RAFAEL PEREIRA MARTINS AZEVEDO

SELEÇÃO DE PADRÕES PARA A ARQUITETURA DE SOFTWARE: UMA ABORDAGEM BASEADA EM PROCURA DE TERMOS E SINÔNIMOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de Magister Scientiae.

VIÇOSA

MINAS GERAIS - BRASIL

2014

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
A994s
2014

Azevedo, Rafael, 1990-
Seleção de padrões para a arquitetura de software : uma abordagem baseada em procura de termos e sinônimos / Rafael Azevedo. – Viçosa, MG, 2014.
92f. : il. (algumas color.) ; 29 cm.

Inclui apêndices.
Orientador: José Luis Braga.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 87-92.

1. Engenharia de software. 2. Software - Desenvolvimento. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 005.12

RAFAEL PEREIRA MARTINS AZEVEDO

SELEÇÃO DE PADRÕES PARA A ARQUITETURA DE SOFTWARE: UMA ABORDAGEM BASEADA EM PROCURA DE TERMOS E SINÔNIMOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 16 de setembro de 2014.

André Luiz de Castro Leal

Jugurta Lisboa Filho

Antônio de Pádua Albuquerque Oliveira

José Luís Braga
Orientador

Dedico essa dissertação à minha mãe
Denilda.

AGRADECIMENTOS

Agradeço a todos os amigos que me acompanharam até aqui. Aos meus familiares, em especial, minha mãe. Agradeço também aos muitos padrinhos que fui adquirindo nesta estrada e que permitiram que eu viesse a conseguir meus objetivos.

Por último, agradeço aos meus orientadores por aceitarem me orientar e fazê-lo com tanto esmero. Obrigado pela oportunidade de trabalhar com vocês.

BIOGRAFIA

RAFAEL PEREIRA MARTINS AZEVEDO, filho de José Carlos de Azevedo e Denilda Pereira Martins, brasileiro nascido em 5 de março de 1990 na cidade de Brasília, no Distrito Federal.

No ano de 2008, após concluir o ensino médio na cidade de Buritis-MG, ingressou no curso de graduação em Ciência da Computação na Universidade Federal de Viçosa, concluído no ano de 2012.

Neste mesmo ano foi aprovado na seleção do programa de pós-graduação do Departamento de Informática – DPI, onde, em março de 2012, iniciou o mestrado em Ciência da Computação na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em setembro 2014.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

RESUMO

AZEVEDO, Rafael Pereira Martins, M.Sc., Universidade Federal de Viçosa, setembro de 2014. **Seleção de padrões para a arquitetura de software: uma abordagem baseada em procura de termos e sinônimos**. Orientador: José Luís Braga.

A construção da arquitetura a partir dos requisitos do software é uma atividade que exige um grau elevado de competência, dado que as decisões tomadas neste processo afetam todos os ciclos posteriores do projeto. Uma arquitetura de software bem projetada maximiza o grau de atendimento aos requisitos do sistema sendo construído. A escolha de soluções compatíveis com o problema a ser resolvido é uma das chaves para o sucesso de uma arquitetura de software. O reuso de soluções usadas com sucesso previamente em problemas semelhantes reduz riscos e aumenta a qualidade da arquitetura. Padrões arquiteturais documentam soluções arquiteturais para problemas recorrentes. A enorme quantidade de padrões somada à quantidade de informação contida na descrição dos mesmos e à inviabilidade de um desenvolvedor de software saber todas estas informações ou adquiri-las em pouco tempo são algumas das motivações para o desenvolvimento de técnicas, métodos e ferramentas que auxiliem na seleção de padrões mais adequados a cada tipo de sistema. O objetivo deste trabalho é propor uma abordagem para gerar recomendações de padrões mais adequados a cada tipo de sistema. As recomendações são baseadas na ocorrência de termos chave na descrição dos padrões. Os passos metodológicos para a construção da abordagem envolvem: a definição dos tipos de informação envolvidos nas decisões sobre a arquitetura do software e que consistirão nos termos chaves; a construção de um conjunto inicial de termos chave proveniente da literatura, a definição de como será realizada a procura destes termos nas descrições de padrões de software e a avaliação dos resultados obtidos a partir do uso da abordagem. Os resultados obtidos foram analisados utilizando-se métricas e comparações com recomendações presentes na literatura e indicaram um desempenho satisfatório da abordagem.

ABSTRACT

AZEVEDO, Rafael Pereira Martins, M.Sc., Universidade Federal de Viçosa, September, 2014. **Selection of software architecture patterns: an approach based on term searching and synonymy**. Adviser: José Luís Braga.

The transformation of requirements into software architecture is an activity that requires a high degree of competence, given that the decisions made in this process affect all subsequent cycles of the project. A well-designed software architecture maximizes the degree of compliance with the requirements of the system. The choice of compatible solutions for the problem is a key to the success of a software architecture. The reuse of solutions previously used successfully in similar problems reduces risk and increases the quality of the architecture. Architectural patterns follows this idea once they are used to document architectural solutions to recurring problems. The huge amount of patterns, the amount of information embodied into their description and the unfeasibility of a software developer to know all this information, or buy them in a short time are some of the motivations to develop techniques, methods and tools that assist in selection of the most appropriate pattern for each kind of system. Our objective is defining an approach to generate appropriated recommendations according to the requirements of a proposed system. These recommendations are based on the occurrence of key terms used to both describe software problems and describe patterns. The methodological steps for the construction of the approach includes defining what kind of information may be involved in software architecture decisions, constructing an initial set of key terms from the literature, defining how the search may be conducted in accordance with these descriptions of software patterns and evaluating the results obtained from the use of the approach. The results were analyzed using metrics and comparisons with recommendations in the literature and indicated satisfactory performance of the approach.

1 INTRODUÇÃO

A tecnologia da informação é uma área em ascensão e caracteriza-se por ser uma área de rápida inovação. Para sobreviver neste ambiente, as empresas de software necessitam produzir sistemas de alta qualidade, com custo mínimo e em tempo hábil (Harter, Krishnan, Slaugther, 2000), sendo que problemas no desenvolvimento e uso do software acarretam riscos de perdas a taxas inaceitáveis (Boehm, Sullivan, 1999).

A arquitetura de software, observada como disciplina, surgiu na agenda de pesquisa em engenharia de software exercendo papel de alta importância no atendimento à tríade de metas de negócio inerentes à produção de qualquer sistema - qualidade, custo e tempo. Bass, Clements e Kazman (2003) e (IEEE, 2000) concordam que a arquitetura de software trata da organização fundamental do sistema, a qual compreende os componentes de software, relacionamentos entre eles, ambiente e princípios que guiam seu projeto e evolução. Trata-se do arcabouço sobre o qual o software será construído. A definição da arquitetura de um sistema envolve a tomada de decisões de projeto cujo efeito se propaga por todas as demais fases do ciclo de vida de desenvolvimento de software.

A arquitetura de um sistema é projetada durante as fases iniciais do processo de desenvolvimento e atua facilitando ou restringindo o atendimento a requisitos funcionais, não funcionais, e metas de negócio. Decisões arquiteturais possuem impacto direto em projetos de software de grande porte (Falessi et al., 2011) (Gorton, 2011).

Validar decisões arquiteturais nas fases iniciais do processo de desenvolvimento de software é difícil e custoso. Para reduzir os riscos embutidos nestas decisões, arquitetos de software confiam em soluções já testadas para resolver certas classes de problemas. Neste contexto, padrões arquiteturais e de projeto atuam diminuindo risco, intermediando soluções de sucesso com conhecidos atributos de engenharia (Gorton 2011).

1.1 Motivação

Durante o projeto de arquitetura, múltiplas soluções costumam ser candidatas, sendo algumas rejeitadas rapidamente e outras afirmadas como fortes opções. Escolher entre diferentes soluções de forma racional constitui um dos maiores desafios enfrentado pelo arquiteto de software (Bass, Clements, Kazman, 2003).

