to discuss with the members (E1).

> A documentação não era muito bem atualizada onde trabalhei, depois que eu adotei essa prática de documentação arquitetural eu tinha muito trabalho pra atualizar esses documentos é tanto que esse documento ele nunca refletia em tempo real a situação do projeto. Então eu tinha, esse documento normalmente tinha em média mais ou menos 15 dias no melhor caso, um mês no pior caso de delay (E3).

> A sua própria arquitetura é uma coisa emergente. Então a questão de documentação é a dificuldade de se manter a sincronia entre o que você tem de fato no momento e o que você documentou no início. No início você tenta fazer uma modelagem um pouco mais abrangente, mais genérica, mas em algum momento você vai ter que fazer um detalhamento daquilo tudo e esse detalhamento é tão volátil quanto o próprio código. E manter isso é muito complicado [...] Acho que o primeiro ponto que eu questiono é a volatividade da documentação porque o software você não consegue modelar ele 100% no início, porque os requisitos não são modelados no início. A gente sabe que o ciclo interativo incremental ele prega que você modele aos poucos. Você vai descobrindo os problemas aos poucos modelando aos poucos, implementando aos poucos e isso faz com que decisões que você tomou no passado tenham que ser revistas ao longo do processo (E5).

> Isso é um dilema. Porque normalmente no nosso processo, inclusive lá na empresa a gente tem trabalhado muito como fazer retro-alimentar esses documentos que, às vezes, realmente por mais que você tenha um processo ele tende a ficar defasado né (E13).

5.3.3 RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

Sobre verificação de conformidade arquitetural, percebemos que é consenso, entre os entrevistados, que esta é uma atividade que checa se o sistema está sendo desenvolvido de acordo com as decisões arquiteturais. Esta definição é a mesma utilizada por Clements *et al.* [9], na academia.

> But my definition is: it's a check that you perform periodically to verify that the designed architecture of your software system actually matches what's being implemented (E12).

Para os entrevistados, é importante que a equipe tenha a mesma visão arquitetural do sistema. Em nosso trabalho anterior [27], tratamos sobre esse mesmo tema, mostrando a importância de se preocupar com os diferentes níveis de abstração/conhecimento para uma mesma arquitetura. Para o desenvolvimento de um sistema, não existe, normalmente, apenas a equipe de desenvolvimento e, portanto, é importante que todas as equipes tenham a mesma visão da arquitetura.

> And I find that very important to make sure the team is on the same page [...] It should be on a regular basis. Everytime someone does a checking there should be feedback: 'oh, your checking was good' or 'your checking was bad'. Should be a feedback from the computers, and should also be feedback from the people in the team, the architect team person to be overseen that but it's responsibility of the whole team to be constantly checking that (E8).

Percebemos, também, que a realização ou não de atividades de verificação de conformidade arquitetural (assim como a periodicidade da checagem) depende do quanto a empresa/arquiteto considera necessário. Em desenvolvimentos de sistemas críticos, normalmente, há um processo voltado para verificação de conformidade arquitetural. Enquanto isso, em pequenas empresas, realizar a atividade de verificação de conformidade arquitetural ou não depende do arquiteto de *software* ou do desenvolvedor *senior*.

> It's depending of the needs. Sometimes I don't need to do that. Because most of the process is just a refactor of something that got... I mean, when you work in a software factory sometimes what you do is just refactor stuff or just pieces of software that you build for another clients, and you have to refactor that piece in something else. So that's why we don't make this conformance checking. That's not what I do daily. That's why I said... in the business that I work I don't do much [...] One part is to check before the deploy. In the industry that I work we like to document before doing a deploy, ... So we can check if the architecture is being followed in what we have designed (E1).

> Nos últimos projetos não tenho usado mas não tenho usado não porque não gosto, porque a equipe era menor, então era mais fácil de gerenciar a coisa, não sentia necessidade. Mas, enfim, não é uma coisa que eu abro mão sempre não (E5).

> Or in smaller companies it can be skill. In smaller companies, architecture conformance checking is skill of the software architect. The software architect has an intention, he has a goal he wants to reach, and if he is clever, he is not going to spread out all of the goal and all of the intention at once. He keeps it in his head, and he constantly checks the work done by people who implement the architecture against his goal. Against his intention [...] In large companies you have a process for that. Large companies, specially in Aerospace you have standards, specification documents that proscribe, that impose certain architectural style. And you also have a process to make sure that your still in conformance with standard specifications. So architectural conformance checking in a large company very often is checking the respect of standards (E7).

Ainda analisando as respostas, percebemos que alguns arquitetos não se preocupam em checar se a implementação está em conformidade com a arquitetura. Alguns consideram que é impossível calcular a arquitetura e aplicar um algoritmo para checá-la completamente. Já outros, consideram que a verificação de conformidade arquitetural é sim importante, porém, não é uma atividade suficiente para garantir a qualidade do código.

> Yes [important to verify], I mean, it's part of, of course, but it's not the silver bullet (E1).

> Yes. It's one important thing to do [verification], but it's not sufficient [...] Once you adhere to these rules, it's not automatic, it doesn't mean that your software will be of good quality and will have no bugs. It's just one small part of the overall problems that we're facing in software development (E2).

> Anyway, I believe that architecture cannot be 'calculated', or best said, I think that it is not factible to apply any algorithm or suite of algorithms and techniques to verify architecture compliance (E4).

Observamos, ainda, em nossa análise, que alguns aspectos estruturais não são facilmente capturados numa verificação de conformidade arquitetural. Os aspectos que os arquitetos normalmente tentam verificar, segundo os entrevistados, são as interfaces definidas na arquitetura (componentes e seus relacionamentos) e aspectos não funcionais.

> In practice, all these things are much more complicated than in theory, because there are dependencies that you don't see very easily in the source code, and you don't see them in the build files, right? So if module A calls module B you see the dependence easy, if module E makes an RPC call to a server that uses module B, then that RPC call is something that you can still see somewhere but it's not as easy to know. And then there are more complicated dependencies, like when you have a published subscribe system where one component publishes events and another component subscribes to these events and then it gets harder (E2).

Nas entrevistas vimos que não é muito comum utilizar ferramentas de apoio à verificação de conformidade arquitetural (depende do quanto a empresa/arquiteto considera necessário). Ao invés disso, a atividade de verificação é, normalmente, feita manualmente e, esta forma, é considerada confiável para os arquitetos de *software*.

> My software I said, I already told you, that this checking is kind of crafty. I mean, it's made by hand. I don't use any tool, because tools are not flexible (E4).

> I'm very much a pupil of two older architects who did the architecture of the Apollo space program in the 60's and the 70's. And they came up with a couple of heuristics that are almost always valid. You know, systems architecture conditions. And one of them is "the human eye is a fine architect, trust it". I think that the eye of a very experienced software architect, most of the time, can and will catch most of the violations. And that trusting 100% in a tool to catch all the violations, first is going to be very costly, is going to cost you a huge amount of resources and money, and second, I'm not sure if it can be done, if a tool can really catch all the violations. So I would rather trust the human eye of a very experienced software architect and a tool than only a tool. When I was 10 or 20 years younger, I was more, let's call this, techno-optimistic and I also thought that tools can catch everything. The experience is that when you grow a bit older, when you have 30 or 20 years of experience in this prospection you also grow little bit more conservative and a little bit more pessimistic about technology. So that's how my answer was inspired (E7).

Para alguns, uma forma de garantir que a arquitetura definida será seguida é definir interfaces (API) para os componentes, de forma que estes só serão acessíveis através dos métodos

fornecidos. Assim, os testes de integração e de unidades são considerados, por eles, meios de verificar a arquitetura do sistema.

> Bom, aí no momento que tu tem a integração dos módulos, eles têm que funcionar, porque as APIs têm que ser respeitadas, essa é a, digamos assim, dentro de um determinado módulo, essa é a obrigação dele, respeitar o que foi acordado na especificação da API [...] Assim, existem alguns níveis de verificação. O primeiro é internamente, no código, são testes unitários e, isso é o que é a responsabilidade do desenvolvedor, fazer os testes unitários, uma vez que os módulos são entregues, existe uma parte de testes de aceitação automatizados, que se roda a cada build do sistema, onde todos os módulos são sincronizados, são construídos e são colocados pra rodar, e aí, rodam os testes de aceitação (E6).

> I think that the architecture conformance, on this level, on implementation, can be easily solved in an object oriented system, like Java. So if we say that we have separate components, we can create interfaces and constant definitions, enumerations, and this kind of things to safely expose the public part of the implementation [...] So I think, in a objector system, if the features are used properly, then most of the conformance can be create in the language itself, in the source code of that, with interface, abstract class and assertion, using properly these things (E9).

Ainda, tivemos relatos de entrevistados que acreditam que deve-se implantar uma cultura, dentro das empresas, onde os desenvolvedores se sintam responsáveis por seus erros. Ou seja, ao invés de desenvolver de qualquer jeito, inserir o erro e depois criar um processo para corrigí-lo, deve-se prezar pela qualidade do código e corrigir o erro assim que detectá-lo.

> Então a nossa filosofia é que todos sejam plenamente responsáveis pelas questões, e tão logo uma não conformidade surja, essa pessoa ela tem que resolver aquele ponto ao invés de você trazer a cultura do 'vou fazer errado e depois vou fazer um processo formal de conformidade pra poder concertar o erro que eu mesmo produzi no projeto'. Então essa é uma visão clássica de engenharia de software que a gente na nossa empresa, não acredita e não promove, a gente acredita numa visão muito mais orgânica e que enfatiza que a qualidade deve ser um processo contínuo, e não um processo que ocorra ao final do processo produtivo (E10).

Os motivos citados pelos entrevistados para não realizar verificação de conformidade arquitetural, assim como na etapa anterior, envolvem motivação, a falta de uma arquitetura bem definida, a falta de necessidade, a falta de um responsável pela atividade e tempo.

> It's depending of the needs. Sometimes I don't need to do that (E1).

> So at the end of the day you can't do any sort of conformance checking because you don't have a reference architecture to compare it against (E12).

> [...] a profissão, o papel do arquiteto não é muito bem definido. E então isso não aumenta na responsabilidade dele. Não está certo e definido o que, quem tem que fazer isso ou que isso tem que ser feito. Desde que foi definido arquiteturas, isso foi implementado de acordo com a arquitetura, e passou pela qualificação, tendo a correção de defeito e então, está tudo certo (E14).

5.3.4 RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

Com relação às ferramentas de apoio à verificação de conformidade arquitetural, percebemos que elas não são muito conhecidas, nem usadas. Um dos motivos para o não conhecimento, na nossa opinião e análise das respostas, é que parece existir uma falta de interesse em encontrar essas ferramentas ou percepção de que tais ferramentas podem existir. Sonar, Enterprise Architect e Rational Software Architect foram as ferramentas citadas pelos únicos 3 entrevistados que afirmaram utilizá-las.

> Maybe for big companies, but not for everyone. I think they are more like underground stuff. I mean, maybe big companies have their own tools, not for common people neither software factories (E1).
>
> Eu nem fazia idéia naquela época que eu podia contar com ferramentas que me auxiliassem nisso [...] São bem pouco conhecidas, só no contexto de quem realmente trabalha com os sistemas críticos, sistemas embarcados é que são bem popularizados (E3).
>
> Olha, eu posso te dar minha experiência, eu nunca utilizei isso (E6).
>
> Agree... Yes, agree. Software architect know them but, as I said, much technical personal, even senior engineers, senior team leads, for example, very often either don't know them or are simply resistant against using them. They see them as a waste of time. So yeah, I agree (E7).
>
> Ok, I don't know any tool because I've never tried to find it (E4).
>
> Sem dúvidas. É porque, na verdade, a indústria conhece muito poucas coisas, a indústria trabalha em parâmetros de qualidade muito baixos, então, eu acho que é uma questão latente de engenharia de software. Se você vai pro mercado e começa a observar o que o mercado pratica de engenharia de software, pratica muito pouco. Em arquitetura menos ainda (E10).
>
> It may be my ignorance, but if I have never heard of any of these tools then probably yes. I agree. But it may just be me being completely ignorant (E12).
>
> Eu, até agora, nunca tinha ouvido falar em verificação de conformidade arquitetural, por exemplo, então isso não é muito falado na indústria. De repente, só em sistemas realmente críticos (E14).

Dentre os motivos para a não utilização foram citados: as ferramentas existentes não conseguem capturar todos os aspectos importantes da arquitetura do sistema; as ferramentas não são tão fáceis de utilizar; não há muita flexibilidade nas ferramentas; existe, em alguns, a descrença de que é possível construir um algoritmo que cheque a arquitetura; muitos acham que não há necessidade de utilizar essas ferramentas, principalmente se a equipe for pequena;

a própria falta de conhecimento das ferramentas; a preferência em utilizar a forma manual de checagem; muitos profissionais dizem não gostar de utilizar ferramentas de verificação de conformidade arquitetural; algumas ferramentas requerem que a documentação arquitetural esteja bem definida e, por fim, alguns citaram o tempo que se perde para configurar uma ferramenta desse tipo.

> That's no automatable... you cannot build a software to comply to architect. Because it depends on the application, it depends on the human positions; it depends on the business, so I don't believe there's a tool for that! It has to be made by humans, just by thinking, comparing, Trial and Error, that thing you learn every day, so you have to learn [...] Anyway, I believe that architecture cannot be 'calculated', or best said, I think that it is not factible to apply any algorithm or suite of algorithms and techniques to verify architecture compliance. Architecture cannot be abstracted to that level; it's just about industry trendings, personal like/dislike about certain technology and, above all, common sense (E4).

> Ah não. tem coisas bem complicadas de fazer. Eu acho que não só, não é nem a questão das ferramentas em si, mas existem coisas que são muito dinâmicas para serem detectadas com essas ferramentas que geralmente são de análise estática (E5).

> No. No, they [tools] can't [check all aspects] [...] But still it takes a lot of work to have a tool, whatever tool it is, to be able to perfectly check architecture conformance. And specially in small teams it takes some experienced architect to do it. And a lot of it is going to be done by his mind, by his brain (E7).

> No, no one tool can do everything. You have to pick and choose and use the tools you need (E8).

> I don't know any that would be good enough for me, so I'm writing myself. Not really. I don't know really reliable and good enough conformance checking solutions (E9).

> É, não, não. Elas merecem melhorias do ponto de vista de usabilidade. Elas são ferramentas que exigem que o usuário tenha um bom conhecimento arquitetural, e muitas vezes elas não são triviais de serem utilizadas não (E10).

Alguns acreditam que um bom arquiteto deveria entender como as ferramentas poderiam ajudá-lo na verificação de conformidade arquitetural e que os desenvolvedores também deveriam se interessar por essas ferramentas. Ao contrário disso, há uma grande resistência, principalmente pelos desenvolvedores, em utilizá-las.

> And if you want to be a professional developer and partial architect, you really need to understand what tools are out there to only help you. [...] So, I find that it's not just the architects that are missing those extra tools. I think the developers too are not taking a serious look at how important it is to look the meta aspects of a project. The developers typically wanna immediately start coding. They want to get visual things on the screen, and they wanna get that positive feedback. That's the endorphins in their heads. They code something and they see it, it's like chocolate to them. But they don't realize that to maintain this over several months or

> several years, and to add more features to it, it's going to get very difficult. So, yes. To answer your question, I think there are some clever tools out there that are not hot to people at school. And if you want to be a professional developer and partial architect, you really need to understand what tools are out there to only help you (E8).

> Software architect know them but, as I said, much technical personal, even senior engineers, senior team leads, for example, very often either don't know them or are simply resistant against using them. They see them as a waste of time. [...] Very often you get a lot of resistance from developers and testers. They very often are much focused upon code and quite often thy more or less refuse to use tools that are not directly related to code. If you're a good software architect then you have enough charisma to make clear to the guys, maybe with humor, maybe easy going, maybe with authority... "Hey guys, this actually brings a benefit. It brings a benefit to me, the architect; it brings a benefit to the team and also to the quality of your work as a developer". That can take some work to overcome those resistances. When you are between architects, when you work only with other architects, most of the time there are no such difficulties and each of them knows at least one of those tools, then it becomes easier. Towards management it can be hard to justify the cost of using those tools. That's also a role of the architect to say "Yes, I know the tool costs 10.000 euros but it's going to bring you something" (E7).

Alguns profissionais entrevistados afirmaram que a possibilidade de burlar a ferramenta de verificação de conformidade arquitetural é considerado um problema pois, a qualquer momento, o desenvolvedor pode inserir violações arquiteturais.

> Bom, tem uma coisa que me incomoda um pouco: às vezes, nessas ferramentas, elas disponibilizam alguns mecanismos pra você evitar que determinado trecho de código seja avaliado ou condenado por ela. Então, geralmente, eles colocam um comentário, umas coisas meio, então... parecendo um pouco gambiarra né. Então isso eu acho meio questionável a forma como é feito e o perigo disso também. Um modo mal intencionado ele vai colocar um comentário em tudo. [...] e as vezes na hora da pressão, o tempo, os custos, existe uma força econômica que pressiona o desenvolvedor para que entregue o que ele tá fazendo, ainda que não esteja pronto. A gente sabe que essas coisas existem e, às vezes, naquela hora alí pra entregar o desenvolvedor acaba recorrendo dessas coisas [...] ele não tá aceitando meu código mas isso aqui eu não sei fazer diferente [...] o cara vai lá coloca aquele comentariosinho alí, engana a ferramenta e manda. Ele fica alí temporariamente pra sempre né. Então esse mecanismo de bular a ferramenta são questionáveis eu não sei se existe uma forma melhor, realmente tem hora que você não quer que a ferramenta contenha. Talvez uma junção de permissões pra isso, determinados commits podem usar isso, essas diretivas, outros não (E5).