O crescente número de padrões documentados na literatura e disponíveis em repositórios online torna inviável a seleção de padrões manualmente. A título de exemplo, Henninger e Corrêa (2007) identificaram 2241 padrões com soluções para problemas de projeto de software. Este problema torna-se particularmente difícil de resolver para desenvolvedores inexperientes. De acordo com Sommerville (2010) somente engenheiros de software experientes que possuem conhecimento aprofundado sobre padrões, são capazes de utilizá-los de forma eficaz. Estes profissionais são capazes de reconhecer situações genéricas onde um padrão pode ser aplicado. Desenvolvedores inexperientes, mesmo lendo livros sobre padrões, sempre esbarrarão na dificuldade de decidir entre reusar um padrão ou desenvolver uma solução de propósito específico. Soares et al. (2011) complementam dizendo que no geral, somente arquitetos experientes são preparados o suficiente para selecionar o padrão mais adequado para cada tipo de problema.

O Capítulo 2 apresenta uma série de trabalhos cujo objetivo é facilitar a seleção de padrões para arquitetura de software, porém, uma análise aprofundada destes trabalhos evidencia que as abordagens propostas preconizam atividades que fogem do ciclo de vida dos processos de desenvolvimento de software mais utilizados pela indústria. Exemplos dessas atividades seriam a definição de requisitos em notações específicas, resposta a questionários e etc. Além disso, o conhecimento contido nos padrões precisa ser processado e reelaborado de forma que a abordagem possa aproveitar melhor o conhecimento encapsulado pelo padrão. Isto demanda atividade manual de especialistas, atividade esta que deve se encerrar após o lançamento das primeiras versões da ferramenta. Isto não atende ao contexto de crescente aumento dos padrões documentados na literatura e dos parâmetros de decisão a serem considerados durante o desenvolvimento de software conforme novos problemas vão surgindo.

Este trabalho propõe uma abordagem para o problema da escolha dos padrões mais adequados a serem utilizados na composição da arquitetura de um sistema. Os

requisitos fundamentais para a abordagem proposta consistem na adequabilidade da mesma ao ciclo de vida do processo de desenvolvimento de software e na facilidade de se estendê-la de modo a considerar novos critérios e novos padrões na tomada de decisão.

1.2 Hipótese

A extração de conhecimento da descrição de padrões utilizando termos chaves pré-definidos e a utilização destes termos como entrada para a seleção de padrões produz recomendações de padrões arquiteturais apropriadas ao contexto de cada projeto.

1.3 Objetivos

O objetivo geral deste trabalho é desenvolver uma abordagem baseada em conhecimento que utilize termos chave como entradas para a seleção de padrões.
Especificamente pretende-se:

Elaborar uma base de conhecimento que auxilie o processo de decisão em arquitetura de software contendo padrões de software e elementos que possam ser utilizados para parametrizar a seleção destes padrões;

Analisar a descrição dos padrões em busca de elementos que os associem com contextos de projetos de software e seus requisitos;

Desenvolver uma abordagem para a seleção de padrões mais apropriados ao contexto de cada projeto, utilizando os elementos identificados;

Desenvolver uma prova de conceito que demonstre a efetividade da abordagem quanto à extração de conhecimento e indicação de padrões em projetos de software;

1.4 Metodologia

O diagrama apresentado na Figura 1.1 apresenta as atividades realizadas durante a concepção da abordagem apresentada neste trabalho.

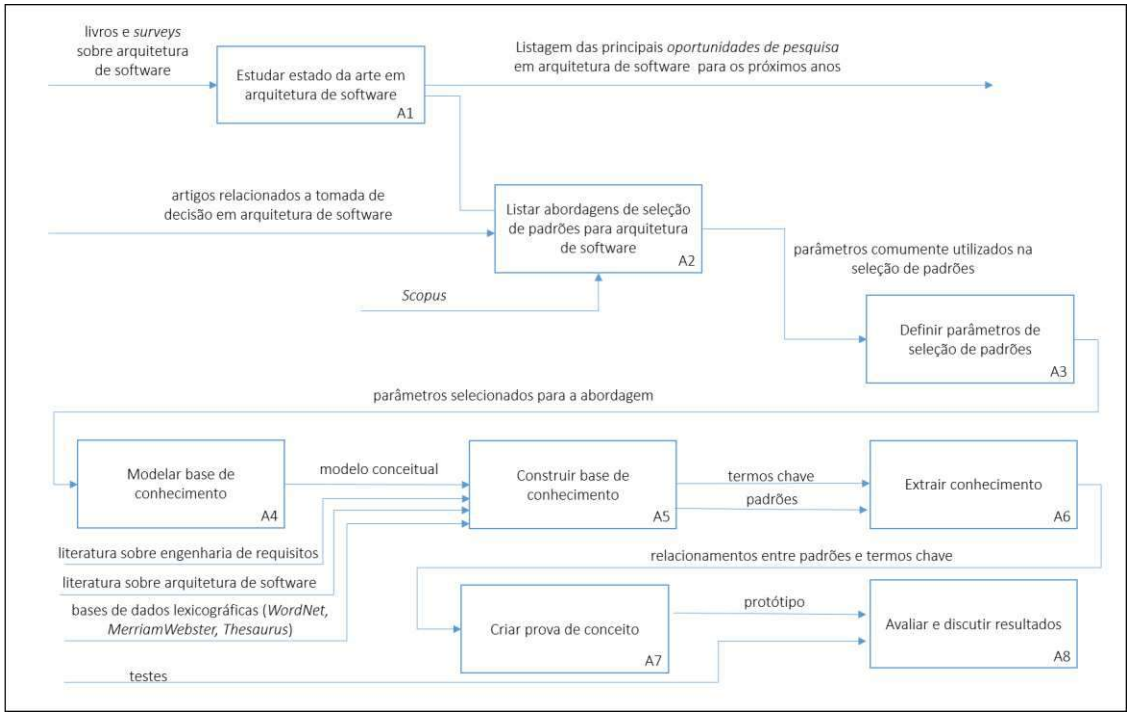

Figura 1.1. Diagrama SADT com as atividades executadas durante a concepção da abordagem

1. **Realização de estudo sobre o estado da arte em arquitetura de software.**
   Inicialmente, os esforços foram concentrados no entendimento da arquitetura de software como um todo, em suas várias vertentes e o esclarecimento quanto à forma como decisões são tomadas em arquitetura de software;
2. **Realização de estudo sobre abordagens já existentes para procura e seleção de padrões.**
   Realizou-se então um estudo aprofundado sobre abordagens de seleção de padrões propostas na literatura desde a emergência da arquitetura de software como disciplina. O estudo buscou evidenciar os pontos fortes e pontos fracos de cada abordagem e serviu como base para a definição das prioridades da abordagem proposta neste trabalho;
3. **Definição dos parâmetros para seleção de padrões a serem utilizados na abordagem.**
   Nesta etapa, foi definido que tipo de conhecimento deveria ser utilizado pela abordagem para indicar padrões para projetos de software. Realizou-se uma pesquisa sobre os conceitos embutidos nas abordagens já propostas e na literatura sobre tomada de decisão em arquitetura de software com o objetivo de

embasar a escolha do tipo de conhecimento que seria utilizado nesta abordagem;

4. **Modelagem da base de conhecimento.**

   Durante esta fase, definiu-se um modelo conceitual da abordagem composto de todos os tipos de conhecimento usados na seleção de padrões e que foram definidos na fase anterior, além do próprio conceito de padrão. Os produtos obtidos nesta fase incluem um template de descrição de padrões adequado aos objetivos da abordagem e compatível com os templates comumente utilizados na literatura, além de uma hierarquia de conceitos nos quais os termos chave posteriormente levantados deveriam se encaixar. Determinou-se que esta hierarquia seria a base tanto da extração de conhecimento das descrições de padrões quanto a base da entrada utilizada para obter recomendações;

5. **Construção da base de conhecimento.**

   A modelagem conceitual concebida na fase anterior foi utilizada como arcabouço para a construção de uma base de conhecimento de apoio a seleção de padrões. Esta base de conhecimento foi construída através da investigação ostensiva dos conceitos envolvidos no modelo conceitual na literatura de arquitetura de software, padrões para arquitetura e engenharia de requisitos e em bases léxicas tais como WordNet Search - 3.1, Merriam Webster Online e Thesaurus.com. Os produtos desta fase incluem uma listagem de trabalhos que lidam com requisitos de qualidade e táticas acompanhada da explicação de como os conceitos envolvidos nestes trabalhos comunicam-se com os conceitos da abordagem proposta além da própria base de termos chave em si;

6. **Extração de conhecimento.**

   Durante esta fase, foram definidas quais seriam as técnicas usadas para extração de informação e processamento natural de linguagem, necessárias para a detecção das relações semânticas entre os conceitos contidos na base de conhecimento;

7. **Prova de conceito.**

   Durante esta etapa, como prova de conceito, construiu-se um protótipo que implementa a abordagem. O protótipo é capaz de extrair conhecimento de padrões e, a partir da descrição de requisitos de um sistema de software, recomendar um conjunto de padrões mais adequados ao problema em questão;

8. **Avaliação e discussão dos resultados.**

   Esta fase consistiu na aplicação de técnicas de validação sobre os resultados obtidos através do protótipo. Os resultados obtidos foram analisados e também, comparados com resultados descritos em trabalhos da literatura sobre padrões e suas aplicações a contextos de software.