5.3.5 Outras discussões

Percebemos em nossa análise que, muitas vezes, o papel e as atividades que um arquiteto irá desenvolver não são bem definidas pela empresa, ficando, normalmente, sobrecarregado. Além disso, o arquiteto precisa ter experiência e perfil para desempenhar bem estas atividades.

> Normalmente quando se faz planejamento de projeto ele não se planeja o papel do arquiteto de software como realmente ele deve ser [...] muita gente entende que o arquiteto de software é um programador que se destaca no meio de todo mundo e que além de programar vai ser responsável pela arquitetura [...] então começa por aí o papel do arquiteto, a pessoa não tem tempo de desenvolver, manter a arquitetura, manter a documentação arquitetural e fazer essas verificações (E3).

> As an architect I am much too busy with other things. [...] Because I founded out that a good software architect as I just said is shutting all through the organization. [...] I think you cannot be a good software architect if you don't have a lot of experience as a developer. You can't be a software architect at 25 years. It's impossible (E7).

> Porque tem essa questão do arquiteto é uma questão de perfil também, você num elege a pessoa e fala: você vai ser o arquiteto. Porque tem uma série de premissas que a pessoa tem que ter né. De modelagem, de compreensão, experiência passado por vários sistemas (E13).

> Arquiteto de, não tem exatamente, assim, muito bem definido o que um arquiteto de computação, de sistema deve ser aqui no Brasil, em cada empresa é uma coisa diferente (E14).

No âmbito das violações arquiteturais, alguns afirmaram que elas podem trazer problemas de complexidade do código e falta de compreensão. Além disso, seu acúmulo pode levar o sistema à morte (há divergências nesse ponto pois alguns acreditam que o sistema pode continuar vivo mesmo com muitas violações). Porém, há aqueles que veem as violações arquiteturais de forma diferente: acreditam que, às vezes, uma violação pode indicar que havia algo errado na arquitetura ou que há falta de tempo durante o desenvolvimento do sistema.

> I'm trying to think back on previous systems. I look that architecture violations is not always a bad thing. Typically they arise because of what I was just talking about, these problems. If there is something difficult to implement in the system and there is a time commitment, there is a time pressure, that's when you get violations. People violate rules when things get too stressful, or they just have to get it done. And when something is being resistant to getting done, typically that's when violations happen. And when there's resistance, it's telling the architect that the solution here may not be the best thing. So, whenever something is not oiled, or greased, or moving smoothly, it should tell the architect there is a problem, perhaps, with the design. A good architect will look at those violations and not blame someone and say "oh, you didn't do what you were supposed to do". Look at it as: "there was a difficult here and it should be wisely easier. Why is it difficult?"Well, perhaps you'd say "the architecture is wrong". It could be just a training issue too. So you have to eliminate that first. Maybe the person just doesn't understand the architecture, or I didn't communicate correctly... But if you can check up all those things and say "I communicated correctly, there was plenty of time for this person implement it"and that's still true, then you really have to come back and look to the architect and say "Ok, this violation was really just because something was something was too difficult to do. Why? So let's change the architecture"(E8).

Por fim, alguns entrevistados apontaram que a indústria e a academia não estão conectadas em suas atividades. Percebemos que eles, assim como nós, consideram importante que haja uma parceria entre os dois ambientes para que sistemas e pesquisas sejam desenvolvidas cada vez mais com qualidade e utilidade, respectivamente.

> I have a comment which is probably relevant for people working in academy like you. Because, before moving into the industry I've done a PhD and I was considering also doing academic research. What I found out is that all the stuff that you learn in the computer engineering classes at the university you can forget about it every day you got to work in a real company. Because you learn a lot of things like: UML and component modeling standards. And you learn a lot of formal ways for describing software systems architecture. I were aware when you go to the reality these formal solutions are not used at all. I have seen systems which are really huge systems, made of distributed systems, made of tens of components each one of them over one million lines, where the architectural knowledge was written nowhere but in order to get information about the architecture of a software component you had to talk to the guy that architected the component. So one important thing is that there is a huge disconnect between theory and practice (E12).

> 90% dos pesquisadores que eu conheço fazem pesquisa em laboratório sem ratinho. Então, isso é muito comum na computação, o pesquisador tem uma ideia, ele acha que a ideia é interessante, ele lê um paper de não sei quem, de não sei onde, e trabalha naquela ideia, ele não vai ao mercado. Então, pra mim o que falta é ..., isso é um problema, pra mim, grave na academia, em muitas das pesquisas arquiteturais. A pesquisa arquitetural tem que se dar em projeto de software, então, pra mim é inútil eu pensar em uma técnica arquitetural, do ponto de vista acadêmico, ou pensar numa prática arquitetural, ou numa ferramenta, se eu não submeto essa ferramenta, dentro de um projeto real. Então é inventar uma droga ou uma medicina nova sem aplicá-la no ratinho. Então, é vital que a academia se aproxime do mercado. Bom, então pra mim, é uma crítica construtiva no sentido que, as pesquisas em arquitetura elas têm que buscar conexão dos centros de pesquisa nas universidades com projetos do mundo real, ou seja, medir projetos, acompanhar, aplicar as ferramentas, pegar feedback, realmente pra que a pesquisa seja útil (E10).

5.4 Considerações finais

Podemos afirmar que todas as suspeitas, hipóteses e discussões levantadas no *survey* exploratório foram corroboradas em nossas entrevistas. Não sabemos, porém, até onde interfere, nos resultados, o fato dos entrevistados serem um subconjunto dos participantes da etapa anterior. Há, portanto, a necessidade de um *survey* confirmatório, com outros e novos respondentes, para dar mais confiabilidade aos nossos resultados.

Podemos responder a questão de pesquisa RQ1 (referente à definição de arquitetura de *software*), até o momento, afirmando que não há, nem na indústria nem na academia, uma

única definição para o termo arquitetura de *software*. A definição utilizada, na prática, depende dos profissionais (suas experiências) envolvidos no desenvolvimento do sistema e a cultura da empresa com relação à arquitetura e os processos que utiliza.

Além disso, não existe um consenso de quais aspectos arquiteturais compõem a arquitetura, assim como não houve no *survey* exploratório. Os aspectos considerados para compor a arquitetura de um sistema em desenvolvimento depende do objetivo e da experiência do profissional responsável.

Consideramos natural essa falta de consenso pois, como não existe um consenso teórico (na academia), cada empresa e profissional vão agir e pensar a partir de suas experiências, vivências, estudos e conceitos formados. Em nossa opinião, isto torna o trabalho da academia, nesta área, muito mais complexa, uma vez que seu contexto de trabalho fica bastante amplo. Consideramos uma missão muito difícil, para a academia, realizar um trabalho que seja relevante para qualquer tipo de arquitetura, de qualquer tipo de sistema. Para realizar seus trabalhos, a academia (assim como já faz) deve definir seu escopo e o grupo de profissionais a quem pretende atingir com sua pesquisa.

Em contrapartida, apesar de não haver consenso entre os aspectos que fazem parte de uma arquitetura, parece ser aceitável a todos, em nossa opinião, que a arquitetura é um conjunto de decisões sobre o projeto. Essa é uma definição simples e completa que pode indicar um caminho para a academia e suas pesquisas.

Com relação à questão de pesquisa RQ2 (diz respeito à documentação arquitetural) percebemos que, na maioria dos projetos, existe documentação arquitetural, mesmo que seja simples e incompleta, porém, ela não é atualizada com frequência. Além disso, percebemos que cada empresa/profissional tem sua forma de documentar a arquitetura (sendo o texto em linguagem natural a forma mais popular).

Dentre os motivos apresentados para uma documentação não atualizada e/ou incompleta estão: falta de tempo, falta de crença na atividade, tem baixa prioridade no processo de desenvolvimento de sistemas reais, falta de necessidade (aplicação trivial, utilização de metodologias ágeis, equipe pequena, sistema é *opensource*).

Percebemos que mesmo que alguns profissionais não desempenhem muito esforço na atividade, eles tem consciência de que esta é uma atividade importante para a manutenção e entendimento do sistema, para evitar a perda de conhecimento e armazenar as decisões e as

suas justificativas.

Ainda, segundo Parnas [29], todo sistema envelhece e precisa de manutenção e atualização. Além disso, as equipes mudam, por diversos motivos, e, em nossa opinião, deveria haver registrado, em algum lugar diferente da cabeça dos profissionais envolvidos no desenvolvimento, todo o conhecimento sobre o sistema para possibilitar uma melhor atualização e evolução do sistema. Consideramos arriscado o fato dos profissionais acharem suficiente que os membros da equipe tenham conhecimento sobre o sistema, sem que haja uma documentação. Achamos importante que o sistema seja documentado para que outras pessoas possam entendê-lo, continuá-lo e mantê-lo, pois as pessoas passam, mas o sistema, normalmente, continua.

Achamos interessante a existência de uma disparidade de opiniões, com relação à documentação arquitetural, entre alguns entrevistados. Podemos notar isso nas citações dos entrevistados E4 e E7.

> No, I don't consider it's [documentation] important, because I believe the best documentation is code itself (E4).

> And you see that in a lot of open-source projects where there's nothing, there's no model, there're no requirements, there're no, hardly, any test results, you know, design papers, and you see many, many, open-source projects simply dying, after certain time, that could have lived for a long time (E7).

Com relação à questão de pesquisa RQ3, podemos, até o momento, afirmar que, na prática, verificação de conformidade arquitetural é uma atividade que checa se o sistema está sendo desenvolvido de acordo com as decisões arquiteturais (definição similar à da academia). Além disso, a realização ou não dessa atividade, assim como sua periodicidade, depende da empresa, profissional e projeto que está sendo desenvolvido.

Em sistemas críticos existe um processo rigoroso para realizar verificação de conformidade arquitetural. Enquanto isso, em outros projetos, essa atividade não é realizada por vários motivos: falta de preocupação da equipe, falta de crença de que é possível desenvolver um algoritmo que cheque outro por completo, falta de necessidade (considerada pela empresa ou responsável pelo projeto), falta de motivação, falta de uma arquitetura bem definida, falta de tempo e falta de responsável pela atividade.

> One more example from Aerospace, if you want to buy a software, so called flying software that is a software that's onboard of an aircraft or a satellite or even a space system, then there is standard called DO178B that is an Aerospace standard. And

> DO178B says that there are different architectural styles. You have to take one and respect that throughout your architecture. If you don't do that, then at the moment of the certification by national airworthiness authorities, simple going to get a no. Going to get a no because you introduced a possibly danger into your architecture, into your software. And the danger can mean loss of human life (E7).

Percebemos, também, que nos projetos que desenvolvem atividades de verificação de conformidade arquitetural, não é muito comum utilizar ferramentas de apoio. A atividade é, normalmente, feita manualmente e é considerada confiável pelos arquitetos. Além disso, alguns entrevistados consideram a atividade importante para a qualidade do código, porém, não suficiente.

Sobre a questão de pesquisa RQ4 (com relação às ferramentas de apoio), podemos afirmar, até o momento, que as ferramentas de apoio à verificação de conformidade arquitetural são pouco conhecidas e pouco utilizadas. Os motivos são inúmeros: falta de interesse em procurá-las, as ferramentas existentes não conseguem capturar todos os aspectos desejados, as ferramentas existentes não são fáceis de utilizar, falta de confiabilidade nas ferramentas, falta de necessidade, falta de uma arquitetura bem definida, resistência dos profissionais em utilizá-las e tempo para configurar a ferramenta.

Achamos interessante que, durante uma das entrevistas, foi levantada a questão de que, em alguns casos, não é necessário ter uma arquitetura do *software*. Segundo o entrevistado E2, a necessidade de ter uma arquitetura depende da complexidade do sistema a ser desenvolvido.

> So, if you have a small system, right? It is probably best not to worry about architecture and conforming to all kinds of rules and things like that. You should be able to focus on writing some code that solves the problem [...] And then there is a state where you have a huge system, so the Eclipse IDE is big or maybe huge, the software at [company name] is definitely huge, it's crazy how much lines of code we have, and at that level of scale, you definitely need enforcement of these kinds of rules (E2).

Por fim, algumas citações (como a do entrevistado E4) nos mostram que há uma falta de interesse/preocupação com a arquitetura do sistema em desenvolvimento e nos faz refletir que, talvez, só haverá uma preocupação maior se os problemas causados pelas violações arquiteturais forem realmente graves.

> When people call themselves software architects, I sometimes see people who basically blindly follow some rules that they believe are true, and they don't check back if following those rules is actually something that makes sense and their specific context (E2).

> But they don't care about architecture, they care about functionality only, so they start to care about architecture and technical aspects when we say: 'Hey, if we keep this pace, and we keep on adding functionality, the software is going to explode. So we'll have to do it again!'. So, developers don't ask you question, just follow guidelines and they don't change architecture because we decide the architecture, along with the leaders, at the beginning of the project, only (E4).

5.5 Ameaças à validade

Como nossos entrevistados foram voluntários da etapa anterior, temos a ameaça interna de seleção de pessoas. Além de ter sido um subconjunto da etapa anterior, esta etapa tem o problema dos respondentes serem voluntários (normalmente são mais motivados que o resto da população, podendo causar um viés nas respostas). Porém, consideramos que os entrevistados foram diversificados em suas características e esta ameaça não tenha afetado tanto nossos resultados.

Temos, também, uma ameaça de constructo pois as perguntas podem, além de não ter conseguido capturar toda a riqueza do objeto de estudo, ter causado um viés nas respostas (devido ao nosso conhecimento na área acadêmica e os resultados da etapa anterior). Ainda como ameaça de constructo temos a expectativa do pesquisador, podendo colocar um viés nas conclusões da pesquisa. Para contornar essas ameaças, o estudo foi discutido e revisado pelos 3 pesquisadores envolvidos.

Por fim, temos a ameaça externa de não poder generalizar nossos resultados para toda a população de profissionais de desenvolvimento de sistemas reais.

Capítulo 6

Etapa 3: *Survey* Confirmatório

Com o *survey* confirmatório, última etapa deste trabalho, temos como objetivo confirmar nossas principais conclusões acerca do estudo que realizamos. Com esta etapa, pudemos quantificar a taxa de concordância de profissionais com relação às nossas principais conclusões. Neste capítulo, apresentamos o planejamento, a execução, os resultados e as possíveis ameaças à validade da última etapa do nosso estudo.

6.1 Planejamento

Planejamos este questionário (Apêndice D) para conter as principais conclusões do nosso estudo, concebidas nas etapas anteriores, que respondiam nossas 4 questões de pesquisa:

- RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?
- RQ2: Qual a visão da indústria com relação à documentação arquitetural?
- RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?
- RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

Foi utilizada a escala Likert na maior parte do questionário, onde as perguntas são respondidas escolhendo-se um valor entre 1 e 5, de acordo com o nível de concordância do respondente com relação à nossa conclusão. O uso dessa escala facilita avaliar se nossas

conclusões se confirmam. Em alguns casos, foi necessário modificar a escala de resposta para termos uma noção de frequência (por exemplo, "sempre", "nunca"). Além disso, disponibilizamos apenas a versão em inglês do questionário, por não termos conhecimento prévio das nacionalidades dos nossos respondentes.

6.1.1 Metodologia de envio de *e-mails*

O survey confirmatório foi enviado para desenvolvedores usuários do GitHub que possuem 100 ou mais seguidores na comunidade e que possuem e-mails visíveis. Apesar de se tratar de uma plataforma que hospeda projetos de código aberto, grande parte dos usuários trabalha em empresas, sendo candidatos a participar da nossa pesquisa. Além disso, acreditamos que usuários com 100 ou mais seguidores são, provavelmente, profissionais importantes em seus projetos no GitHub, visto que outros desenvolvedores estão interessados em acompanhar seus trabalhos. Esse critério nos deu um total de 4410 desenvolvedores do GitHub. Na Figura 6.1 podemos observar a distribuição de seguidores por desenvolvedor.

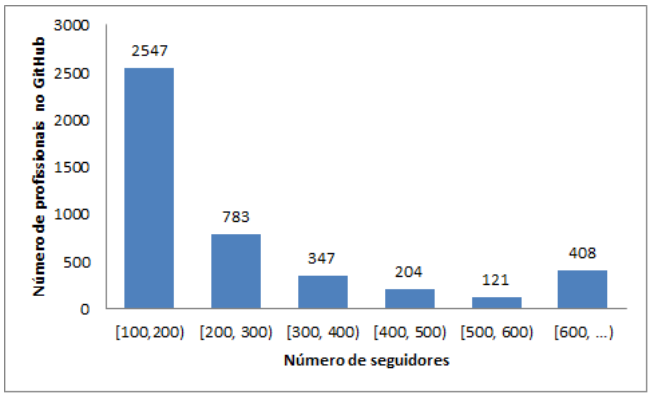

Figura 6.1: Distribuição de seguidores por desenvolvedor.

Como o GitHub é uma espécie de rede social de desenvolvedores, geralmente, segue-se o perfil de desenvolvedores que produzem coisas interessantes nos projetos em que estão engajados. Com isso, consideramos que a chance de encontrarmos pessoas experientes e que realizem coisas relevantes em seus projetos a partir do número de seus seguidores é maior. Seguem os 3 (três) primeiros desenvolvedores com mais seguidores como forma de exemplo para corroborar nossa suspeita (é notável a importância e experiência deles):

1. Linus Torvalds: criador do Linux [43] que possui 26.300 seguidores.

2. Tom Preston-Werner: co-fundador do GitHub [14] que possui 17.900 seguidores.

3. Paul Irish: desenvolvedor *front-end* do Google Chrome [13] que possui 15.500 seguidores.

6.2 Execução

No *e-mail* que enviamos solicitando a participação em nosso *survey*, informamos o teor acadêmico do questionário, o objetivo de nossa pesquisa e o motivo de contactarmos o profissional. Informamos, também, como conseguimos o *e-mail* do desenvolvedor (a partir do GitHub), e nos identificamos como pesquisadores da UFCG e UFMG.