1.5 Organização desse documento

O restante desta dissertação está estruturada da seguinte maneira: o Capítulo 2 apresenta o referencial teórico deste trabalho, descreve o estado da arte da arquitetura de software discutindo o problema da tomada de decisão em arquitetura e mostrando como este problema têm sido enfrentado. É descrito também, um conjunto de abordagens já concebidas para seleção de padrões para arquitetura.

O Capítulo 3 discorre sobre a construção da base de conhecimento, apresentando o modelo conceitual utilizado, acompanhado de todo o embasamento científico utilizado para compor o modelo. Além disso, o processo de povoamento da base de dados de apoio à seleção de padrões é exposto juntamente com a listagem de todas as fontes consultadas para compor a referida base.

O Capítulo 4 descreve como a abordagem utiliza a base de conhecimento criada para gerar recomendações de padrões para arquitetura de software.

O Capítulo 5 apresenta uma discussão sobre os resultados obtidos através da aplicação do protótipo a alguns contextos de projeto de software.

Por fim, o Capítulo 6 apresenta as conclusões e comentários finais sobre o problema da seleção de padrões e sobre a abordagem proposta. Uma listagem de trabalhos futuros e oportunidades de pesquisa é apresentada.

2 REFERENCIAL TEÓRICO

2.1 Qualidade do Software

Durante o processo de desenvolvimento de software é fundamental que o aspecto qualidade seja considerado um fator crítico para o sucesso do projeto. Qualidade de software pode ser entendida como sendo a conformidade do sistema com requisitos funcionais e de desempenho pertinentes ao projeto em desenvolvimento.

Algumas características implícitas são desejáveis em todo software e indicam o grau de qualidade de um sistema. Entre elas aspectos como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade destacam-se como as principais.

Mas como construir software que apresente características de qualidade de alto-nível? Segundo Dromey (1996) não é possível construir atributos de qualidade de alto nível dentro do software. O que é possível fazer é ajustar propriedades do processo e do produto de modo que reflitam nas características de qualidade desejadas em grau de intensidade que pode variar conforme as prioridades do cliente. O mesmo autor completa dizendo que o conceito de qualidade de software têm sido utilizado livremente tanto no contexto de processos quanto de produto de software. Isto tem criado confusão e desviado a indústria da meta principal: melhorar a qualidade de produtos das várias fases do desenvolvimento de software. Tomando por base este viés, este trabalho insere-se no contexto de melhorar a qualidade do produto da fase de desenvolvimento da arquitetura de software.

2.2 Arquitetura de Software

2.2.1 Definição

A disciplina de arquitetura de software surgiu com o objetivo de apoiar a construção de software de forma que o mesmo contenha as capacidades requeridas com

qualidade suficiente no tempo e preços combinados. Do ponto de vista de processo, a elaboração da arquitetura tem o papel de operacionalizar requisitos não funcionais e disponibilizar infraestrutura para que os requisitos funcionais possam ser implementados com a qualidade desejada.

$$arquitetura\ de\ software = \{elementos, organização, raciocínio\}$$

Uma das primeiras definições de arquitetura de software foi esboçada em Foundations for the Study of Software Architecture (Wolf, Laboratories, Hill, 1992), onde arquitetura de software é descrita em termos de elementos, organização e raciocínio, sendo que:

> Elementos referem-se a elementos que guardam, usam ou transformam informação ou ainda elementos que conectam elementos de qualquer tipo entre si;
>
> Organização refere-se à forma como estes elementos estão estruturados, seus relacionamentos e restrições;
>
> Raciocínio trata-se da motivação por trás das decisões arquiteturais, isto é, a motivação por trás dos elementos e formas definidos.

Bass, Clements e Kazman (2003) definem arquitetura de software como a estrutura (ou estruturas) do sistema, os elementos contidos nestas estruturas, as propriedades externas visíveis destes elementos e os relacionamentos entre estes elementos.

A definição apresentada no documento IEEE1471 (IEEE, 2000) segue a mesma linha, definindo arquitetura de software como a organização fundamental de um sistema, englobando seus componentes, os relacionamentos entre estes componentes, o ambiente em que estão e os princípios que guiam o seu desenvolvimento e evolução.

As três definições apresentadas enfatizam o aspecto estrutural da arquitetura de software, sendo que Bass, Clements e Kazman (2003) mencionam o fato de que uma arquitetura de software é caracterizada pelas suas propriedades externas, ou seja, pelos seus atributos de qualidade e que Wolf, Laboratories e Hill (1992) e IEEE (2000) complementam Bass, Clements e Kazman (2003) reforçando o aspecto que molda estas propriedades: as decisões arquiteturais.

Bengtsson et al. (2004) alegam que a arquitetura de um software é resultado de decisões tomadas no estágio inicial do ciclo de vida do software. Vale também mencionar a definição dada por Solms e Gruner (2012) que associa o conceito de arquitetura de software a níveis de abstração. Segundo este trabalho, arquitetura de software é a infraestrutura de software sob a qual uma lógica de aplicação contendo funcionalidades dirigidas ao usuário pode ser construída e executada. Neste viés, arquitetura é uma questão de perspectiva, um desenvolvedor de um sistema de vendas na internet pode escolher uma arquitetura baseada em servidor de aplicação como solução arquitetural. Por outro lado, para um desenvolvedor que está construindo o servidor de aplicação, o servidor é a própria aplicação e a arquitetura sob a qual o servidor é construído pode ser o próprio sistema operacional. Nesta linha de pensamento, uma arquitetura de software pode ser descrita pela especificação através de níveis de granularidade dos seguintes aspectos:

conceitos básicos e restrições nas quais a lógica de uma aplicação será especificada;

componentes arquiteturais endereçando preocupações técnicas;

infraestrutura de integração entre os componentes e o ambiente;

táticas arquiteturais utilizadas para atender a requisitos de qualidade;

Ainda que não tenha sido realizada de forma consciente, todo software possui uma arquitetura implícita (IEEE, 2000). Contudo, um projeto bem elaborado de arquitetura de software contribui diretamente para o atendimento a requisitos não funcionais. Serve ainda como ponte entre stakeholders interessados nos detalhes de negócio e aqueles interessados nos detalhes técnicos, já que se trata de um artefato que apresenta alto nível de abstração. Permite também avaliar a qualidade do software nas fases iniciais, antes de implementar a arquitetura. Por fim, decisões de arquitetura restringem escopo, ou seja, definem as características que o software não irá apresentar. O processo de arquitetura de software ajuda abrandar riscos destas decisões, reforça a capacidade de evolução do software e consequentemente a expectativa de vida do mesmo.

2.2.2 O estado da arte

Durante os últimos vinte e cinco anos a disciplina de arquitetura de software desenvolveu-se em vários segmentos. Estes segmentos foram mapeados pelo grupo International Federation of Information Processing Working Group 2.10. O Quadro 2.1 apresenta uma listagem destes segmentos e um resumo do estado da arte da arquitetura de software e oportunidades de pesquisa em cada um destes. Esta tabela foi criada através da análise de surveys (Shaw 2001), (Shaw e Clements, 2006), (Clements e Shaw, 2009), (Stafford, 2006) e da exploração da literatura corrente em arquitetura de software.

O presente trabalho relaciona-se com as oportunidades listadas no segmento de Análise e Seleção de arquiteturas de software uma vez que utiliza a exploração dos relacionamentos entre atributos de qualidade e decisões arquiteturais como ferramenta para a seleção de padrões de software.

Quadro 2.1. Estado da arte nos vários segmentos da arquitetura de software.

Área	Descrição	Estado da arte	Oportunidades
Projeto de Arquitetura	Como criar ou selecionar uma arquitetura de software baseando-se em um conjunto de requisitos funcionais e de qualidade?	Arquiteturas consolidadas para sistemas recorrentes (bancos, aviação, comércio). Estilos arquiteturais. Padrões para arquitetura (arquiteturais e de projeto)	Organização de conhecimento arquitetural além de catalogação de padrões.
Análise e seleção	Como medir a qualidade de um software baseando-se na arquitetura do mesmo? Como comparar duas arquitetura e escolher a mais adequada?	Checklists de requisitos de qualidade, métodos de análise e avaliação de arquiteturas de software (ATAM, SAEM, ALMA). Emergência de processos de tomada de decisão apoiados em atributos de qualidade, táticas arquiteturais e padrões.	Exploração dos relacionamentos formais entre decisões arquiteturais e atributos de qualidade. Sistemas de recomendação que sugiram elementos arquiteturais (padrões, táticas, middleware).
Implementação:	Como implementar um sistema tomando por base a descrição de sua arquitetura?	Engenharia de software baseada em componentes. Automatização por meio de engenharia de software baseada em modelos. MDA.	Conformidade entre arquitetura e código. Derivar arquitetura do código. Testes de software sob perspectiva de arquitetura.
Representação:	Como construir artefatos que possibilitem comunicar a arquitetura de um software seja para humanos ou máquinas? Quais tipos de informação devem ser apresentados em um documento de arquitetura?	Frameworks de representação de arquitetura (The 4+1 View Model of Software Architecture, Zachman Framework). ADLs. UML.	Linguagens que capturem conceitos arquiteturais com mais eficiência. Linguagens de descrição de arquitetura que viabilizem engenharia de software baseada em modelos.
Evolução:	Como a arquitetura de um sistema impacta decisões de negócio? A arquitetura proposta está preparada para receber incrementos decorrentes de modificações no modelo do negócio no qual o software está envolvido?	Abordagens de gerência de arquiteturas. Rastreabilidade arquitetural. Abordagens de análise de impacto de mudanças na arquitetura de software.	Arquiteturas adaptativas.

2.3 Padrões para arquitetura de software

O trabalho de Alexander et al (1977) sobre padrões de projeto para construções civis é a origem do conceito de padrões em engenharia de software. Segundo o referido

trabalho, padrões descrevem um problema recorrente que ocorre em um contexto específico e sua solução.

Os padrões de software começaram a ser mapeados na década de 90 e consolidaram-se como uma maneira popular e complementar de se descrever e expandir projetos de software, capturando e nomeando técnicas que provaram funcionar. Geralmente, padrões de software são descritos textualmente, descrições estas com informação que englobam desde contexto de aplicação até estrutura, consequências positivas e negativas advindas da aplicação do mesmo, relações com outros padrões, variantes entre outras.

Existem vários tipos de padrões de software como, por exemplo, padrões de projeto, padrões de arquitetura, padrões de análise, entre outros. O presente trabalho está interessado em padrões que possam servir como blocos de construção durante a elaboração de arquiteturas de software, logo os padrões arquiteturais são de interesse deste trabalho. Padrões arquiteturais especificam componentes, responsabilidades e relacionamentos entre estes componentes que possam servir como solução para um problema recorrente em contextos específicos de software.

Vale observar como o conceito de padrões arquiteturais alinha-se com a definição de arquitetura de software dada por Bass, Clementz e Kazman (2003). De fato, padrões arquiteturais são templates, ou seja, representações abstratas para arquiteturas de software concretas. Padrões arquiteturais são opções arquiteturais e, portanto a escolha do padrão mais adequado para um dado problema carrega toda a carga de complexidade e risco contida em uma decisão arquitetural.

Assim como qualquer decisão arquitetural, a seleção e aplicação de padrões arquiteturais fundamentam-se nos requisitos do sistema, particularmente, nos requisitos não funcionais. Demanda resolução dos tradeoffs entre requisitos conflitantes e atividades de avaliação de risco que permitam acessar o valor agregado da decisão às metas de negócio prioritárias da aplicação.

Ainda que padrões de projeto como os descritos em Gamma et al. (1995) não possuam enfoque em preocupações arquiteturais, muitos deles podem ser interpretados e aplicados de um ponto de vista arquitetural. Alguns autores como Buschmann, Henney e Schmidt (2007) e Avgeriou e Zdun (2005) reconhecem inclusive que é difícil estabelecer os limites entre padrões arquiteturais e padrões de projeto. A título de exemplo, padrões de projeto como Composite Pattern, Mediator ou Observer podem

ser aplicados sob uma perspectiva global para modificar o comportamento de toda uma arquitetura de software (Giesecke, Hasselbring, Riebisch, 2007). Na verdade, a categorização de um padrão depende fortemente do ponto de vista sob o qual ele é analisado: ponto de vista de designer ou ponto de vista de arquiteto. Assim, padrões como os descritos em Gamma et al.(1995), por conterem em suas descrições detalhes de design como é próprio das descrições contidas nesta fonte, são comumente apontados como padrões de projeto. Porém, instâncias destes padrões podem eventualmente se apresentar como componentes de software externamente visíveis, como acontece com o padrão Interpreter, e podem portanto serem tratados como padrões arquiteturais nestes contextos.

Tomando por base esta ótica, padrões de projeto são do interesse do presente trabalho. A fim de evitar problemas terminológicos, padrões de projeto e padrões de arquitetura são referidos neste trabalho como **padrões para arquitetura**, ou seja, padrões que podem ser aplicados sob uma perspectiva arquitetural.

As vantagens de utilizar padrões durante a elaboração de um projeto de arquitetura de software extrapolam a possibilidade de reusar soluções já testadas. Padrões facilitam o entendimento da arquitetura pelos desenvolvedores, promovem conformidade entre arquitetura proposta e arquitetura implementada, constituem uma maneira eficiente de transferir experiência, reduzem o tempo gasto com documentação de soluções. Além disso, contribuem diretamente para a operacionalização de atributos de qualidade (Klein, Kazman, 2007) (Buschmann et al., 1996), já que as características de qualidade esperadas para uma arquitetura que aplica um padrão estão embutidas na descrição do próprio padrão (Bass, Clements e Kazman, 2003).

O processo de seleção de padrões apresentado na Figura 2.1 segue o fluxo habitual de qualquer técnica de tomada de decisão: identificar um problema, investigar opções, escolher uma opção e aplicá-la.

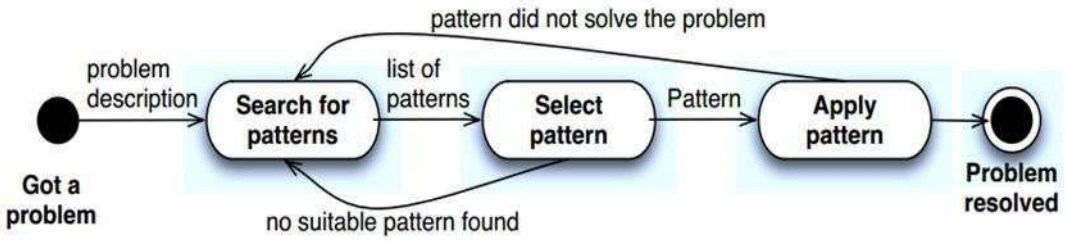

Figura 2.1. Processo de seleção de padrões. Fonte: (Birukou, 2010)

A eficiência deste processo está diretamente relacionada à quantidade de opções disponíveis ao tomador de decisão. Hennigger e Correa (2007) listaram mais de 2000 padrões para arquitetura catalogados em artigos, livros e repositórios. Considerando a quantidade de informação contida na descrição de um único padrão, os relacionamentos colaterais entre padrões e ainda a experiência do desenvolvedor, a tarefa de selecionar padrões torna-se bastante complicada. De acordo com Sommerville (2010), apenas engenheiros de software experientes que possuem conhecimento aprofundado sobre padrões são capazes de utilizá-los de forma eficaz. Estes profissionais são capazes de reconhecer situações genéricas onde um padrão pode ser aplicado. Desenvolvedores menos experientes, mesmo tendo acesso a livros e repositórios sobre padrões, sempre esbarrarão na dificuldade de decidir entre reuso ou desenvolvimento de uma solução de propósito específico.