Para o envio dos *e-mails* utilizamos o serviço web MandrillApp através de um programa JavaScript. Segundo o relatório disponibilizado pelo MandrillApp, foram enviados 4352 *e-mails* para desenvolvedores do GitHub. Apenas 58 dos 4410 *e-mails* não foram enviados com sucesso, pois eles estavam escrito em um padrão inválido (utilizando, por exemplo, termos por extenso, como "dot"e "at"ao invés dos respectivos símbolos) ou não existiam. Obtivemos 337 respostas até o dia 18/05/2015 (duas semanas após os envios dos *e-mails*), o que nos dá uma taxa de resposta de, aproximadamente, 7,74%.

Observamos que os 337 respondentes eram de 47 países diferentes. São eles: Taiwan, Israel, Kuwait, Espanha, China, Cazaquistão, Argentina, Japão, Reino Unido, Canadá, Dinamarca, Colômbia, França, Brasil, Grécia, Itália, Bielorrússia, Luxemburgo, Rússia, Venezuela, Alemanha, Finlândia, Suíça, Ucrânia, Países Baixos, Portugal, Holanda, México, Tailândia, Eslovênia, Austrália, Turquia, Índia, Estados Unidos da América, Camboja, Suécia, Irã, Bélgica, Peru, Bulgária, Coréia do Sul, República Checa, Romênia, Indonésia, Bolívia, Holanda e Nova Zelândia.

Além disso, os 337 respondentes afirmaram trabalharem ou já terem trabalhado com 71 linguagens diferentes (a Figura 6.2 mostra as principais linguagens citadas no *survey*). Assim, podemos considerar que nossa amostra é bastante diversificada, tanto com relação à localização, quanto às linguagens que os profissionais trabalham.

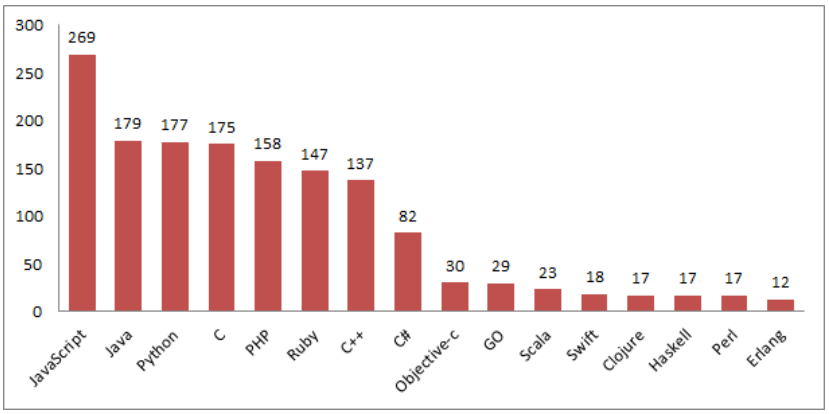

Figura 6.2: Principais linguagens citadas pelos respondentes.

6.3 Resultados

Nesta seção, apresentamos nossos resultados e análises decompondo-os de acordo com as questões de pesquisa, visando facilitar a leitura dos resultados.

6.3.1 RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

A partir das respostas das entrevistas, percebemos que não há um consenso na definição de arquitetura de *software* na prática. Para confirmar este ponto, questionamos o nível de concordância dos respondentes com a conclusão *"Não existe uma única definição para arquitetura de software"*.

A Figura 6.3 mostra os números de respostas por nível de concordância. A primeira coluna apresenta todas as respostas fornecidas. A segunda coluna, nomeada "Nível de Concordância", apresenta as respostas sem considerar o nível 3 ("nenhuma opinião"). Essa coluna visa uma melhor compreensão, possibilitando uma melhor comparação entre a proporção de concordâncias e discordâncas. Esta mesma linha de pensamento foi utilizada para as demais figuras apresentadas neste capítulo.

A partir da Figura 6.3 podemos observar que a maioria dos respondentes (pouco mais de 60%, se considerar os respondentes que não possuem opinião sobre a afirmação; e pouco mais de 80%, sem considerar estes mesmos respondentes) concordam que não existe uma única definição para o termo "Arquitetura de *software*", confirmando, assim, nossa primeira

conclusão.

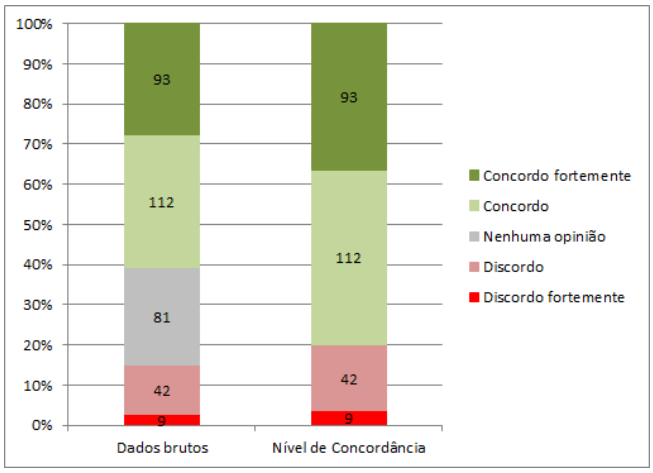

Figura 6.3: Concordância com a afirmação: "Não existe uma única definição para arquitetura de software".

Percebemos, ainda, que cada profissional considera diferentes aspectos de acordo com sua experiência, visão e o local de trabalho. Porém, percebemos, também, que a maioria considera aspectos estruturais como pertencentes à arquitetura de *software*. Por este motivo, questionamos os aspectos que os respondentes consideravam como parte da arquitetura, disponibilizando os 5 principais aspectos citados na etapas anteriores. Vale salientar que o respondente poderia escolher mais de um aspecto como resposta.

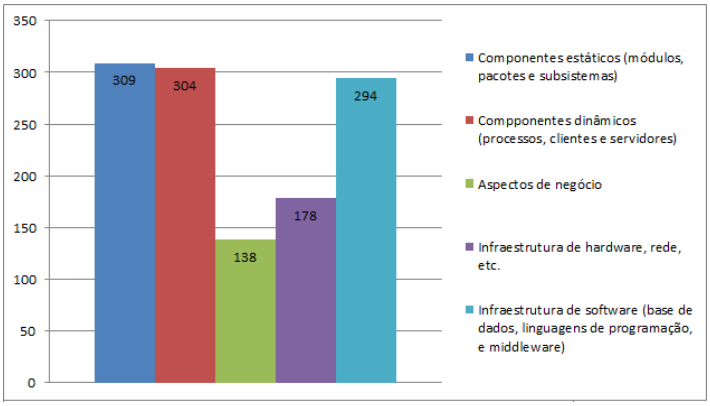

Figura 6.4: Quais aspectos fazem parte da definição de arquitetura?

Com a Figura 6.4 podemos observar que nossa conclusão anterior foi confirmada. Além disso, podemos notar que a maioria dos respondentes do *survey* confirmatório (87,24%) também consideram que aspectos de infraestrutura do *software*, como linguagem de programa-

ção a ser usada e o banco de dados, são aspectos contidos na arquitetura.

6.3.2 RQ2: Qual a visão da indústria com relação à documentação arquitetural?

Nas etapas anteriores, percebemos que também existem divergências, entre os profissionais participantes, com relação à documentação arquitetural de um sistema. Enquanto a maioria considera bastante importante esta atividade, outros consideram um atraso no processo de desenvolvimento. Por este motivo, decidimos questionar com relação ao nível de importância que os respondentes dão à atividade de documentar a arquitetura. A grande maioria concorda que é importante documentar aspectos arquiteturais, como podemos observar na Figura 6.5 (mais de 90%, se não considerarmos as pessoas que não opinaram).

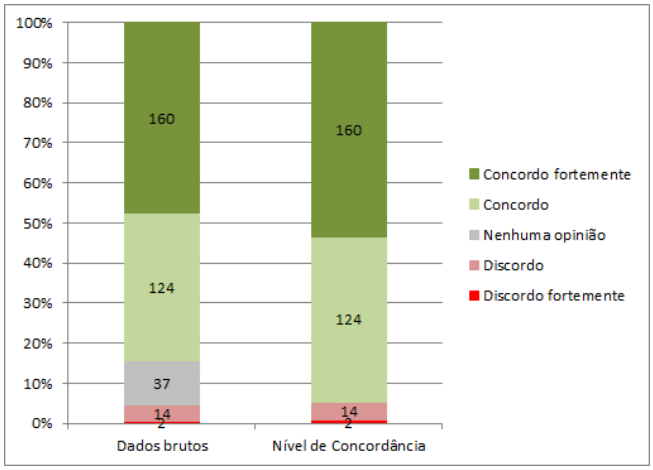

Figura 6.5: Concordância com a afirmação: "É importante documentar aspectos arquiteturais".

Percebemos que existem documentações em seus diferentes níveis de completude. Por este motivo, decidimos questionar sobre a frequência em que uma arquitetura deve ser documentada detalhadamente. Nesta questão, optamos por colocar níveis relacionados à frequência, pois, assim, poderíamos obter respostas mais precisas. Como é possível ver na Figura 6.6, aproximadamente 60% dos respondentes acreditam que a arquitetura deve ser documentada, detalhadamente, na maioria dos projetos ou sempre.

Além disso, havíamos notado que a forma mais comum de documentar a arquitetura é com linguagem natural, sem uma notação específica (ainda que alguns utilizem notações

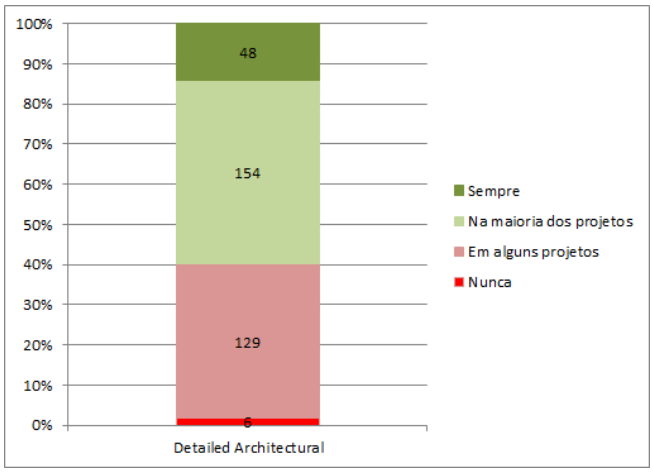

Figura 6.6: Quando a documentação arquitetural deve ser disponibilizada de forma completa e detalhada.

arquiteturais). Questionamos, portanto, o nível de concordância com a frase *"A forma mais comum de documentar aspectos arquiteturais é por meio de texto livre"*. A partir da Figura 6.7, podemos observar que, se não considerarmos os respondentes que preferiram não opinar nesta questão, 57% dos respondentes concordam com a afirmação.

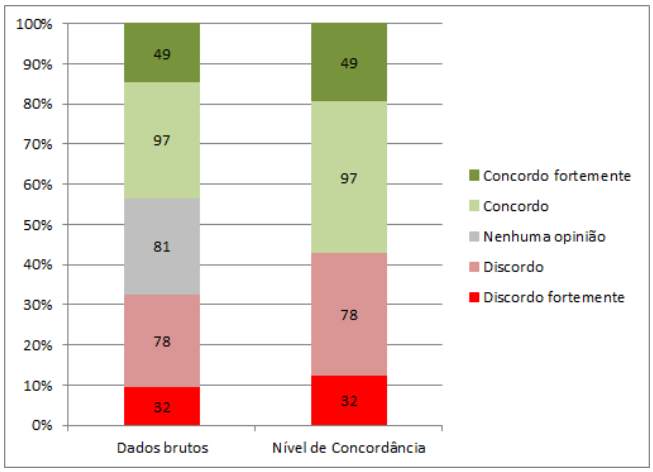

Figura 6.7: Concordância com a afirmação: "A forma mais comum de documentar aspectos arquiteturais é por meio de texto livre".

Ainda, 84% dos respondentes (sem considerarmos os que preferiram não opinar) concordam que a documentação, quando existente, não é atualizada, como pode ser visto na Figura 6.8.

Nas etapas anteriores, vimos que os profissionais consideram a documentação importante

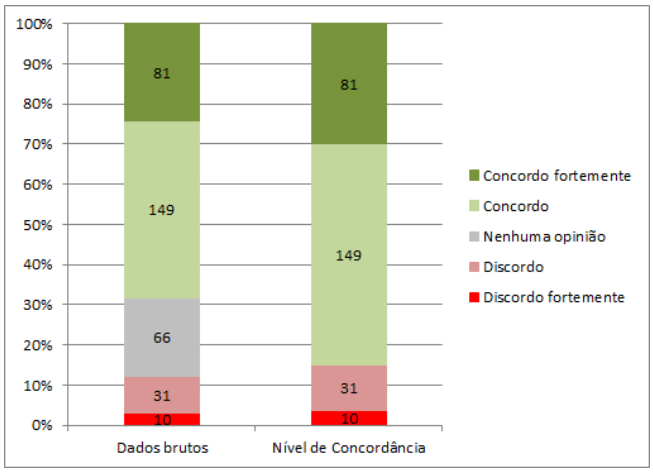

Figura 6.8: Concordância com a afirmação: "Documentação arquitetural, quando existente, não é atualizada".

para o entendimento do sistema e que as violações arquiteturais podem prejudicar o *software*. Por este motivo, questionamos se os profissionais consideram que há um impacto negativo na produtividade dos desenvolvedores, quando a documentação não está atualizada.

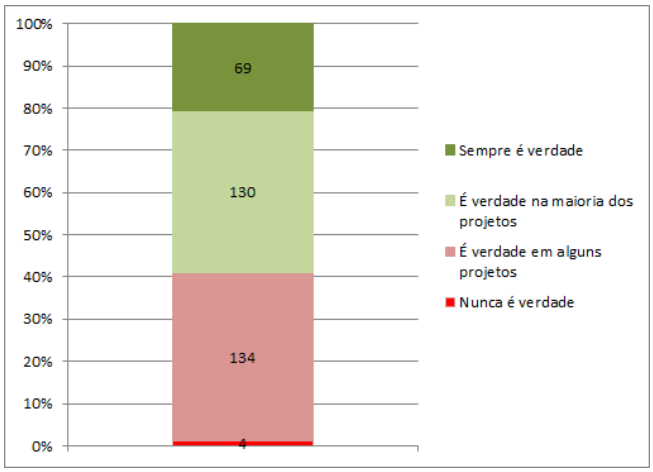

Figura 6.9: Quando a não atualização de documentos arquiteturais impacta, negativamente, na produtividade dos desenvolvedores.

Segundo a Figura 6.9, 59% das pessoas concordam que a não atualização dos documentos arquiteturais causa impacto negativo na produtividade dos desenvolvedores na maioria dos projetos em desenvolvimento. Além disso, 39,7% concordam que há impacto negativo em alguns projetos, não descartando a possibilidade levantada na questão.

6.3.3 RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

Também percebemos que há uma divisão (quase igualitária) entre os que realizam verificação de conformidade arquitetural e os que não realizam. Por essa divisão, decidimos questionar a importância da atividade, fornecendo uma escala de frequência para termos uma resposta mais precisa.

É possível observar, na Figura 6.10, que nossa conclusão é confirmada. Um pouco mais da metade dos respondentes consideram que não é importante ou é importante em apenas alguns projetos realizar a verificação de conformidade arquitetural. Na mesma linha, a outra metade considera que é sempre ou quase sempre importante realizar tal atividade.

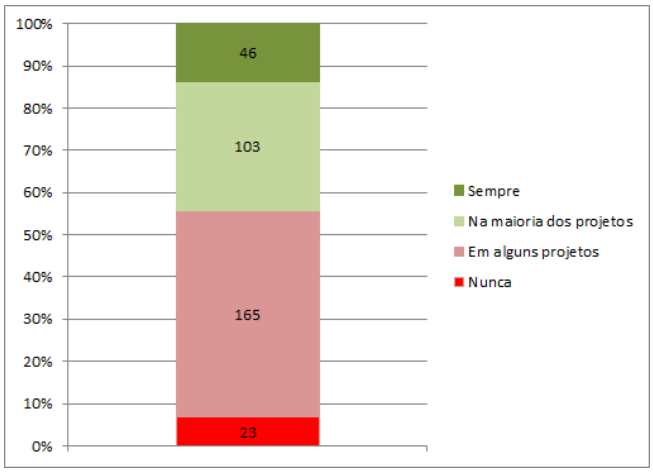

Figura 6.10: Quando é importante realizar verificação de conformidade arquitetural.

É, ainda, uma conclusão de nosso estudo o fato de que, mesmo os que consideram a atividade de verificação de conformidade arquitetural importante e a realizam, geralmente não a realizam em sua completude, de forma precisa. A partir da Figura 6.11, podemos corroborar nossa conclusão.

Segundo a Figura 6.12, quase 100% dos respondentes acham que a atividade de verificação de conformidade arquitetural não é suficiente para garantir a qualidade do código. Assim, concluímos que, apesar de importante (conforme dito por 44,21% dos respondentes - Figura 6.10) essa atividade não é suficiente para garantir a qualidade do sistema.

Os que não realizam a atividade de verificação apresentaram vários motivos, desde a falta de tempo até o desconhecimento dessa atividade (como já foi dito nos capítulos anteriores).