Mesmo entre desenvolvedores experientes, percebe-se o uso deficiente da informação arquitetural embutida em padrões (Babar, 2004). Os templates de documentação consolidados e comumente utilizados na literatura facilitam o processo de armazenar a informação porém demandam esforço quando o objetivo é extrair esta informação. Alguns autores como Babar (2004) e (Hasheminejad e Jalili, 2012) argumentam que estes templates deveriam englobar outros tipos de informação como relacionamentos do padrão proposto com cenários e atributos de qualidade. Há ainda aqueles que consideram a adoção de formalismos essencial para facilitar a extração de informação das descrições dos padrões.

Babar (2004) propõe inclusive um template para documentação de padrões (Figura 2.2) com seções específicas para registrar informações significantes do ponto de vista arquitetural, estando entre estas informações: as forças que dirigem a aplicação do

padrão (justificativas), táticas de arquitetura utilizadas pelo padrão, atributos de qualidade afetados positivamente e negativamente entre outras informações.

Deve-se reconhecer porém que os templates de documentação mais utilizados na literatura estão consolidados e dificilmente haverá um consenso entre os criadores de padrões para que novos templates sejam adotados. Além disso, reorganizar a infinidade de padrões já existentes de acordo com novos formalismos seria uma tarefa árdua. A opção encontrada para contornar este problema no presente trabalho foi automatizar a extração destas informações importantes que estão implícitas nas descrições de padrões através da procura de termos chave e sinônimos.

Pattern Name: *Name of the software pattern*		**Pattern Type:** *Architecture, design, or style*	
Brief description	*A brief description of the pattern.*		
Context	*The situation for which the pattern is recommended.*		
Problem description	*What types of problem the pattern is supposed to address?*		
Suggested solution	*What is the solution suggested by the pattern to address the problem?*		
Forces	*Factors affecting the problem & solution. Justification for using pattern.*		
Available tactics	*What tactics are used by the pattern to implement the solution?*		
Affected Attributes	**Positively**	**Negatively**	
	Attributed supported	*Attributed hindered*	
Supported general scenarios	S1		
	S2		
	S..n		
Usage examples	*Some known examples of the usage of the pattern to solve the problems.*		

Figura 2.2. Template de descrição de padrões proposto por (Babar, 2004)

2.4 Requisitos não funcionais

Requisitos não funcionais descrevem como uma aplicação deve disponibilizar uma determinada funcionalidade (Gorton, 2011). Podem ser de três tipos: restrições de negócio, restrições técnicas e atributos de qualidade.

As restrições técnicas são requisitos não funcionais acoplados a tecnologia, tais como o fato de uma aplicação ter que executar em determinado sistema operacional, ou ser portável para uma plataforma específica. As restrições de negócio possuem teor semelhante, porém são motivadas por questões de negócio tais como time-to-market, custo, etc. Os atributos de qualidade referem-se às características de qualidade desejadas pelos usuários da aplicação e outros stakeholders envolvidos na construção do software. Constituem exemplos de características de qualidade desejáveis em software desempenho, manutenibilidade, disponibilidade, segurança, etc.

Atributos de qualidade interagem entre si de maneira que apresentar uma determinada característica de qualidade pode reforçar outras características ou implicar em abrir mão de outras. Requisitar alta performance de um sistema por exemplo pode acarretar na necessidade de um alto grau de acoplamento entre plataforma e sistema e afetar negativamente a portabilidade do sistema em questão. Implementar táticas que aumentem a tolerância a falhas como replicação de recursos e dados consome recursos computacionais que impactam negativamente na performance do sistema. Os dois exemplos tratam de tradeoffs clássicos entre performance x portabilidade e confiança (reliability) x performance e demonstram o quanto os atributos de qualidade estão relacionados entre si (Gorton, 2011).

Outra característica marcante dos atributos de qualidade é a subjetividade. Dizer que um sistema deve ser escalável é bastante impreciso, uma vez que um sistema pode escalar em termos de volume de dados, quantidade de conexões, processamento, etc. (Soares et al., 2012). O mesmo problema é perceptível em atributos de qualidade como performance, disponibilidade, etc. Tal subjetividade reflete diretamente na dificuldade de elicitar estes requisitos. A capacidade de tomar decisões arquiteturais compatíveis com o projeto de software em questão está diretamente relacionada com a capacidade de preservar-se desta subjetividade. As abordagens utilizadas para driblar este problema envolvem desde o uso árvores de utilidade (Wojcik et al., 2006) até especificação de cenários de atributo de qualidade (Bass, Clements, Kazman, 2003), (Kazman et al., 1996) ou adoção de frameworks baseados em engenharia de requisitos orientada a metas como i star (Yu, 1997), NFR (Mylopoulos, Chung e Yu, 1999).

A abordagem proposta neste trabalho utiliza uma hierarquia de termos para caracterizar atributos de qualidade e guiar o levantamento destes requisitos. A hierarquia é composta primariamente de termos que denotam atributos de qualidade e termos que parametrizam estes atributos chamados parâmetros de atributo de qualidade. A hierarquia de termos é por si só uma das principais contribuições deste trabalho devido a sua dupla utilidade: constitui uma maneira relativamente simples de caracterizar atributos de qualidade e ainda pode ser utilizada para categorizar e selecionar opções arquiteturais durante o processo de elaboração de arquitetura.

2.5 O processo de tomada de decisão em arquitetura de software

A tarefa de um arquiteto de software pode ser resumida na atividade de transformar requisitos em um projeto tomando decisões que em conjunto dão forma à arquitetura do sistema. Uma vez que o processo de arquitetura de software é essencialmente um processo decisório, é importante caracterizar aspectos chaves tais como os parâmetros usados como entrada e a natureza das decisões tomadas neste processo.

Decisões arquiteturais compreendem decisões estratégicas feitas durante o projeto de arquitetura. Possuem alto impacto sobre o alcance de metas do negócio que o software apoia. Além disso, possuem impacto sistêmico, ou seja, implicam em mudanças na estrutura e no comportamento de todo o sistema. Uma decisão tomada dentro de um escopo mais fechado, sob uma ótica local, não é uma decisão arquitetural.

Envolvem criação de regras de projeto e de restrições aos componentes, definindo o que o sistema ou partes dele não poderá realizar e inclusive como não poderá ser utilizado. Uma decisão pode também acrescentar novos requisitos aos componentes do sistema.

Decisões de arquitetura fundamentam-se em algum requisito funcional ou atributo de qualidade desejado para a arquitetura em questão. Possuem natureza dúbia: uma mesma decisão pode favorecer o atendimento a um determinado requisito e atrapalhar o atendimento a outro requisito. Decisões arquiteturais demandam portanto a resolução de tradeoffs para que requisitos conflitantes sejam resolvidos de forma ótima, ou seja, da forma mais conveniente para os stakeholders do sistema em questão. São decisões tomadas para satisfazer requisitos de alta prioridade para o negócio sob o qual o sistema se apoia.

Uma decisão arquitetural é em parte solução em parte requisito. O entendimento sobre como estas partes se relacionam é essencial para que decisões arquiteturais corretas sejam tomadas.

Conforme levantado por Thiel (2005), arquiteturas de software são primariamente criadas ou evoluídas tomando por base requisitos errados. Não muito raramente, são dirigidas por funcionalidade, sendo aspectos como segurança, desempenho, ou modificabilidade relegados a segundo plano e tratados de maneira ad-hoc. Além disso, não existe um processo adequado para priorizar requisitos, o que dificulta lidar com requisitos conflitantes. As decisões são tomadas por instinto, sendo o raciocínio por trás das mesmas desprezado, não documentado e perdido conforme o sistema envelhece.

Entre os elementos que devem ser considerados durante a seleção de uma opção arquitetural inclui-se:

Restrições de negócio: a quantidade de recursos disponíveis, as características da indústria, o tempo disponível para finalização do projeto entre outros fatores relativos ao negócio que o software apoia devem ser considerados durante a escolha de uma opção arquitetural. Frequentemente, a opção arquitetural ideal para um problema de software não está de acordo com as restrições de negócio impostas, uma vez que podem ter custo financeiro alto ou não encaixar no cronograma previsto;

Restrições técnicas: uma arquitetura deve estar em conformidade com restrições impostas por opções tecnológicas resolvidas de antemão;

Atributos de qualidade: os atributos de qualidade também influenciam as decisões arquiteturais uma vez que estes são operacionalizados em boa parte pelo projeto de arquitetura;

Domínio da aplicação: o fato de o software ser um sistema para internet, ou um sistema de tempo real, ou um sistema stand-alone limita as opções arquiteturais disponíveis para escolha. Porém, de acordo com (Thiel, 2005), um arquiteto seleciona opções baseando-se mais em seus efeitos nos atributos de qualidade que no domínio específico da aplicação. Entretanto, o conhecimento do domínio ajuda o arquiteto a pensar nos problemas típicos que o referido domínio inclui a respeito de qualidades particulares;

Requisitos funcionais: uma vez que entre os objetivos de se arquitetar figura a necessidade de criar arquiteturas que possibilitem dividir o trabalho de implementação do software de maneira razoável, os requisitos funcionais também influenciam na estruturação da arquitetura de software.

Dentre os elementos levantados, julgamos que as restrições de negócio e os atributos de qualidade sejam os mais críticos. O primeiro devido à dificuldade inerente a tarefa de determinar o valor de uma arquitetura ou o esforço necessário para implementar e manter uma, o que necessariamente implica em avaliação econômica de COTS, padrões de software e infraestrutura sob a qual a arquitetura será executada. O segundo, devido à subjetividade dos atributos de qualidade, os quais apesar da

importância, são difíceis de especificar, mensurar e acessar, principalmente durante a elaboração da arquitetura.

Este trabalho promove uma abordagem de seleção de opções arquiteturais com enfoque em atributos de qualidade. Ainda que se compreenda a relevância de outros fatores na tomada de decisão em arquitetura de software, devido à necessidade de estreitar o escopo, este trabalho restringe-se a considerar requisitos de qualidade como parâmetros de tomada de decisão.

2.6 Trabalhos relacionados

O ponto de partida da investigação dos trabalhos relacionados a seleção de padrões foi a inspeção das bases de dados acadêmicas Scopus[1] e Google Scholar[2]. Inicialmente, as seguintes combinações de palavras chave foram utilizadas nas consultas: *"decision making" + "software architecture" + "survey"* e *"pattern selection" + "survey"*. O objetivo foi descobrir se existia estudos relevantes e recentes sobre o estado da arte em tomada de decisão em arquitetura de software. Os trabalhos de Falessi et al. (2011) e Birukou (2010) destacaram-se entre os resultados da consulta por serem trabalhos recentes e bem estruturados. O trabalho de Falessi et al. (2011), relacionado a tomada de decisão em arquitetura de software, além de levantar os tipos de entradas comumente utilizados por abordagens de tomada de decisão arquiteturais, lista alguns trabalhos relacionados a seleção de padrões. O trabalho de Birukou (2010) é focado em revisar abordagens de seleção de padrões e foi portanto a principal fonte de trabalhos relacionados citados nesta seção. Objetivando não limitar-se aos trabalhos listados nos surveys citados, as seguintes consultas foram submetidas às referidas bases de dados acadêmicas: *"pattern selection", "recommender systems" + "architectural patterns"* entre outras consultas similares. Por fim, realizou-se um rastreio de citações aos repositórios de padrões mais conhecidos – Gamma et al. (1995) e Buschmann et al. (1996) - nestas mesmas bases de dados a fim de encontrar algum outro trabalho relevante não encontrado pelas consultas anteriores.

O critério de inclusão dos trabalhos compôs-se das seguintes regras:

[1] http://www.scopus.com/

[2] http://scholar.google.com.br/

i) o título deixa claro que trata-se de um trabalho que apresenta técnicas de seleção, aplicação ou classificação de padrões;
ii) o trabalho apresenta resultados aceitos pela comunidade, aceitação verificada através da inspeção da quantidade de citações relativa a data de publicação.

Uma vez que consideramos que tanto padrões arquiteturais quanto padrões de projeto podem contribuir para a arquitetura, a investigação considera trabalhos que apresentam abordagens de seleção para qualquer um dos dois tipos de padrões. Notou-se inclusive que alguns dos trabalhos levantados introduzem padrões de projeto como mecanismos de refinamento de arquiteturas de software, o que reforça mais ainda a necessidade de listá-los entre os trabalhos relacionados. O Quadro 2.2 apresenta uma listagem com os trabalhos identificados durante a revisão de literatura, destacando o escopo dos padrões tratados nestes trabalhos e as características gerais dos mesmos.

Quadro 2.2. Trabalhos levantados que introduzem os padrões e as características dos projetos

#	Título do Trabalho	Escopo	Características do trabalho
1	Design pattern selection: a solution strategy method (Sahly e Sallabi, 2012)	Projeto	Seleção de padrões baseada em consultas.
2	Design patterns selection: An automatic two-phase method. (Hasheminejad e Jalili, 2012)	Projeto	Utilizam classificação de texto supervisionada e algoritmos de clusterização para agrupar padrões similares. Não demanda especificação formal dos padrões. Padrões sugeridos de acordo com a similaridade com o problema. Enxergam a aplicação de padrões de projeto como refinamento da arquitetura. Uma vez que foca em problemas de design, não aborda a questão dos tradeoffs.
3	Selecting architectural patterns through a knowledge-based approach (Soares et al., 2011)	Arquitetura	Abordagem baseada em conhecimento. Utiliza regras para armazenar e inferir conhecimento de padrões.
4	Design Pattern Recommendation System (Methodology, Data Model and Algorithms) (Suresh, Naidu e Kiran, 2011)	Projeto	Seleção de padrões baseada em consultas. Utiliza rótulos para indicar grau em que padrão é recomendado para determinado problema: *"strongly recommended", "satisfactorily recommended", "not recommended*".
5	Context-aware privacy design pattern selection (Pearson e Shen, 2010)	Projeto	Utiliza regras para armazenar e inferir conhecimento de padrões. Calcula grau de confiança da recomendação.
6	QoS Architectural Patterns for Self-Architecting Software Systems Categories and Subject Descriptors (Menascé et al., 2010)	Arquitetura	Define o coeficiente QoS - quality of service – de um padrão em termos de estrutura, comportamento e métricas de qualidade. Utiliza deste coeficiente para recomendar padrões.
7	Service for selecting patterns (Birukou e Weiss, 2009)	Arquitetura e Projeto	Apresenta um serviço de apoio ao uso de padrões que combina funcionalidades como busca, indexação e etiquetação de padrões, recomendação de padrões via histórico de uso dos mesmos, geração de código a partir de definição de padrão, serviço de detecção de padrões, etc.
8	A Simple Recommender System for Design Patterns (Guéhéneuc e Mustapha, 2007)	Projeto	Utiliza palavras chave pré-definidas como entrada para classificação e seleção de padrões.