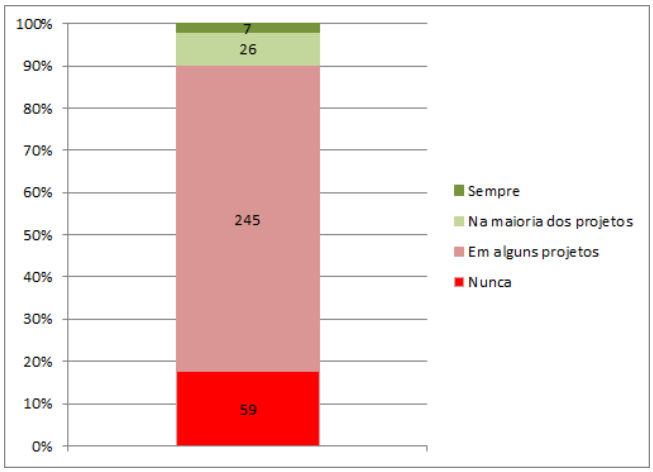

Figura 6.11: Quando a verificação de conformidade arquitetural é realizada de forma precisa.

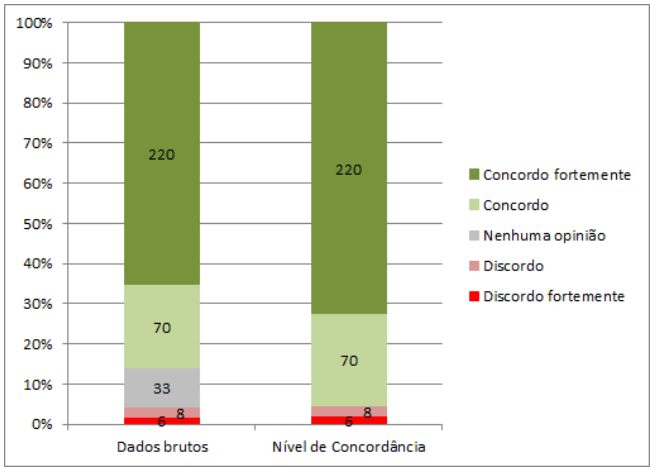

Figura 6.12: Concordância com a afirmação: "Verificação de conformidade arquitetural não é suficiente para garantir a qualidade do código".

Os que realizam, normalmente, não utilizam ferramentas de suporte, ou seja, a verificação é feita manualmente.

Na Figura 6.13 é possível observar que, se não considerarmos as pessoas que optaram não opinar nessa questão, mais de 70% concordam que, quando realizada, a verificação de conformidade arquitetural é feita sem uma ferramenta de suporte, ou seja, manualmente. Podemos observar, ainda, que 37% dos respondentes preferiram não opinar nessa questão, o que nos dá a ideia de que grande parte dessa porcentagem realmente não realiza tal atividade e não sabe como é realizada em outras empresas e locais.

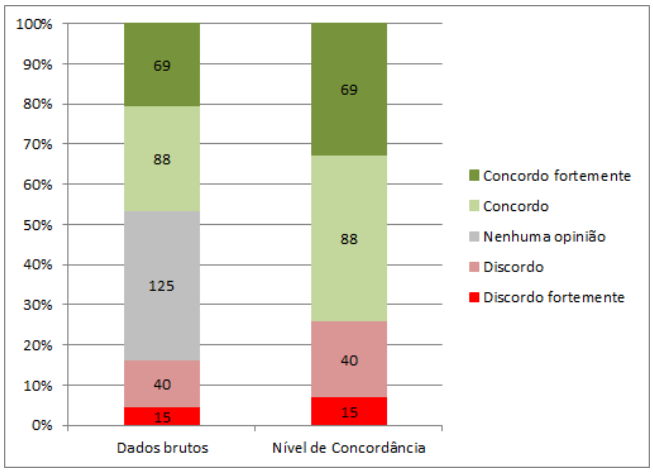

Figura 6.13: Concordância com a afirmação: "Verificação de Conformidade Arquitetural, quando realizada, é, feita sem uma ferramenta de suporte".

6.3.4 RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

A partir das etapas anteriores, concluímos que são poucos os que utilizam ferramentas de apoio à verificação de conformidade arquitetural. Confirmamos esta conclusão com a quase unanimidade entre os respondentes desta etapa (Figura 6.14).

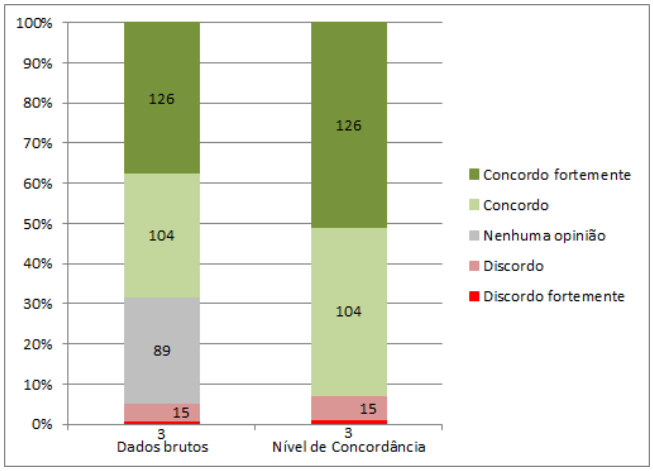

Figura 6.14: Concordância com a afirmação: "Ferramentas de apoio à verificação de conformidade arquitetural são raramente utilizadas".

Também questionamos, aos respondentes, quando essas ferramentas deveriam ser utilizadas. A partir da Figura 6.15, podemos observar que a maioria (mais de 70%) considera

que ferramentas de apoio à verificação de conformidade arquitetural devem ser utilizadas em poucos projetos ou em nenhum, confirmando ainda mais que a taxa de utilização de ferramentas para esta atividade é baixa.

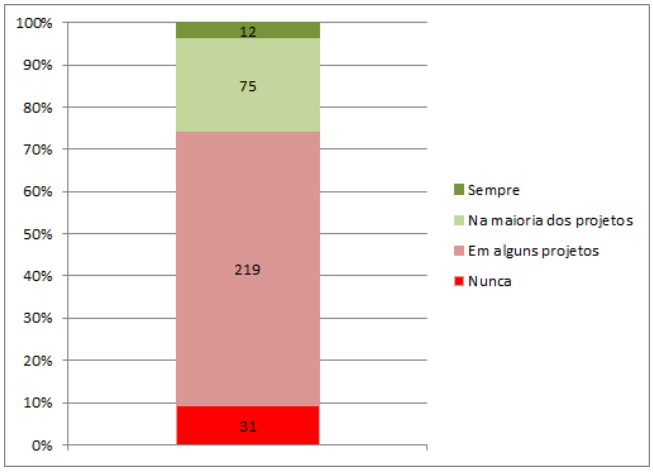

Figura 6.15: Quando ferramentas de apoio à verificação de conformidade arquitetural devem ser utilizadas.

Percebemos, pelos relatos nas etapas anteriores, que existem ferramentas complexas e que requerem especificações detalhadas da arquitetura. Na Figura 6.16 podemos observar que, dentre os que opinaram, é quase unanimidade, também, com relação a essa conclusão. Vale citar que a taxa de "omissão" para essa pergunta foi alta (48,66%), o que nos dá indícios de que essas ferramentas não são muito difundidas e conhecidas na prática.

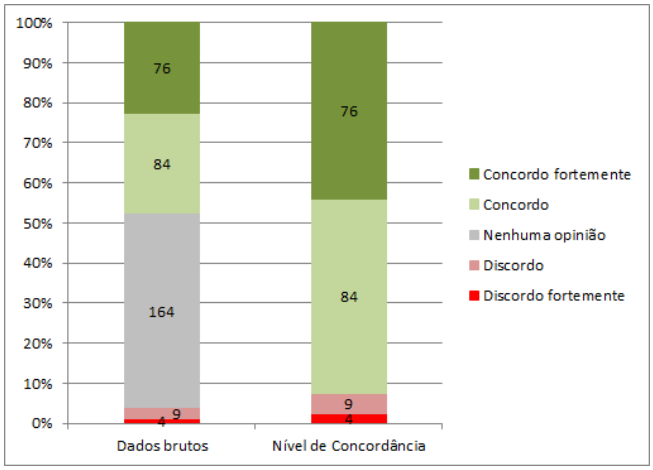

Figura 6.16: Concordância com a afirmação: "Ferramentas de apoio à verificação de conformidade arquitetural são complexas e requerem especificações detalhadas".

Notamos, também, que há uma resistência entre os desenvolvedores para utilizar essas ferramentas no desenvolvimento dos sistemas. A partir da Figura 6.17, podemos observar que, se não considerarmos as pessoas que preferem não opinar na questão, 85,5% concordam com esta afirmação. Novamente, a taxa de "omissão"foi alta o que nos faz refletir se a questão é a falta de conhecimento das ferramentas ou pelos respondentes preferirem não opinar por outras pessoas.

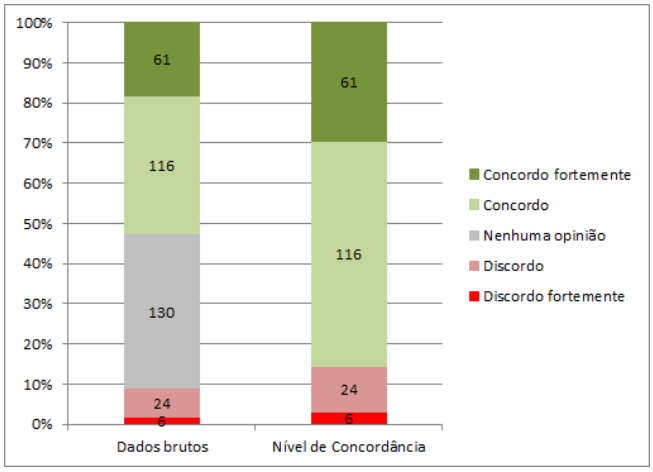

Figura 6.17: Concordância com a afirmação: "A maioria dos desenvolvedores resistem a utilizar ferramentas de apoio à verificação de conformidade arquitetural".

6.3.5 Outras discussões

Achamos importante, ainda, confirmar outros pontos nesse *survey* confirmatório. O primeiro deles foi relacionado ao papel de arquiteto. Percebemos que este não é bem definido. Iniciamos esta suspeita em nossa primeira etapa, ao procurarmos profissionais para participar de nosso estudo. A partir da Figura 6.18 é possível observar que 92,8% dos respondentes que expressaram alguma opinião concordam que o papel de um arquiteto de *software* não é bem definido na prática, variando de projeto para projeto, de empresa para empresa, confirmando, assim, mais uma de nossas conclusões.

Além disso, quisemos confirmar que, mesmo não demonstrando desempenhar muito esforço nas atividades de documentação e verificação, os profissionais consideram que violações arquiteturais tem um impacto negativo no código, podendo levar o sistema à morte (como muitos entrevistados concordaram). A Figura 6.19 mostra que 78,6% dos responden-

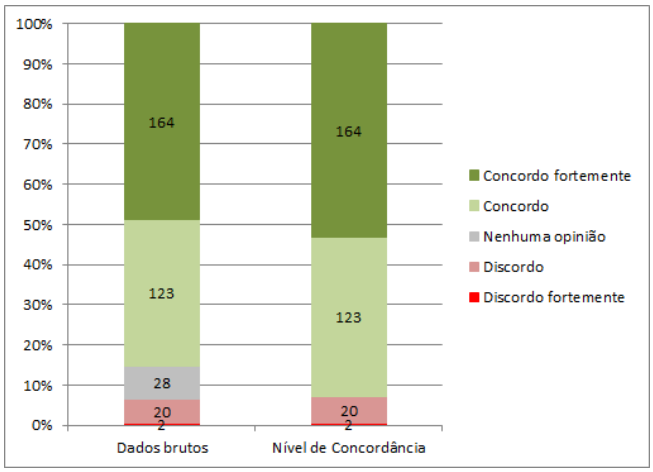

Figura 6.18: Concordância com a afirmação: "O papel de arquiteto de software varia para cada projeto".

tes concordam que violações arquiteturais possuem um impacto negativo na qualidade do código.

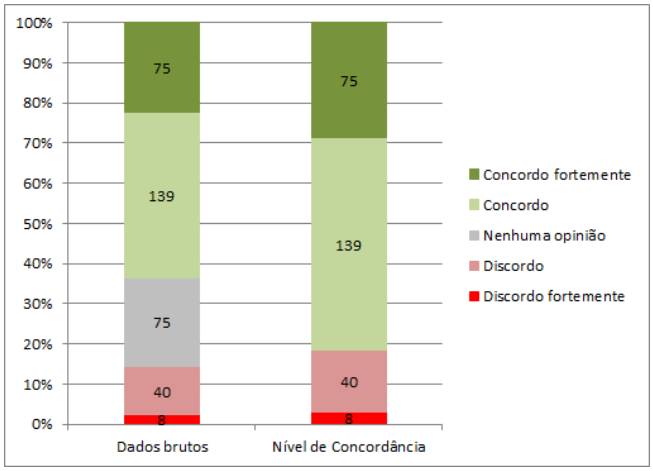

Figura 6.19: Concordância com a afirmação: "Violações arquiteturais (ou seja, código que não segue as decisões arquiteturais) possuem um impacto negativo na qualidade do código".

Por fim, quisemos confirmar o que alguns entrevistados apontaram questionando o nível de concordância com a afirmação *"Academia e indústria não possuem um interesse mútuo em sincronizar seus trabalhos e agendas em arquitetura de software"*. A partir da Figura 6.20 observamos que 80% de nossos respondentes que opinaram concordam que a academia e a indústria não possuem interesse em sincronizar as pesquisas e trabalhos realizados no tema de arquitetura de *software*, ficando, assim, cada setor trabalhando separadamente, de

diferentes formas.

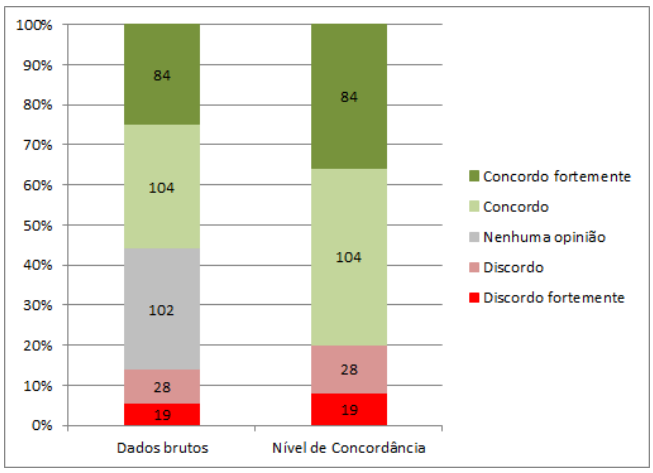

Figura 6.20: Concordância com a afirmação: "Academia e indústria não possuem um interesse mútuo em sincronizar seus trabalhos e agendas em arquitetura de software".

Além disso, deixamos um espaço no *survey* confirmatório para o respondente que quisesse discorrer sobre o tema e/ou questionário em questão. Obtivemos muitos comentários e, alguns, são interessantes para compartilhar.

Por exemplo, alguns respondentes fizeram questão de afirmar que a arquitetura do sistema e sua equipe de desenvolvimento estão sempre em mudança e que, por isso, utilizar ferramentas bloqueia a evolução e o crescimento do sistema e da arquitetura. Isso já havia sido citado nas etapas anteriores, quando os profissionais afirmaram que utilizar ferramentas poderia engessar o processo de desenvolvimento.

> Architecture change over time, and stuff that is relevant at a particular point isn't important afterward. This is especially true when conditions (business, requirement, technology) changes. Trying to enforce architecture decisions made ahead of time, especially using a tool is going to create an environment in which devs are blindly following the tool. That leave no place for exploration, or growth in the architecture of the solution (R11).

> Interestingly, in open source, architectural conformance is typically a moving target. Leaders come and go, and with them the vision for a given project. Even when leadership is stable, contributors come and go, and often bring new perspectives and ideas that inform the architecture of a project. As such, "architectural conformance tools"are relatively rare for OSS projects, because the architecture is a moving target (R5).

Continuando na linha de pensamento relacionada ao cargo de arquiteto de *software*, outro respondente afirmou que, em muitas organizações, o arquiteto de *software* é um desenvol-

vedor *senior* (o que corrobora afirmações anteriores da sobrecarga de atividades que um arquiteto possui).

> In many organizations, the title of "Product Manager"or "Architect"is a sign or greater responsibility or seniority, but not clearly distinct from that of a senior software developer. I believe we would all benefit from giving these terms more of a formal meaning and clearing what are the benefits of Software Architecture (R22).

A partir dos comentários, observamos que existem diferentes culturas de programação, baseadas em diferentes linguagens e com diferentes conceitos. Então, termos utilizados em uma cultura podem não ser conhecidos por outra. Por exemplo, mesmo havendo uma breve explicação no questionário sobre o conceito de Verificação de Conformidade Arquitetural, muitos afirmaram não saber do que se tratava. Ou seja, talvez os termos acadêmicos sejam diferentes dos utilizados na prática.

> “Architectural conformance" doesn’t seem to be on people’s minds as a term, at least in the community around here (SF Bay Area). I had to read some articles about it to figure out exactly what was going on. There’s probably some overlap that’s hard to capture because different people use different terms for similar things (R173).