#	Título do Trabalho	Escopo	Características do trabalho
9	Object-oriented design : A goal-driven and pattern-based approach (Lin, 2009)	Projeto	Especifica a atividade de seleção de padrões no processo de desenvolvimento de software orientado a metas. Não apresenta um método para selecionar padrões. Apresenta uma forma de classificar padrões baseando-se no papel que um padrão pode exercer: facilitador, melhorador de qualidade, funcional. Afirma que um padrão pode exercer vários papéis, e podem ser aplicados em diferentes fases do ciclo de vida do software para resolver diferentes tipos de problemas. Mostra como alguns padrões são apropriados para resolver conflitos entre requisitos de qualidade.
10	An Ontological Interface for Software Developers to Select Security Patterns (Khoury et al., 2008)	Projeto	Seleção de padrões de projeto para segurança através da definição de ontologias. Propriedades de segurança devem ser descritas formalmente. Considera perfis de usuário: desenvolvedor, arquiteto, etc.
11	Formalizing Architectural Patterns with the Goal-Oriented Requirement Language. (Mussbacher, Weiss e Amyot, 2006)	Arquitetura	Utiliza GRP (Goal Oriented Requirement Language) para formalizar padrões para arquitetura de modo que uma análise rigorosa de tradeoffs seja possível. As forças de um padrão são representadas como metas flexíveis (soft goal). As metas flexíveis são um misto de características da arquitetura e táticas arquiteturais. Padrões são modelados como tarefas, representando maneiras de atender a uma meta flexível. Sugere ainda um processo para comparar padrões. Promove o entendimento do impacto da combinação de padrões em requisitos não funcionais.
12	From software architecture to design patterns: a case study of an NFR approach (Wang, Song e Chung, 2005)	Projeto	Propõe o refinamento da arquitetura de software através da aplicação de padrões de projeto. Modelos NFR contendo metas não funcionais e metas arquiteturais obtidas do refinamento das metas não funcionais são refinadas por meio da aplicação de padrões. Retorna um ranking de padrões mais adequados para o refinamento da arquitetura.
13	Developing adaptable software architectures using design patterns: an NFR approach (Chung, Cooper e Yi, 2003)	Arquitetura	Apresenta Proteus, um framework voltado para o desenvolvimento de arquiteturas de software adaptáveis utilizando padrões de projeto. Utiliza NFR.
14	Systematic pattern selection using pattern language grammars and design space analysis (Zdun, 2007)	Arquitetura e Projeto	Propõe uma abordagem de seleção de padrões fundada em atributos de qualidade e seus relacionamentos com padrões de software. Demanda formalização dos relacionamentos entre padrões em uma linguagem específica e anotação de padrões com seus efeitos sobre atributos de qualidade.

#	Título do Trabalho	Escopo	Características do trabalho
15	Using CBR for Automation of Software Design Patterns (Gomes et al., 2002)	Projeto	Organiza o conhecimento sobre padrões em torno da documentação de experiências passadas de uso dos padrões.
16	Recommendation System for Design Patterns in Software Development : An DPR Overview (Palma e Farzin, 2012)	Projeto	A descrição dos padrões é formalizada de maneira que facilite o processo de seleção. Utiliza Goal Question Metrics para inferir conhecimento, e consequentemente questionários como interface para o arquiteto ou desenvolvedor de software.
17	From Non-Functional Requirements to Design through Patterns (Gross e Yu, 2000)	Projeto	Mapeia relacionamento entre padrões de projeto e requisitos não funcionais.
18	Uma Abordagem para a Seleção de Padrões Arquiteturais Baseada em Características de Qualidade (Xavier, Werner e Travassos, 2002)	Arquitetura	Utiliza atributos de qualidade como entrada para a seleção. Modela características como dimensões de um espaço vetorial e padrões como pontos deste espaço. O problema é convertido para um ponto deste espaço de acordo com a importância de cada atributo de qualidade. O padrão com menor distância vetorial para o problema é o mais indicado.

Os trabalhos levantados preconizam etapas de construção da base de conhecimento, extração de conhecimento e interação com usuário em suas abordagens. A Figura 2.3 ilustra estes passos e organiza os trabalhos relacionados sob esta perspectiva.

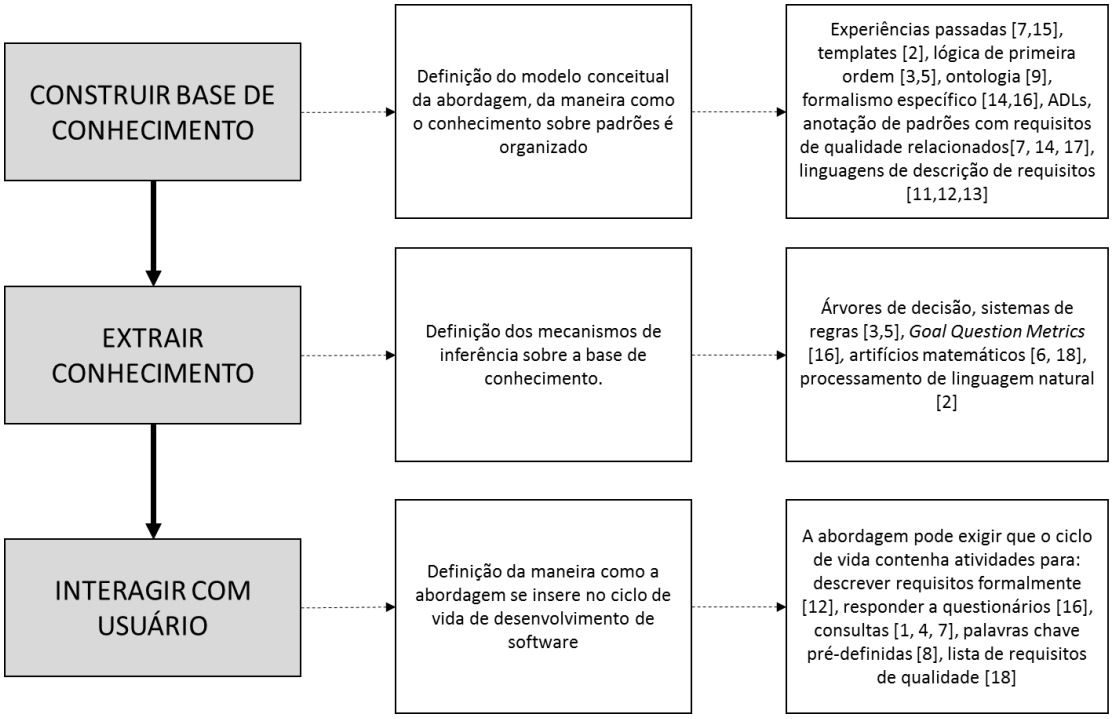

Figura 2.3. Classificação das abordagens quanto às técnicas que aplicam

Entre as características comuns às abordagens levantadas percebe-se que a maioria demanda esforço adicional. Os autores precisam especificar padrões em algum formalismo qualquer, enriquecer suas descrições, coletar experiências de uso. Uma vez descritos segundos estes formalismos, estas informações precisam ser mantidas e atualizadas conforme a base de padrões cresce ou novos problemas são resolvidos. Existe um problema persistente relacionado a escalabilidade das abordagens, o que fica evidente se analisados os mecanismos de extração de conhecimento comumente utilizados. Aumentar a quantidade de padrões suportados pela abordagem implica aumentar a quantidade de questões de um questionário ou o número de regras de inferência contidas na base de dados.

A adequação ao ciclo de vida de desenvolvimento de software é outro desafio inerente à tarefa de selecionar padrões. Definir requisitos de software em notações específicas ou responder questionários podem tornar o ciclo de vida do projeto moroso e onerar o desenvolvimento.

3 CONSTRUÇÃO DE BASE DE CONHECIMENTO DE APOIO A SELEÇÃO DE PADRÕES

3.1 Modelo conceitual

Técnicas de tomada de decisão são estruturadas de acordo com modelos conceituais os quais apresentam os conceitos pertinentes ao domínio do problema, suas associações e atributos. O problema com o qual lidamos neste trabalho refere-se à seleção de padrões para arquitetura de software utilizando procura de termos chave e sinônimos. Assim, **padrão** e **termo** são conceitos fundamentais para este trabalho. Padrões são documentados utilizando-se templates com várias **seções** e podem ser vistos como conjuntos de **táticas** de arquitetura. Por fim, os padrões, especialmente aqueles voltados para a resolução de problemas arquiteturais objetivam estruturar o software de modo que a transparecer características de qualidade particulares, que podem ser descritas em termos de **atributos de qualidade** e **parâmetros de atributo de qualidade**. A Figura 3.1 mostra como estes conceitos estão associados e as seções subsequentes explicam em detalhes cada um deles.