> Really sorry, but I have a 20+ year career with a variety of languages and I have no idea what "architectural conformance checking"is supposed to mean. I can’t even imagine how it would be possible for significant variance to exist between code and architecture, except in the case of near-catastrophic incompetence, although to be fair, near-catastrophic incompetence is pretty common in software. [...] Programming’s made up of a staggering number of subcultures, and these subcultures are intensely tribal and often very explicitly hostile and aggressive to one another. And they correspond loosely to development philosophies, and understandings of "common"terms. (Which typically do not really have shared meanings.) [...] "Software architecture"is a phrase which invites ridicule in the Ruby subculture, excessive seriousness in the Java subculture, and which probably only leads to a really intelligent discussion in subcultures like JavaScript’s or Clojure’s. Probably Python’s, too (R35).

> Sorry I don’t know/never used Architectural conformance checking (R79).

> I think the language should be adjusted a bit when reaching out to people who practice software development. You use the phrase "Architectural Conformance Checking"which I have never heard anyone say in my 13+ years of working professionally with software development. I assume that the phrase is an umbrella term for "unit testing", "integration testing", "code/software review", "software architecture planning", "code style guidelines", "code linting", "reference counting", "code coverage", "static code analysis", and other terms commonly used by developers to keep their code functional and maintainable. When asking developers out there about these things, I think the terms I mentioned would be better to get more accurate answers (R140).

Uma visão diferente sobre a relação entre a indústria e a academia é que o fato de não haver interesse mútuo entre as duas não significa ter conflitos entre elas.

> WRT the final question, I think that there being no mutual interest in synchronising should not be read as there being a conflict of interest ... I think that WRT software architecture industry and academia are almost entirely talking past one another (R56).

> From taking this survey, it seems that academia and industry are on entirely different pages. Would be interested in seeing a link to any publications that result from this (R165).

Por fim, nos chamou a atenção o comentário feito por um respondente, onde ele afirmou que, durante sua experiência industrial, não viu nenhuma discussão sobre arquitetura de *software*.

> In my industry experience, no one talks about software architecture. Very few people think at an architecture level. There are no formal architecture definition. Just informal very high level documentation (e.g. a figure with a list of interacting process, big boxes for big systems). I never saw the use of ADL. Noone ever called himself a software architect (R99).

6.4 Ameaças à validade

A primeira é a ameaça externa pois não podemos generalizar os resultados (assim como todo o nosso estudo não pode ser generalizado) uma vez que nossa amostra de respondentes, nesta etapa, foi composta, exclusivamente, por usuários do GitHub.

Além disso, tivemos alguns percalços durante a execução da etapa. O MandrillApp, por motivo desconhecido até o momento, após o envio dos *e-mails*, cancelou automaticamente nossa conta devido a uma dita baixa reputação. O MandrillApp havia modificado a URL do *hiperlink* nos primeiros 1300 *e-mails* enviados para contabilizar o número de *clicks*. Essa opção foi desativada assim que percebemos sua existência. Porém, após o cancelamento da conta, a URL ficou inválida e pode ter causado problemas no acesso para esses usuários.

Para contornar esta situação, decidimos reenviar esses 1300 *e-mails* informando o *link* correto. Porém, no meio do processo de re-organização de nossa metodologia, recebemos uma mensagem do suporte do GitHub solicitando que parássemos de enviar mensagens para os membros da rede. Suspeitamos que erramos em nossa estratégia ao mencionar que obtivemos os *e-mails* dos desenvolvedores pelo GitHub, possibilitando que reclamassem para o suporte.

Com isso, mesmo havendo enviado os *e-mails* necessários (segundo o relatório gerado pelo serviço, dos 4410 *e-mails*, foram enviados apenas 4352, conforme mencionado anteriormente), alguns ficaram com o *link* inválido, o que pode dificultar o recebimento de respostas por esses profissionais. Alguns entraram em contato informando que o *link* estava errado e solicitando o *link* correto. Para esses, respondemos prontamente.

Há, também, a ameaça interna com relação à escolha dos candidatos a respondentes. Para contornar estas situações, tentamos utilizar critérios racionais e bem definidos, além de ter cuidados para o envio de *e-mails* (apesar dos problemas citados anteriormente).

Ainda há a ameaça de constructo, pois as pessoas podem ter respondido sem possuir um conhecimento profundo sobre o assunto. Ainda nessa linha, o questionário pode não ter capturado toda a riqueza do ambiente em questão, além de que nossas análises podem ter sido tendenciosas (tentamos contornar as duas últimas ameaças com planejamento e revisões).

Muitos respondentes deixaram comentários afirmando não saber o que significava Verificação de Conformidade Arquitetural (mesmo na presença de uma nota de explicação no *survey*), ou afirmando não ter prática em arquitetura. Por exemplo: *"To some of the questions, my answer would have been a strong 'I do not know', but that was not an option. Your results will be somewhat skewed as a result as not every respondant may even know what an architectural conformance tool even is." (R134).* As questões que o respondente citou foram as que possuíam as opções *never* (nunca), *in some of the projects* (em alguns projetos), *in most of the projects* (na maioria dos projetos), *always* (sempre). Todas as outras possuíam opções para responder "nenhuma opinião".

Consideramos que podem haver distorções nas respostas pela falta de conhecimento de alguns respondentes no assunto completo (alguns podem ter conhecimento em documentação, mas não em verificação, por exemplo). Porém, além de ter utilizado critérios bem definidos para enviar os *e-mails* com o *survey* (consideramos que os profissionais possuem um bom nível de experiência), imaginamos, também, que se um respondente percebe não ter conhecimento suficiente para responder às questões, este, provavelmente, abortará o *survey*. Além disso, no corpo do *e-mail* do convite, deixamos claro qual era nosso público alvo.

Após a execução desta etapa, aprendemos algumas lições que servirão para estudos futuros:

- Ao utilizar *e-mails* disponibilizados no GitHub, é preciso deixar claro que não envi-

aremos mais *e-mails* além daquele. Como informamos no corpo da mensagem que os desenvolvedores estavam sendo contactados a partir do GitHub, os profissionais podem ter ficado com receio de receber novas mensagens, realizando, assim, uma denúncia de *spam* para o suporte.

- É importante enviar um *e-mail* para o suporte do GitHub solicitando permissão e informando o teor de nossa pesquisa. Se o suporte soubesse o teor de nosso estudo e tivesse nos dado permissão para realizá-lo, provavelmente, não receberíamos um *e-mail* solicitando que não mais enviássemos mensagens para os membros da rede.

- Recebemos a mensagem do suporte do GitHub após começar a reenviar os *e-mails* para corrigir o *link* quebrado. Portanto, tiramos como lição não enviar mais de um *e-mail* para a mesma pessoa. Com isso, é muito importante o planejamento de como enviar os *e-mails* (principalmente se forem muitos). Apesar de nossa preocupação e atenção para este ponto, ainda assim, tivemos problemas.

Capítulo 7

Considerações Finais

Ao final deste trabalho, podemos elencar algumas considerações finais e ter alguns indícios de como é o ambiente de desenvolvimento de sistemas reais com relação à arquitetura de *software*, documentação e verificação de conformidade arquitetural, além do uso de ferramentas de apoio à esta atividade. Além de conclusões e indícios, nos restaram muitas perguntas e questões de pesquisa para trabalhos futuros.

7.1 Discussões sobre os resultados

É importante frisar que as nossas principais conclusões das duas primeiras etapas (*survey* exploratório e entrevistas) foram confirmadas pela última etapa (*survey* confirmatório). Com isso, temos uma maior confiabilidade em elencar algumas considerações e alguns indícios a partir de todas as respostas. Além da confirmação com a última etapa, a triangulação de dados que utilizamos em nossa pesquisa também nos dá uma maior confiança nos resultados obtidos.

Temos consciência de que nosso estudo não pode ser generalizado e não contempla uma percepção completa do ambiente estudado, porém, após as 3 (três) etapas, possuímos uma visão geral para cada questão de pesquisa.

RQ1: Qual é a visão da indústria sobre a definição de "arquitetura de *software*"?

A visão da indústria, segundo nossos respondentes e entrevistados, é de que não existe uma única definição para o termo "arquitetura de *software*". Nossos dados nos permitem

concluir que a forma como este termo é utilizado depende de vários fatores, dentre eles a cultura da empresa, o profissional e o projeto. Inclusive, foi citado por alguns respondentes que os termos que envolvem esta área podem variar de lugar para lugar, possibilitando, assim, interpretações diferentes para o mesmo termo.

Além disso concluímos, a partir dos dados e análises, que diversos aspectos podem fazer parte ou não da arquitetura (aspectos estruturais, dinâmicos, de infra-estrutura, do domínio do problema, etc.). Os aspectos utilizados na arquitetura dependem de alguns fatores como experiência e visão do profissional responsável, tamanho do projeto e objetivo com a arquitetura.

Consideramos natural esses achados e já os esperávamos, uma vez que também não existe consenso, no meio acadêmico, sobre essa definição. Como vimos, o SEI possui catalogados mais de 90 (noventa) definições diferentes para este termo. Como é a academia que forma os profissionais que trabalham no mercado, nada mais natural do que estes também não possuírem consenso na definição para arquitetura de *software*.

Apesar de não existir consenso nessa definição (nem na academia, nem na indústria), conseguimos identificar que, independente de quais aspectos fazem parte ou não, uma arquitetura de *software* é um conjunto de decisões sobre o sistema em desenvolvimento (e isto vale tanto na teoria quanto na prática).

RQ2: Qual a visão da indústria com relação à documentação arquitetural?

Com nosso estudo, podemos concluir que, assim como na academia [8], os profissionais da indústria, de uma forma geral, consideram que documentar a arquitetura é uma atividade importante. Essa atividade, segundo os profissionais participantes de nosso estudo, auxilia a manutenção e o entendimento do sistema, evita a perda de conhecimento sobre o sistema e armazena as decisões e justificativas sobre elas.

Porém, notamos que, pelas ponderações dos nossos respondentes, eles consideram que não são todos os projetos que necessitam de uma arquitetura precisa e bem definida. Para a maioria dos respondentes, dependendo do projeto, é bastante aceitável existir documentações informais, incompletas ou desatualizadas. Os sistemas críticos, por outro lado, exigem documentações rigorosas, uma vez que sua falha pode causar grandes problemas.

Em nossa opinião, considerar "aceitável não ter uma boa documentação"pode ser preo-

cupante pois não existem critérios bem definidos e testados para afirmar qual sistema precisa ou não de uma boa documentação. Além disso, essa atitude de considerar "aceitável", unido ao sentimento de "ser uma atividade chata" (como alguns respondentes afirmaram), pode ser prejudicial para os profissionais envolvidos no desenvolvimento do sistema, devido à possível falta de comprometimento com a atividade de documentação.

Em nossa visão, se os profissionais, em sua maioria, consideram a documentação arquitetural uma atividade importante (como dito, anteriormente), ela deveria ser executada para qualquer tipo de sistema desenvolvido.

Nosso trabalho nos permitiu, a partir dos dados coletados, criar a hipótese de que documentar a arquitetura e deixá-la atualizada é mais uma questão de prática cultural dos profissionais. Pode faltar o hábito/costume, dentro das equipes, de que é preciso realizar essas atividades e se esforçar para que a documentação esteja sempre atualizada.

É necessário frisar que documentação, geralmente, não é um artefato que precisa ser entregue para o cliente, mas é um artefato importante para o desenvolvimento de um sistema de qualidade, que é o objetivo de todo profissional de desenvolvimento de sistemas reais.

Vale ressaltar que essas hipóteses, apesar de intuitivas, não são embasadas por resultados de estudos e dados concretos, ainda.

Também, a partir dos dados coletados, podemos concluir que os principais motivos para produzirem documentações incompletas, informais e/ou desatualizadas são: a baixa prioridade dada a atividade; não ser considerada uma atividade atrativa por parte dos desenvolvedores; falta de crença de que a documentação pode trazer benefícios; por considerar desnecessário em aplicações triviais, com metodologias ágeis, com uma equipe pequena, com sistemas *opensource*; e, principalmente, a falta de tempo durante o processo de desenvolvimento.

Com isso, levantamos algumas questões para refletir sobre cada motivo que leva uma equipe a ter uma documentação ruim. Por exemplo, a baixa prioridade dada a essa atividade seria causada pelo fato de que a documentação é um artefato que não precisa ser entregue ao cliente? Se a documentação fosse um artefato a ser entregue, assim como o sistema o é, seria empregado mais esforço e preocupação nessa atividade?

Além disso, justificar que a documentação não é atualizada pois essa é uma atividade, considerada por alguns, como sendo chata, em nossa visão, causa certo estranhamento, pois

provém de profissionais de desenvolvimento de sistemas reais. Nos preocupa muito esse tipo de resposta pois nos mostra que pode ser um problema de cultura (de profissionais e das empresas), de ensino e/ou de consciência profissional. Este é um problema apresentado tanto pelos profissionais que participam da equipe de desenvolvimento quanto pelos superiores que devem cobrar de seus desenvolvedores um compromisso para desenvolver produtos/serviços de qualidade.

Ainda, a falta de tempo para documentar a arquitetura e atualizá-la pode gerar *bugs* e defeitos no sistema, além de perder o conhecimento e estrutura do *software*. Com isso, acreditamos que será necessário empregar mais tempo para entender o código e adicionar qualquer nova funcionalidade, além de perder mais tempo corrigindo os erros. Certamente podemos notar que, talvez, quanto menos tempo para documentar, poderão existir mais problemas no sistema e menos tempo haverá para solucioná-los de forma eficaz e eficiente. Porém, mais uma vez, esta é uma hipótese nossa que precisa ser estudada e analisada a partir de novas pesquisas.

A partir dos dados podemos concluir que cada empresa tem sua forma de documentar, tanto em formatos de documentos utilizados quanto na notação utilizada. Consideramos esse aspecto "natural", uma vez que existem diversas opções, linguagens e ferramentas propostas pela academia. Percebemos, porém, que a maioria dos respondentes utiliza texto em linguagem natural como principal forma de documentação.

Por fim, encerramos esta questão de pesquisa com outro questionamento: se a maioria dos profissionais considera a documentação arquitetural uma atividade importante, por que não aplicam mais esforço em tal tarefa?

RQ3: Qual a visão da indústria com relação à verificação de conformidade arquitetural?

De acordo com nossos respondentes, a visão da indústria, de uma forma geral, é que a verificação de conformidade arquitetural é importante para a qualidade do código, porém, não é uma atividade suficiente.

Além disso, a partir de nossos dados, concluímos que nem sempre são realizadas atividades com esse propósito por diversos motivos, dentre eles: falta de crença de que é possível calcular a arquitetura para aplicar um algoritmo que faça a checagem completa; falta de ne-

cessidade, na visão da empresa ou do profissional; falta de um profissional responsável pela atividade; falta de tempo; falta de motivação dos profissionais envolvidos no desenvolvimento do sistema; e a falta de uma arquitetura bem definida.

Em sua maioria, os motivos pelos quais os profissionais não realizam verificação de conformidade arquitetural são os mesmos relatados para justificar uma documentação arquitetural ruim. Consideramos esse aspecto como algo "natural", uma vez que observamos as opiniões dos mesmos profissionais sobre duas atividades que são sequenciais e tratam sobre o mesmo tema (arquitetura).

Por fim, para os que realizam a atividade de verificação, concluímos que não é comum utilizar ferramentas de apoio a essa atividade e, quando utilizadas, são ferramentas construídas pela própria indústria. Existe uma preferência por realizar a atividade manualmente e os profissionais possuem confiança em seus métodos de checagem manual. Isto confirma nossa motivação inicial (que nos impulsionou para executar esse estudo) de que a indústria não parecia utilizar as ferramentas produzidas pela academia. Além disso, a periodicidade da checagem depende de cada empresa em que trabalham ou dos profissionais envolvidos no projeto.

RQ4: Como é a relação da indústria com as ferramentas de apoio à verificação de conformidade arquitetural existentes?

A indústria, de acordo com nossos estudos, conhece/utiliza pouco as ferramentas de apoio à verificação de conformidade arquitetural. Nossos dados nos permitem concluir que existe uma falta de interesse/necessidade por parte de muitos profissionais em encontrar/utilizar essas ferramentas.

Os profissionais que conheciam alguma ferramenta do gênero, em sua maioria, afirmaram que as ferramentas existentes não são tão fáceis de utilizar e, além disso, não conseguem capturar todos os aspectos importantes da arquitetura de um sistema. Esperávamos esse tipo de resposta pois, em nossos estudos anteriores, já havíamos percebido que aspectos dinâmicos (em tempo de execução), por exemplo, não eram capturados pelas ferramentas que testamos (*dclchek* [46] e *DesignWizard* [5]).

Em tempo, pelas respostas fornecidas, podemos concluir que algumas ferramentas exigem uma documentação bem definida para serem utilizadas. Após todo o nosso estudo, sa-

bemos que não seria possível, para a maioria dos profissionais participantes desta pesquisa, utilizar essas ferramentas pois não possuem uma documentação bem escrita e definida.

Por fim, notamos que alguns respondentes afirmaram gastarem muito tempo na configuração das ferramentas. Assim, refletimos que, talvez, a união dessas justificativas sejam os motivos para que exista uma certa resistência, por parte de alguns profissionais, em utilizar esses tipos de ferramentas.

Outras discussões

Concluímos, a partir das respostas às questões de pesquisa, que a maioria dos profissionais envolvidos neste estudo considera importante documentar a arquitetura e verificá-la, além de concordar que violações arquiteturais impactam negativamente para a produtividade do desenvolvimento do sistema (similar a resultados da academia [5; 6; 46]). Porém, mesmo assim, os profissionais não demandam muito esforço/tempo para a realização precisa das atividades de documentação e verificação (por diversos motivos já citados anteriormente).

Essa constatação nos dá a impressão de que documentar e verificar a arquitetura de *software* só serão atividades com alta prioridade e terão a devida atenção por parte de todos os envolvidos no desenvolvimento de um sistema real, se os problemas causados pelas violações arquiteturais forem muito graves.

Além disso, no decorrer de nossa pesquisa percebemos que o papel que um arquiteto de *software* (quando existe) irá desempenhar depende da empresa em que ele trabalha, local e experiência, não havendo, assim, atividades padrões e universais para um arquiteto de *software*.

Levantamos, ainda, a reflexão sobre o quanto a indústria e a academia desempenham suas atividades de forma separada. Isso dificulta, em nossa opinião, a evolução e o melhoramento de processos e de atividades, tanto no desenvolvimento de sistemas quanto no desenvolvimento de pesquisas. A academia, em nossa visão, poderia olhar mais para as necessidades da indústria e abrir seu leque de conceitos arquiteturais para realizar pesquisas e trabalhos que possam ser mais utilizados na indústria. Por sua vez, a indústria poderia dialogar mais com os pesquisadores da academia, buscando um trabalho mais colaborativo, visando a eficiência e qualidade de seus produtos. Em outros termos, podemos dizer que é uma "via de mão dupla", ou seja, uma precisa da outra.

Para a academia, além de levantar questionamentos sobre a forma de pesquisar o tema, nossos resultados também levantam reflexões em relação ao ambiente educacional, uma vez que os profissionais que desenvolvem os sistemas atualmente (e futuramente) eram (são) estudantes egressos dos cursos de graduação em Computação. Nessa perspectiva, indagamos: seria necessário empenhar mais esforço no ensino de arquitetura de *software* [35], para que fique claro para os profissionais as diferentes definições que o termo pode assumir, a importância e as formas de documentar e verificar uma arquitetura, bem como as ferramentas que podem dar suporte a essas atividades?

Em nossa opinião, sem um ensino efetivo e significativo sobre Arquitetura de *Software*, não tem como mudar o contexto industrial atual e querer que sejam realizadas as atividades arquiteturais de forma completa e eficaz. É oportuno que a academia reflita, também, sobre a formação dos profissionais, em como inserir a cultura arquitetural no ambiente profissional e formar, cada vez mais, profissionais conscientes e compromissados com a arquitetura do sistema que desenvolve.

7.2 Trabalhos Relacionados

Todos os artigos apresentados na seção 2.2 [41; 31; 20; 21; 6; 48; 25] nos serviram de base para planejar nossos estudos. Esses artigos se relacionam com nossa pesquisa a partir da metodologia utilizada. Com eles, aprendemos que combinar técnicas qualitativas com técnicas quantitativas traz uma maior confiança nos resultados. Além disso, é interessante citar trechos das entrevistas para corroborar a análise realizada pelo(s) pesquisador(es). Também é importante, a fim de adquirir maior confiança nos resultados, realizar a triangulação dos dados e obter um número e variabilidade satisfatórios de participantes. A utilização de técnicas para analisar as entrevistas também traz confiança para os resultados, além de ser importante explicar a metodologia utilizada e os motivos das decisões tomadas.

Além disso, os mesmos artigos [41; 31; 20; 21; 6; 48; 25] se relacionam com nossa pesquisa por procurarem entender o ambiente industrial, independente do foco principal do estudo. Enquanto nossa pesquisa objetivou entender o ambiente industrial com relação à arquitetura de *software* num contexto mais geral, os outros artigos objetivaram: entender como os desenvolvedores utilizavam o Twitter em seu contexto de trabalho [41], entender a

utilização de UML na prática [31], entender a visão dos profissionais sobre sistemas legados [20], entender a visão dos profissionais sobre os desafios da refatoração [21], entender o processo de verificação de conformidade arquitetural no Eclipse [6], entender como se dá o processo de tomada de decisão dos arquitetos profisionais [48] e entender o que a indústria precisa das linguagens arquiteturais [25].

No artigo "Bridging the Software Architecture Gap" [24], de Lindvall e Muthing, é apresentada a ferramenta SAVE. Este artigo se relaciona com nossa pesquisa pois, em sua contextualização, os autores afirmam algumas de nossas conclusões como a lacuna que existe entre a indústria e a academia, o fato de que nem toda empresa possui um arquiteto de *software*, a documentação não é sempre atualizada e aponta a prioridade dada para outras atividades devido ao tempo curto.

O artigo de Brunet *et al.*, "Five years of Software Architecture Checking: A Case Study of Eclipse" [6], se relaciona com nossa pesquisa (além das questões citadas anteriormente) pelo fato de haver a preocupação com a utilização de ferramentas de apoio à conformidade arquitetural na prática. O estudo apresentado neste artigo focou em entender as violações mais frequentes do Eclipse e como os desenvolvedores lidam com a presença delas. Para realizar o estudo, assim como o nosso, utilizaram um método misto analisando relatórios da ferramenta de verificação utilizada no processo de desenvolvimento e entrevistas com os desenvolvedores.

No artigo "Mature Architecting - A Survey about the Reasoning Process of Professional Architects" [48], Heesch *et al.* procuraram entender, a partir de um *survey*, como os arquitetos de *software* realizam suas tomadas de decisões, como priorizam os problemas, análises e soluções para a arquitetura. Este trabalho se relaciona com o nosso por procurar arquitetos profissionais para entender uma parte do contexto arquitetural na prática. Sua preocupação, semelhante à nossa, é que a academia possuía um baixo conhecimento sobre a tomada de decisões e, por esse motivo, não há ferramentas para auxiliar esta atividade. Assim como nosso estudo, eles perceberam que documentação arquitetural é uma atividade importante, é necessário compreender os requisitos e o problema que envolve o sistema e o tipo de arquitetura (simples ou complexa) depende de orçamento e tempo do projeto.

O artigo de Malavolta *et al.*, "What Industry Needs from Architectural Languages: A Survey" [25], se relaciona com nossa pesquisa (além da metodologia) por perceber uma lacuna

entre a teoria e a prática. Eles também percebem que são realizadas pesquisas, desenvolvidas definições, linguagens e ferramentas arquiteturais que muitas vezes possuem diferenças com relação à prática. Malavolta *et al.*, diferentemente do nosso estudo, focaram nas linguagens que foram desenvolvidas na academia para a realização de atividades arquiteturais e procuraram entender se elas satisfaziam as necessidades da indústria. Este artigo percebeu que as linguagens não suprem as principais necessidades da indústria (semelhante à nossa conclusão com relação às ferramentas de verificação). Assim como nosso estudo, este concluiu que, na prática, os profissionais não consideram necessário utilizar uma linguagem formal para documentar a arquitetura. Por fim, Malavolta *et al.* perceberam que entender o que os arquitetos de *software* precisam pode ajudar a direcionar os pesquisadores da área (assim como concluímos com nosso estudo).

7.3 Contribuições

Acreditamos que nenhuma de nossas considerações apresentadas nesta pesquisa seja uma surpresa para os pesquisadores da área, porém, contribuímos para a academia com um estudo empírico que formaliza algumas reflexões e percepções sobre a relação indústria x arquitetura de *software*.

Contribuímos com um estudo que disponibiliza uma noção e um entendimento (mesmo que superficial) sobre como os profissionais da indústria definem o termo "Arquitetura de *Software*", como eles documentam (se documentam), como verificam a conformidade arquitetural (se verificam), o que pensam sobre este assunto, bem como sua relação com as ferramentas de apoio à verificação de conformidade arquitetural propostas pela academia.

Consideramos como ponto mais relevante de nosso trabalho o fato de trazer mais uma percepção prática dos profissionais para dentro da academia. Compreendendo melhor o contexto prático sobre o tema, pode ser possível, para a academia, refletir sobre suas pesquisas visando gerar resultados que sejam cada vez mais utilizados nos processos de desenvolvimento de sistemas reais. Consideramos que academia e indústria devem caminhar juntas para evoluir e aperfeiçoar, cada vez mais, os processos de desenvolvimento de sistemas reais.

Por fim, com nosso estudo, muitas questões de pesquisa ficam em aberto para serem

pesquisadas em trabalhos futuros como, por exemplo: Como melhorar, se preciso for, o ensino de arquitetura de *software* nos cursos de graduação?; Por que muitos profissionais não conhecem o termo verificação de conformidade arquitetural?; Há alguma relação entre conhecimento teórico da equipe sobre arquitetura e a realização de atividades voltadas para a arquitetura do sistema?

7.4 Ameaças à validade

Em nosso estudo, temos a ameaça externa de não poder generalizar os resultados, uma vez que nossa amostra é pequena com relação a todos os profissionais da área. Para contornar esta situação, tivemos a preocupação de tentar selecionar profissionais de diferentes linguagens, lugares, tempo de experiência e de diferentes tamanhos de empresas.

Temos, ainda, a ameaça de conclusão, uma vez que tivemos problemas nos *e-mails* enviados na última etapa, diminuindo nossa possibilidade de diversidade de profissionais respondentes. Porém, mesmo com esse problema, tivemos uma boa variedade de linguagens de programação e países onde trabalhavam os respondentes da última etapa (diminuindo, assim, essa ameaça).

Também temos a ameaça interna de seleção de pessoas candidatas a responder nossas etapas. Consideramos que nossos critérios utilizados nas 3 (três) etapas foram suficientes para permitir uma variabilidade de profissionais respondentes, diminuindo, assim, essa ameaça.

Por fim, temos a ameaça de constructo, uma vez que nossos questionários e roteiros, apesar de terem sido planejados e revisados, podem ter sido construídos de forma que não tenha capturado toda a riqueza do nosso objeto de estudo. Além disso, nossos respondentes podem ter tentado responder a partir do que eles consideravam corretos para nossa pesquisa. Com relação a isso, tentamos ser o mais imparcial possível. Entretando, nossas análises e conclusões podem ter sofrido um viés com a visão de mundo do pesquisador, uma vez que temos nossos pré-conhecimentos do assunto (mais uma vez, tentamos ser imparciais e revisamos todas as conclusões e análises).

7.5 Trabalhos Futuros

Como trabalhos futuros temos um conjunto de questões de pesquisa que surgiram no decorrer da pesquisa e que podem gerar novos estudos e investigações.

Por exemplo, o que é preciso que uma ferramenta tenha para auxiliar a indústria na verificação de conformidade arquitetural? Ou, ainda, por que muitos profissionais não conhecem as ferramentas de verificação de conformidade arquitetural? E se todos os profissionais passam por problemas no desenvolvimento de sistemas reais, o que diferencia um profissional que documenta/verifica a arquitetura de um profissional que não documenta/verifica?

Também pode ser um trabalho futuro tentar entender por que muitos profissionais não sabem o que é verificação de conformidade arquitetural. Existe outro nome que eles conhecem ou eles realmente não sabem sobre isso? Ou, ainda, quais são os diversos termos/conceitos utilizados, na prática, nesta área? Os termos/conceitos realmente mudam de lugar para lugar? Se sim, quais são as diferenças?

Há alguma relação entre o conhecimento teórico arquitetural da equipe e a atividade de documentar/verificar a arquitetura? Também é uma questão a ser estudada se o nível dos problemas causados por violações arquiteturais tem alguma relação com documentar/verificar ou não a arquitetura do *software*. Ou, ainda, documentar/verificar a arquitetura interfere na frequência e forma de discussão da equipe sobre ela?

Continuando os questionamentos que surgiram com nosso estudo: por que parece existir uma distância entre o que a academia estuda e a forma como a indústria produz seus sistemas? Ou, por que o papel de arquiteto de *software*, na prática, não é bem definido? E, ainda, por que a documentação arquitetural tem prioridade menor que outras atividades?

Por fim, restam-nos as perguntas relacionadas ao ensino de arquitetura de *software*. Como a academia pode melhorar, se preciso for, o processo de ensino-aprendizagem sobre o tema arquitetura de *software*? Existe alguma relação entre o que se aprende na graduação e a forma que o profissional trata a arquitetura do sistema que desenvolve?

Bibliografia

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2003.

[2] V. Braun and V. Clarke. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology*, pages 77–101, 2006.

[3] V. Braun and V. Clarke. Teaching Thematic Analysis: Overcoming Challenges and Developing Strategies for Effective Learning. *The Psychologist*, pages 120–123, 2013.

[4] H.P. Breivold, I. Crnkovic, and M. Larsson. A Systematic Review of Software Architecture Evolution Research. *Information and Software Technology archive*, 54:16–40, 2012.

[5] J. Brunet, D. Guerrero, and J. Figueredo. Design Tests: An Approach to Programmatically Check you Code Against Design Rules. *In Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), New Ideas and Emerging Results*, May 2009.

[6] J. Brunet, G.C. Murphy, D. Serey, and J. Figueiredo. Five Years of Software Architecture Checking: A Case Study of Eclipse. *IEEE Software*, 2014.

[7] K. Charmaz. Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis. *SAGE Publications*, 2006.

[8] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, M. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, second edition, 2002.

[9] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures: Views and Beyond. *In Proceedings of the 25th International Conference on Software Engineering – IEEE*, pages 740–741, 2003.

[10] L. Cohen, L. Manion, and K. Morrison. *Research Methods in Education*. Routledge, sixth edition, 2007.

[11] T. Conte, R. Cabral, and G.H. Travassos. Aplicando Grounded Theory na Análise Qualitativa de um Estudo de Observação em Engenharia de Software: Um Relato de Experiência. *In V Workshop "Um Olhar Sociotécnico sobre a Engenharia de Software"(WOSES)*, pages 26–37, 2009.

[12] J.W. Creswell. Research Design: Qualitative, Quantitative, and Mixed Method Approaches. *SAGE Publications*, (2), 2003.

[13] Site de Paul Irish. http://www.paulirish.com/about/. Última visualização em 15/06/2015.

[14] Site de Tom Preston-Werner. http://tom.preston werner.com/. Última visualização em 15/06/2015.

[15] S. Ducasse and D. Pollet. Software Architecture Reconstruction: a Process-Oriented Taxonomy. *Software Engineering, IEEE Transactions on*, 35:573 – 591, 2009.

[16] M. Eichberg, S. Kloppenburg, K. Klose, and M. Mezini. Defining and Continuous Checking of Structural Program Dependencies. *In 30th International Conference on Software Engineering*, pages 391–400, 2008.

[17] D. Garlan and D. Perry. Introduction to the Special Issue on Software Architecture. *In IEEE Transactions on Software Engineering*, 21(4), 1995.

[18] B.G. Glaser and A.L. Strauss. The Discovery of Grounded Theory: Strategies for Qualitative Research. *New York: Aldine de Gruyter*, 1967.

[19] A. Jansen and J. Bosch. Software Architecture as a Set of Architectural Design Decisions. *In Proceedings of the 5th Working Conference on Software Architecture – IEEE*, pages 109–120, 2005.

[20] R. Khadka, B.V. Batlajery, A.M. Saeidi, S. Jansen, and J. Hage. How do Professionals Perceive Legacy Systems and Software Modernization? *In Proceedings of the 36th International Conference on Software Engineering*, pages 36–47, 2014.

[21] M. Kim, T. Zimmermann, and N. Nagappan. A Field Study of Refactoring Challenges and Benefits. *SIGSOFT*, 2012.

[22] J. Knodel, D. Muthig, M. Naab, and Mikael Lindvall. Static Evaluation of Software Architectures. *In 10th European Conference on Software Maintenance and Reengineering*, pages 279–294, 2006.

[23] J. Knodel and D. Popescu. A Comparison of Static Architecture Compliance Checking Approaches. *In IEEE/IFIP Working Conference on Software Architecture*, pages 44–53, 2007.

[24] M. Lindvall and D. Muthing. Bridging the Software Architecture Gap. *Computer*, pages 41(6):98–10, 2008.

[25] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang. What Industry Needs from Architectural Languages: A Survey. *Software Engineering, IEEE Transactions on (Volume:39 , Issue: 6)*, pages 869 – 891, 2013.

[26] I. Melo, J. Brunet, D. Guerrero, and J. Figueiredo. Verificação de Conformidade Arquitetural com Testes de Design – Um Estudo de Caso. *I Workshop Brasileiro de Visualização, Evolução e Manutenção de Software (VEM – CBSoft)*, 2013.

[27] I. Melo, D. Guerrero, and M.T. Valente. Uma Ferramenta para Verificação de Conformidade Visando Diferentes Percepções de Arquiteturas de Software. *Sessão de Ferramentas (CBSoft)*, 2014.

[28] G. Murphy, D. Notkin, and K. Sullivan. Software Reflexion Models: Bridging the Gap Between Design and Implementation. *Software Engineering, IEEE Transactions*, pages 27(4):364,380, 2001.

[29] D.L. Parnas. Software Aging. *In 16th International Conference on Software Engineering*, pages 279–287, 1994.

[30] D.E. Perry and A.L. Wolf. *Foundations for the Study of Software Architecture*. Software Engineering Notes, 4 edition, 1992.

[31] M. Petre. UML in Practice. *In 35th International Conference on Software Engineering*, pages 722–731, 2013.

[32] C.M. Pinto. A Teoria Fundamentada como Método de Pesquisa. *In XII Seminário Internacional em Letras*, 2012.

[33] W.F.N. Pires, J.A.M. Brunet, F.S. Ramalho, and D.S. Guerrero. UML-based Design Test Generation. *In 23nd Annual ACM Symposium on Applied Computing*, 2008.

[34] A. Postma. A Method for Module Architecture Verification and its Application on a Large Component-based System. *Information & Software Technology*, pages 171–194, 2003.

[35] C.S.C. Rodrigues and C.M.L. Werner. Uma Revisão Sistemática sobre as Iniciativas Realizadas no Ensino de Arquitetura de Software. *Rio de Janeiro: COPPE/UFRJ, ES-728/09*, 2009.

[36] P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14:131–164, April 2009.

[37] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using Dependency Models to Manage Complex Software Architecture. *In 20th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 167–176, 2005.

[38] C.B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 25(4), 1999.

[39] M. Shahin, P. Liang, and M.A. Babar. A Systematic Review of Software Architecture Visualization Techniques. *Journal of Systems and Software*, 94:161–185, 2014.

[40] C.A.C. Simoni and M.C.C. Baranauskas. Pesquisa Qualitativa em Sistemas de Informação. *Instituto de Computação (UNICAMP)*, 2003.

[41] L. Singer, F. Figueira Filho, and M. Storey. Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter. *In 36th International Conference on Software Engineering*, 2014.

[42] D.I.K. Sjoberg, T. Dyba, and M. Jorgensen. The Future of Empirical Methods in Software Engineering Research. *FOSE - Future of Software Engineering*, pages 358–378, 2007.

[43] Site sobre Linus Torvalds. http://www.techtudo.com.br/tudo-sobre/linus torvalds.html. Última visualização em 15/06/2015.

[44] A.L. Strauss and J.M. Corbin. Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory. *Sage Publications, Inc*, (2), 1998.

[45] J. Strauss, A.L.; Corbin. Basics of Qualitative Research: Grounded Theory. *Procedures and Techniques. Newbury: SAGE*, 1990.

[46] R. Terra. Conformação Arquitetural Utilizando Restrições de Dependência entre Módulos. *Dissertação de Mestrado. Belo Horizonte, Brasil*, 2009.

[47] R. Terra, L. Passos, M.T. Valente, R. Diniz, and N. Mendonça. Static Architecture Conformance Checking: An Illustrative Overview. *IEEE*, 2010.

[48] U. van Heesch and P. Avgeriou. Mature Architecting - A Survey About the Reasoning Process of Professional Architects. *In Software Architecture (WICSA), 9th Working IEEE/IFIP Conference on*, pages 260 – 269, 2011.

[49] J. Wainer. Métodos de Pesquisa Quantitativa e Qualitativa para a Ciência Computação. *Atualização em informática - Sociedade Brasileira de Computação e Editora PUC-Rio*, pages 221–262, 2007.

Apêndice A

Roteiro do Survey Exploratório

Primeiro, nós queremos conhecer um pouco de você!

1. Qual o seu nome?
2. Como podemos entrar em contato com você?
3. Há quanto tempo você trabalha na computação?
4. Em quais linguagens de programação você tem experiência?
5. Onde você trabalha atualmente?
6. Você tem alguma experiência com arquitetura de software?
7. Quais papéis você já desempenhou em projetos na área da computação?
    a. Opções: Desenvolvedor, Arquiteto de software, Designer de software, Engenheiro de software, Gerente de projeto, Gerente de TI, Analista de sistema, Engenheiro de teste, Outro (a citar).

Nos revele sua visão sobre arquitetura!

Apesar de existirem diferentes visões arquiteturais, nossa pesquisa é focada na visão estrutural de arquitetura de software, que costumamos definir da seguinte forma:

arquitetura de software = *componentes* + *dependências*

onde componentes são os principais módulos, camadas e/ou classes de um sistema. E dizemos que um componente A depende de um componente B se ele utiliza serviços de B (por meio de chamada de métodos, acesso a atributos, herança, etc).

Uma regra arquitetural é a formalização de decisões arquiteturais. Considerando 2 componentes hipotéticos A e B, são exemplos de regras arquiteturais:

→ “A não pode depender de B”
→ “A não pode herdar de B”
→ “A não pode implementar B”

1. Numa escala de 1 a 6, onde 1 significa “não relevante” e 6 significa “totalmente relevante”, qual o nível de relevância essa visão da arquitetura de software tem para o seu cotidiano profissional?
2. Quais outros aspectos da arquitetura de software que você considera relevantes e não são contemplados na definição acima?

Nos mostre mais detalhes sobre a arquitetura de um seus projetos!

1. Para responder as questões a seguir, pedimos que você considere um sistema em que você esteja trabalhando ou em que já tenha atuado. Poderia nos informar qual o nome desse sistema?

2. Você diria que a arquitetura desse sistema é bem documentada?
    a. SIM
        i. Que tipos de documentos são usados no projeto para registrar a arquitetura de software?
        ii. No sistema citado em questão anterior, há/houve mudança nas regras arquiteturais ao longo do desenvolvimento e/ou evolução?
        iii. Numa escala de 1 a 6, onde 1 significa “nunca” e 6 significa “sempre”, com que frequência a equipe atualiza a documentação de arquitetura de software?
    b. NÃO
        i. Por que a arquitetura não é bem documentada?
3. Numa escala de 1 a 6, onde 1 significa “a maioria dos desenvolvedores desconhece as regras” e 6 significa “todos os desenvolvedores têm plena consciência das regras”, qual o nível de conhecimento da equipe sobre as regras arquiteturais desse sistema em questão?
4. Se um novo desenvolvedor entrar na equipe, como ele faz para conhecer as principais regras arquiteturais desse sistema em questão?
5. Quantas pessoas participam da tomada de decisões arquiteturais?
    a. Opções: Todos os desenvolvedores, Apenas um grupo pequeno, Apenas os seniores, Apenas 1 pessoa.
6. Numa escala de 1 a 6, onde 1 significa “nunca” e 6 significa “sempre”, com que frequência a equipe discute sobre arquitetura?
7. Onde/como se discute sobre arquitetura em seu projeto?
    a. Opções: Email, Conversas presenciais, Chats, Comentários/anotações no código.

Nos revele sua visão sobre verificação de conformidade arquitetural!

Costumamos dizer que, na prática, todo sistema tem duas arquiteturas:

- a planejada: a arquitetura idealizada pela equipe de arquitetura (e que talvez esteja documentada);
- e a implementada: a arquitetura efetivamente implementada no código fonte do sistema.

Verificação de conformidade arquitetural é a tarefa de verificar se a arquitetura implementada é consistente em relação às regras que constam na arquitetura planejada. As divergências entre as duas arquiteturas são chamadas de violações arquiteturais.

1. Você já teve problemas devido a violações arquiteturais no seu projeto?
2. Em caso afirmativo, que problemas enfrentados pelo projeto você atribui às violações arquiteturais? Ou quais consequências você atribuiria às violações?
3. No projeto em questão, há atividades cujo propósito seja a verificação de conformidade arquitetural?

a. SIM
    i. No projeto em questão, com que periodicidade é realizada a verificação de conformidade arquitetural?
        1. Opções: Diariamente, 1 vez por semana, 1 vez em 15 dias, 1 vez no mês.
    ii. Utilizou alguma ferramenta de apoio para realizar a verificação de conformidade arquitetural?
        1. SIM
            a. Que ferramentas sua equipe utiliza ou já utilizou?
            b. Numa escala de 1 a 6, onde 1 significa "insatisfeito" e 6 significa "muito satisfeito", qual o grau de satisfação com a(s) ferramenta(s) utilizada(s)?
            c. Quais as principais limitações das ferramentas utilizadas?
        2. NÃO
            a. Por que não foi utilizada nenhuma ferramenta de apoio?
            b. Você conhece alguma técnica ou ferramenta de verificação de conformidade arquitetural? Se sim, qual(is)?
b. NÃO
    i. Por que não há atividades de verificação de conformidade arquitetural no projeto?

Quem você nos indicaria para contactar?

1. Você poderia nos indicar profissionais de computação que você julgue interessantes para responder este questionário?
2. Você acha que seria apropriado/interessante para a nossa pesquisa enviar este questionário para sua lista de seu projeto/empresa? Ou algum outro projeto?

Por fim ...

1. Considerando o exposto, você teria algum comentário que considere relevante para nossa pesquisa?

Agradecemos a disponibilidade em responder esse questionário!

Apêndice B

Roteiro de Entrevista

1. Por quanto tempo você foi arquiteto/designer de software?
2. Em que país você trabalha atualmente?
3. Qual é a sua definição para arquitetura de software?
    a. *Se fosse necessário (dependendo da resposta anterior):* Algumas pessoas, no survey, afirmaram que aspectos de negócios, requisitos não-funcionais, tecnologias utilizadas e aspectos de implantação e integração fazem parte da definição de arquitetura de software. Você concorda com isso? Por que?
4. Você considera importante documentar a arquitetura do sistema? Por que?
5. Você documenta a arquitetura de seus sistemas?
    a. SIM
        i. Que aspectos arquiteturais você documenta?
        ii. Como documenta cada aspecto? Você utiliza alguma notação especial?
        iii. *Se tiver citado algum aspecto na definição de arquitetura que ele não tenha citado nesta etapa:* Por que você não documenta este aspecto?
    b. NÃO
        i. Por que?
6. Quando a arquitetura sofre modificação, como os membros da equipe tomam conhecimento disso?
7. O que é verificação de conformidade arquitetural para você?
8. Você acha que verificação de conformidade arquitetural é uma atividade importante para garantia de qualidade do sistema?
9. Você realiza verificação de conformidade arquitetural?
    a. SIM
        i. Que aspectos arquiteturais você verifica?
        ii. Como verifica?
        iii. *Se tiver citado algum aspecto na definição de arquitetura que ele não tenha citado nesta etapa:* Por que você não verifica este aspecto?
    b. NÃO
        i. Por que?
10. Você acha que o acúmulo de violações arquiteturais pode acabar com o sistema?
11. Você conhece alguma ferramenta de apoio à verificação de conformidade arquitetural?
    a. SIM
        i. Você acha que essa(s) ferramenta(s) consegue(m) verificar todos os aspectos de arquitetura de software?
        ii. Você acha fácil utilizar a ferramenta?
12. Você concorda que as ferramentas de apoio à verificação de conformidade arquitetural são pouco conhecidas e pouco usadas? Por que?
13. Deseja fazer algum comentário que considere relevante?

Apêndice C

Códigos da análise das entrevistas

Os grafos a seguir foram todos gerados na ferramenta nVivo 10.

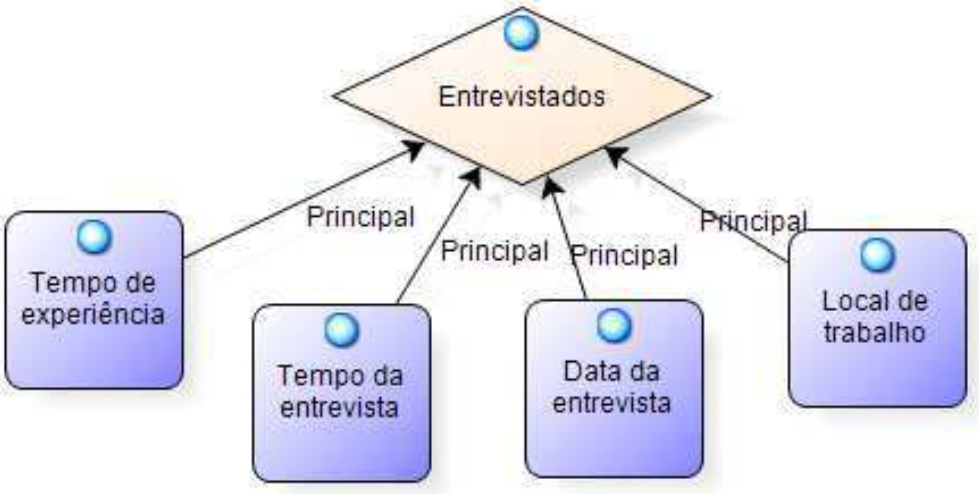

Figura C.1: Códigos de contexto.

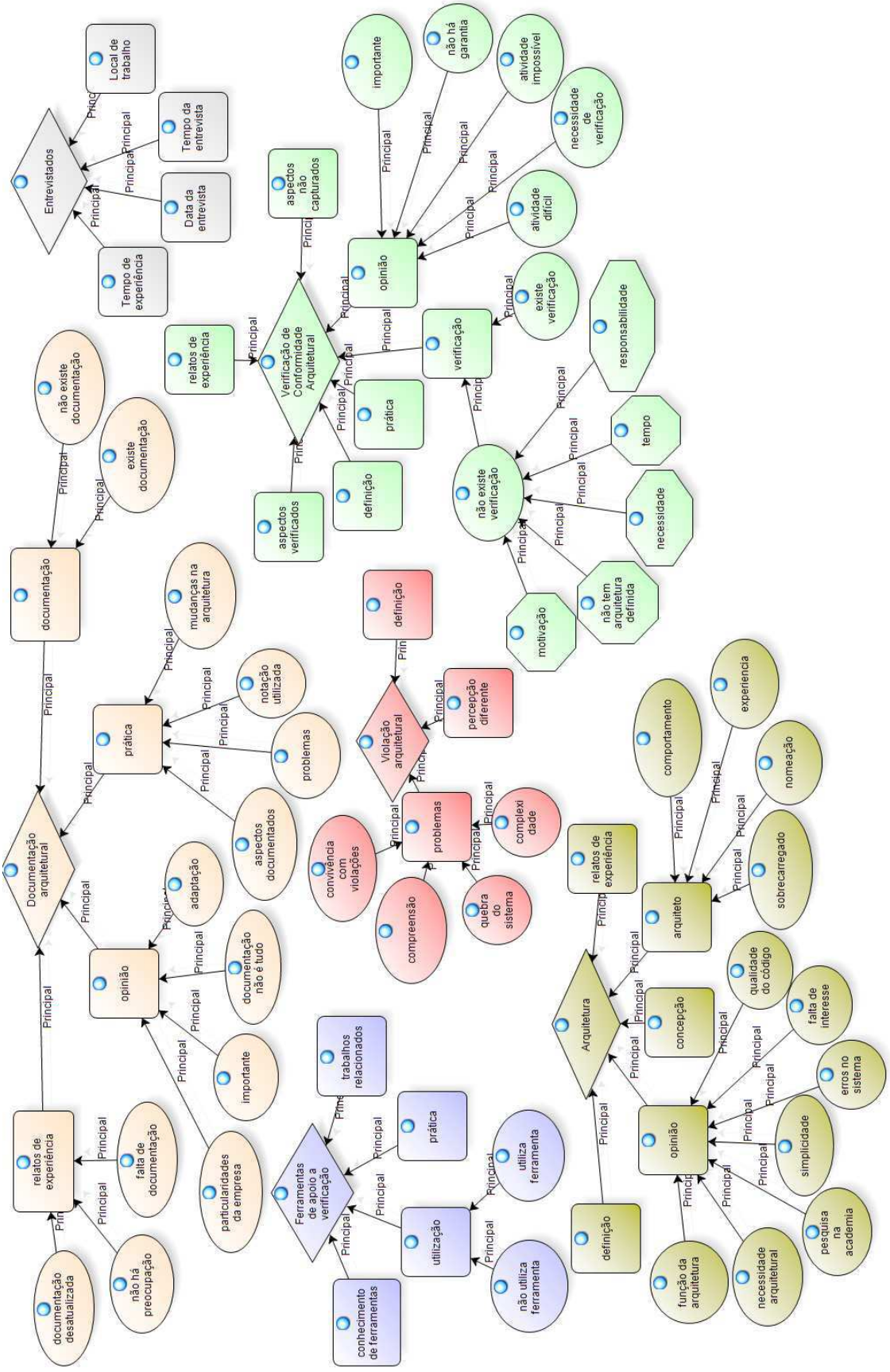

Figura C.2: Modelo com os principais códigos gerados no nVivo.

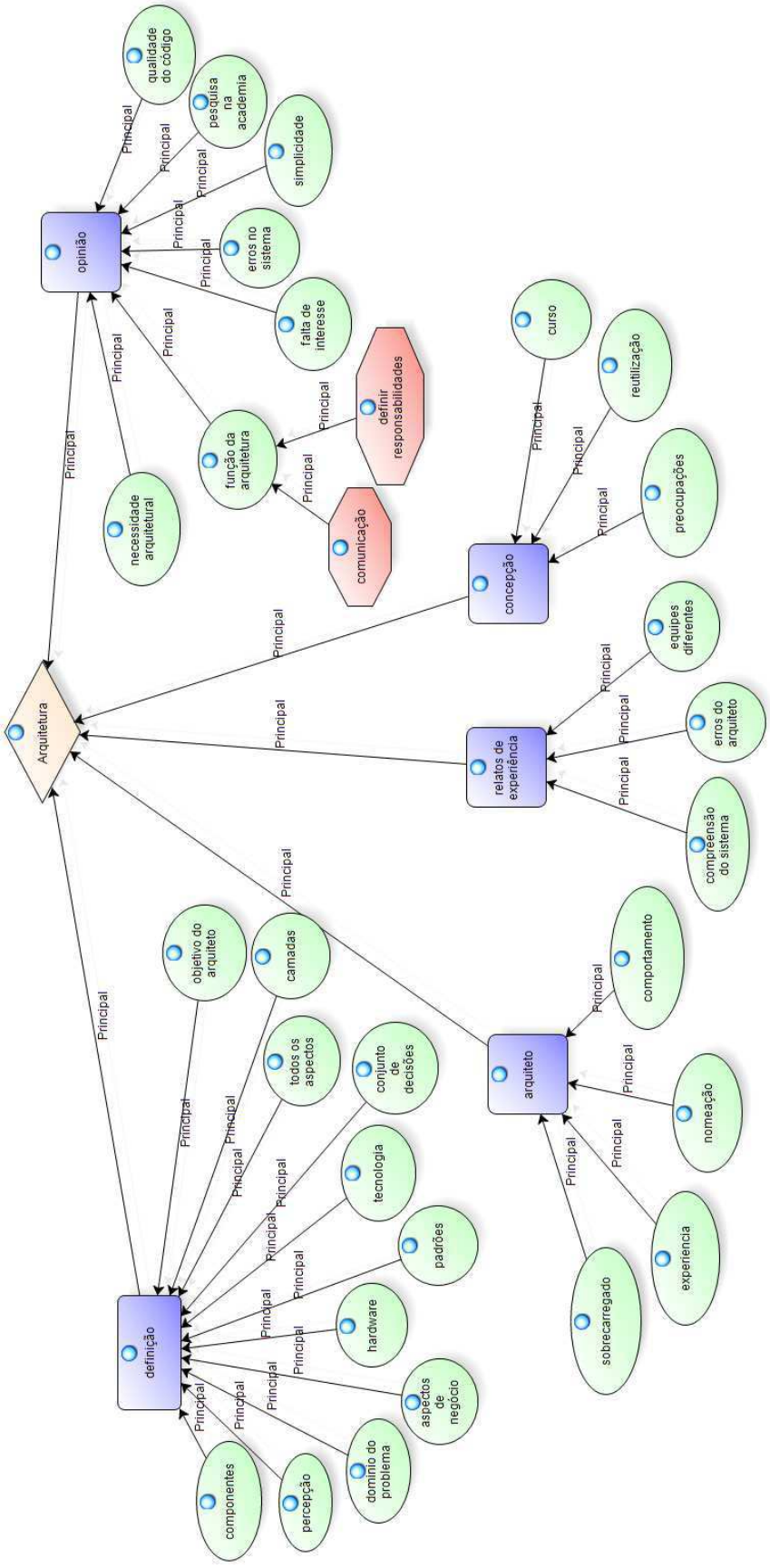

Figura C.3: Modelo com os códigos do tema Arquitetura de Software no Geral.

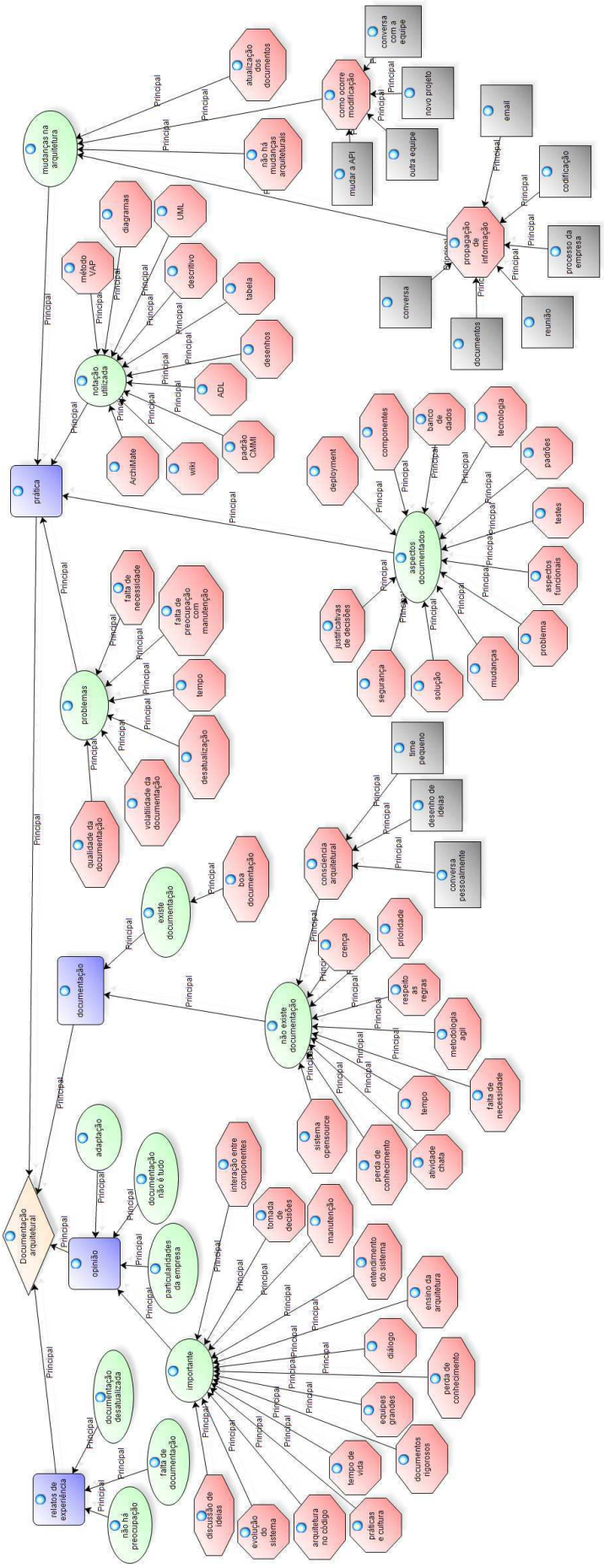

Figura C.4: Modelo com os códigos do tema Documentação arquitetural.

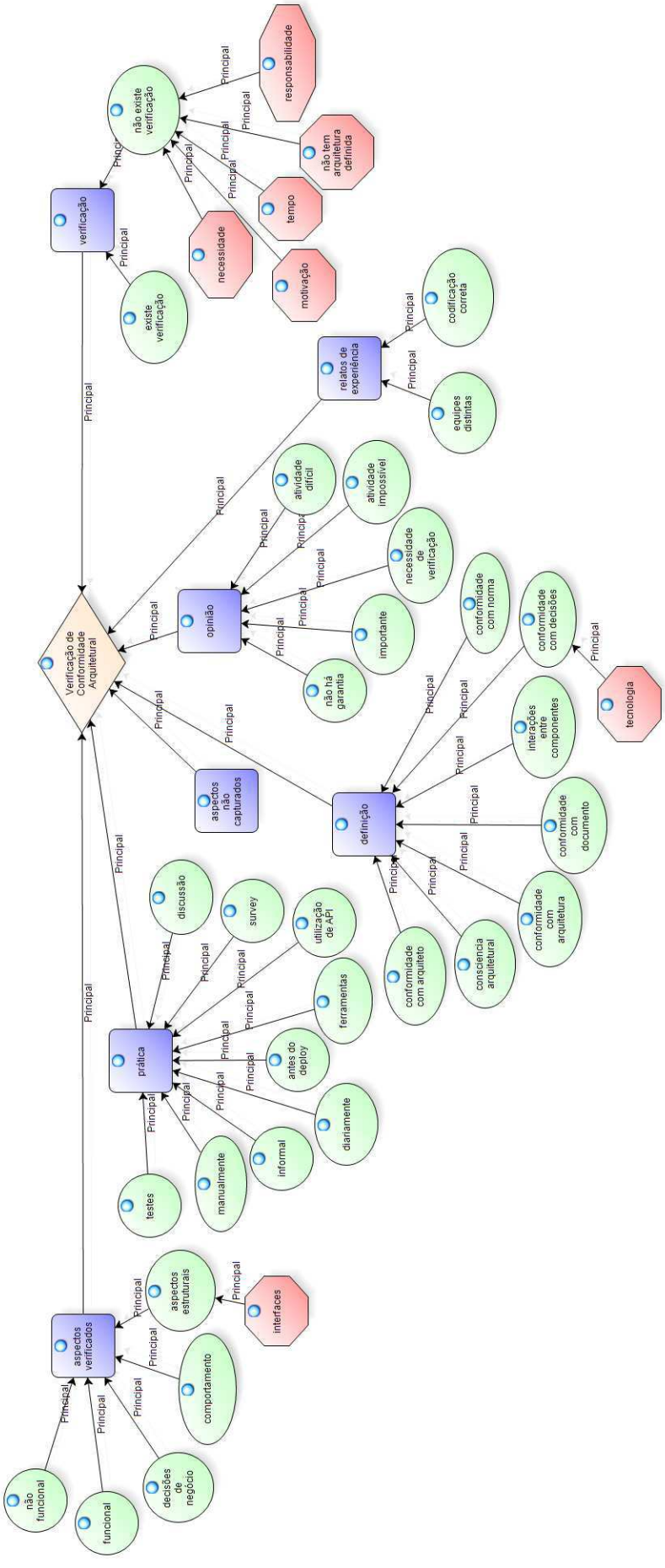

Figura C.5: Modelo com os códigos do tema Verificação de conformidade arquitetural.

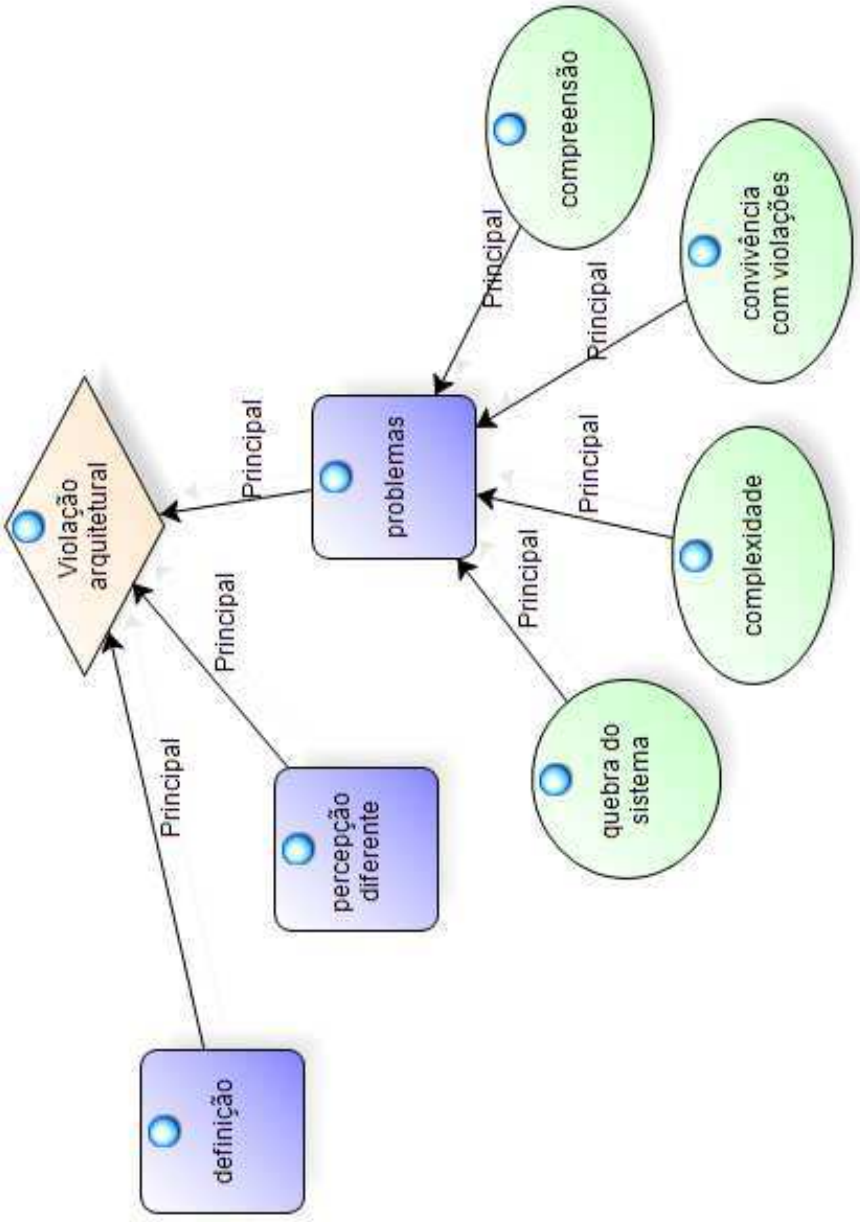

Figura C.6: Modelo com os códigos do tema Violações arquiteturais.

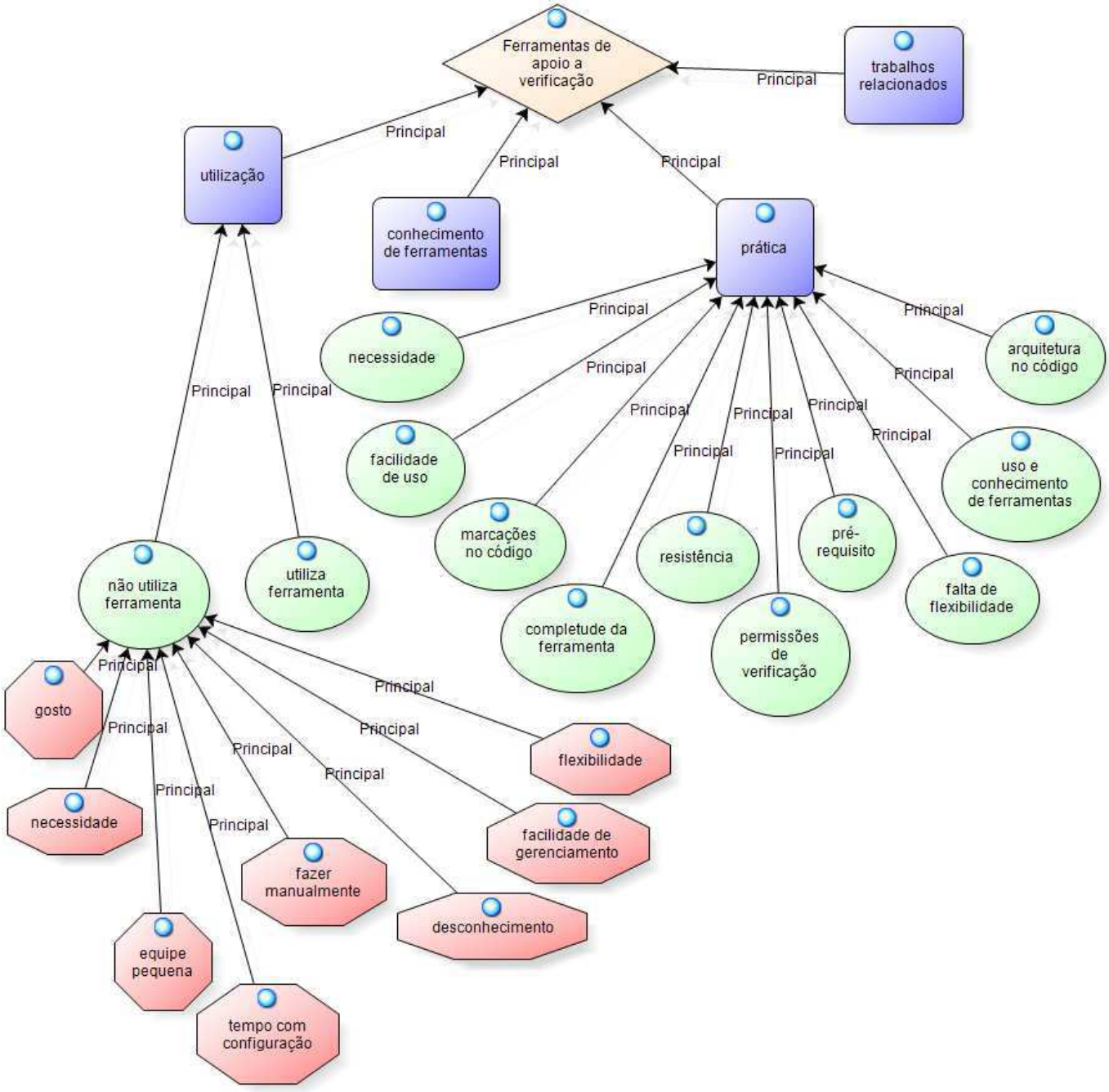

Figura C.7: Modelo com os códigos do tema Ferramentas de apoio a verificação de conformidade arquitetural.

Apêndice D

Roteiro do Survey Confirmatório

Rank from 1 to 5 according to your level of agreement.

1 - strongly disagree
2 - disagree
3 - no opinion about the statement
4 - agree
5 - strongly agree

About you

1. What's your name?
2. In which city do you work?
3. In which programming languages do you have experience? *(You can pick more than one)*
    a. Options: Java, C, C++, C\#, Python, PHP, Ruby, JavaScript, Other.

About the definition of Software Architecture

1. There's no single definition for software architecture.
2. Which aspects are part of a software architecture?
    a. Static components (e.g. modules, packages, and subsystems)
    b. Dynamic components (e.g. processes, clients, and servers)
    c. Business aspects
    d. Hardware infrastructure, network, etc.
    e. Software infrastructure (e.g. database, programming language, and middleware)

About architectural documentation

1. It is important to document architectural aspects.
2. Detailed architectural documents must be available:
    a. Never
    b. In some projects
    c. In most projects
    d. Always
3. The most common way of documenting the architecture is in plain text.
4. Architectural documents, when existing, are usually not updated.
5. Non-updated architectural documents negatively impact on developers' productivity:
    a. This is never true
    b. This is true for some projects
    c. This is true for most projects
    d. This is always true

About Architectural Conformance Checking
(i.e. the task of checking whether the code follows the architecture)

1. It is important to perform architectural conformance checking:
    a. Never
    b. In some projects
    c. In most projects
    d. Always
2. Architectural conformance checking, when applied, is usually performed without tool support
3. Architectural conformance checking is accurately done:
    a. Never
    b. In some projects
    c. In most projects
    d. Always
4. Architectural conformance checking is not enough to ensure code quality.

About Architectural Conformance Checking tools

1. Architectural conformance checking tools are rarely used.
2. Existing architectural conformance checking tools are complex and require detailed specifications.
3. Architectural conformance checking tools should be used:
    a. Never
    b. In some projects
    c. In most projects
    d. Always
4. Most developers resist to use architectural conformance checking tools.

Just 3 more statements

1. The role of software architects varies widely among projects.
2. Architectural violations (i.e code that does not follow the architecture decisions) have a negative impact on code quality.
3. Academy and industry do not have mutual interest in synchronizing their working agendas in software architecture.

Comments

1. Would you have any comments that you consider relevant for our research?

Thanks for answering this survey!