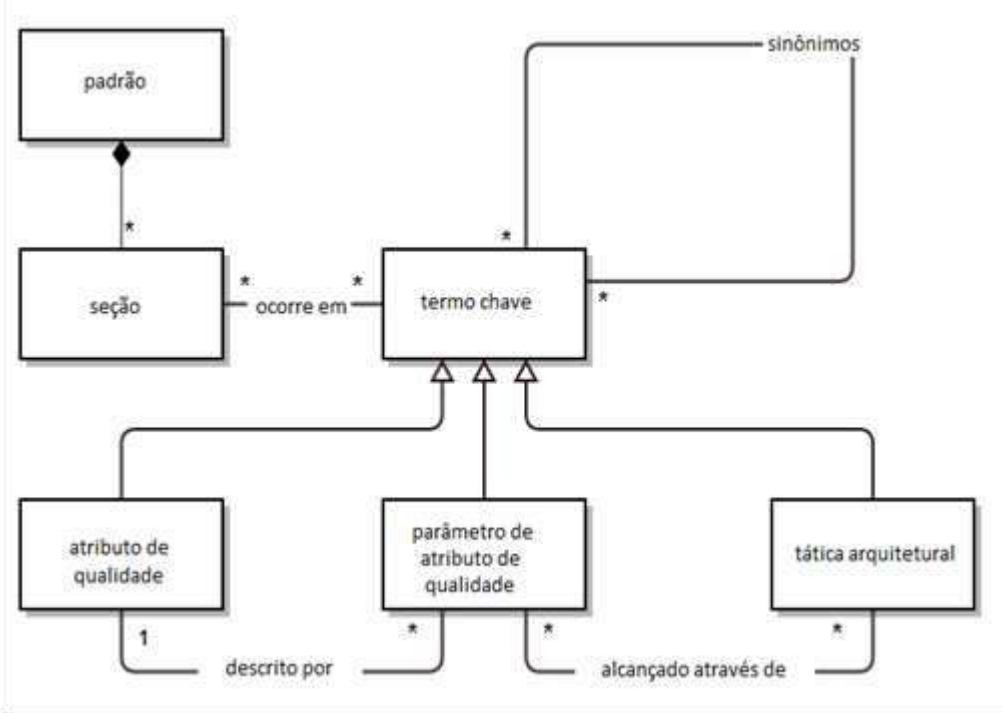

Figura 3.1. Modelo conceitual utilizado na abordagem.

3.1.1 Padrões e seções

Assim como mostrado no Capítulo 2, padrões de software costumam ser especificados através da descrição textual de seus componentes, relacionamentos e maneiras pelas quais eles relacionam entre si. Esta descrição é estruturada de acordo com algum template que indica as seções que a descrição do padrão deve ter. Figuram entre os templates mais utilizados os da série Pattern Oriented Software Architecture (POSA) (Buschmann et al., 1996) e do livro Design patterns: elements of reusable object-oriented software (Gamma et al., 1995). Apesar das diferenças entre os templates é possível inferir relações de correspondência entre seções dos diferentes templates, assim como demonstrado no Quadro 3.1.

Uma das premissas desta abordagem é que algumas seções carregam carga contextual suficiente de modo que é possível inferir o teor semântico dos termos que nelas ocorrem. Em outras palavras, algumas seções se preocupam majoritariamente em apresentar as características positivas do padrão de modo a justificar o uso do mesmo, enquanto outras se preocupam em apresentar cenários desfavorecidos pela aplicação do padrão. Outras seções não apresentam nenhuma destas características de forma preponderante e são consideradas neutras. Assim, termos que ocorrem em seções que

apresentam conotação positiva indicam relacionamento positivo entre estes termos e o padrão enquanto termos que ocorrem em seções com conotação negativa indicam relacionamento negativo.

Quadro 3.1. Correspondência entre seções dos templates do POSA e do GAMMA

Buschman	GOF	Descrição
Name	Name	Um nome conciso que transmita a essência do padrão
Also known as	Also known as	Outros nomes que o padrão tenha, caso os tenha.
Problem	Intent	Seção responsável por apresentar o problema que o padrão resolve, incluindo uma discussão das forças associadas ao padrão. Em outras palavras, descreve o que o padrão de faz, o raciocínio embutido.
Example	Motivation	Um cenário que ilustra o problema de projeto e como a estrutura proposta pelo padrão resolve o problema.
Context	Applicability	Descreve as situações onde o padrão pode ser aplicado.
Structure	Structure	O princípio fundamental da solução contida no padrão, comumente composto por representações gráficas que ilustrem os elementos contidos na solução e como eles relacionam, interagem e colaboram entre si.
Dynamics	Collaborations	Cenários típicos descrevendo o comportamento em tempo de execução do padrão.
Implementation	Implementation	Diretrizes para implementação do padrão, incluindo possíveis armadilhas, sugestões ou técnicas deve-se estar consciente ao implementar o padrão.
Consequences	Consequences	Os benefícios provenientes da aplicação do padrão e suas fraquezas. Tradeoffs que devem ser considerados e os resultados de utiliza-se este padrão.
Known Uses	Known Uses	Exemplos de uso do padrão encontrado em sistemas reais.
See Also	Related Patterns	Referências à padrões que resolvem problemas similares

Dito isto, o questionamento que se segue é identificar quais seções apresentam conotação positiva, negativa ou neutra. O Quadro 3.2 classifica as seções comumente utilizadas na descrição de padrões de acordo com a carga contextual predominante nas mesmas e apresenta as justificativas para as classificações atribuídas.

Quadro 3.2. Conotação predominante nas seções das descrições de padrões

Seção	Conotação	Observações
Name	Positiva	Uma vez que o objetivo desta seção é comunicar em poucas palavras o objetivo do padrão, a ocorrência de termos chave nesta secção possui alta probabilidade de indicar alguma característica beneficiada pela aplicação do padrão.
Problem - Intent	Positiva	Expressar o objetivo de um padrão é basicamente expressar que problema ele resolve. Assim, termos chave que ocorrem nesta secção também possuem probabilidade considerável de indicar alguma característica beneficiada pela aplicação do padrão.
Context - Applicability	Positiva	Apresenta variáveis de decisão chave que devem ser observadas antes de adotar o padrão. Apesar de em alguns casos apresentar também situações em que não se deve aplicar o padrão, boa parte das vezes a conotação dos termos que ocorrem nesta seção é positiva.
Consequences (Benefits and Liabilities)	Positiva ou negativa	Esta seção objetiva enumerar os impactos positivos e negativos advindos da aplicação do padrão. No template do POSA, impactos positivos e negativos não chegam a estar separados em subseções mas são apresentados com grau de separação facilmente identificável. Já no template do GOF o grau de separação é menor sendo mais difícil identificar o que é benefício ou fraqueza. Apesar desta dificuldade de inferir contexto na secção de consequências, esta não poderia ser excluída da abordagem uma vez que trata-se da secção mais propensa a apresentar os termos utilizados como parâmetros de decisão nesta abordagem. Diante disso, fez-se necessário subdividir esta secção em benefícios e fraquezas. Esta separação deverá ser feita manualmente ao se alimentar a base de dados.

3.1.2 Atributos de qualidade

Atributos de qualidade são definidos como propriedades não funcionais desejadas em um sistema (Barbacci et al., 2003). Exemplos de atributos de qualidade incluem desempenho, manutenibilidade, segurança, portabilidade, confiança e etc.

Arquiteturas de software são desenhadas para promover o atendimento a requisitos em especial, requisitos não funcionais. Logo, o entendimento dos efeitos de decisões arquiteturais sobre requisitos não funcionais é fundamental para que decisões corretas sejam tomadas. De fato, o estudo da relação entre atributos de qualidade e

decisões arquiteturais é uma das áreas de pesquisa mais promissoras em arquitetura de software. Alguns trabalhos inclusive convergem para a catalogação dos efeitos da aplicação de táticas arquiteturais ou mesmo de padrões para arquitetura sobre atributos de qualidade.

Lidar com atributos de qualidade de maneira isolada durante um projeto de arquitetura é perigoso, uma vez que costumam impactar entre si. Os atributos confiabilidade (reliability) e desempenho (performance) constituem um bom exemplo de atributos de qualidade conflitantes, uma vez que táticas como replicar comunicação e computação comumente utilizadas para elevar o grau de confiabilidade de um sistema impactam negativamente no desempenho enquanto colocar processos críticos em um mesmo local de modo a diminuir o overhead gerado durante a comunicação entre processos para alcançar desempenho impacta negativamente a confiabilidade do sistema. De fato, como será mostrado durante a discussão dos resultados, o impacto entre diferentes atributos de qualidade é descrito explicitamente nos padrões, sendo que alguns padrões lidam diretamente com requisitos de qualidade conflitantes o que constitui outra vantagem de aplicar padrões durante o projeto de arquitetura.

Outro problema com atributos de qualidade é que os termos que os denotam possuem sentido generalista. O atributo de qualidade performance (desempenho), por exemplo pode ser entendido tanto sob a perspectiva da quantidade de tempo dispensada para realizar uma determinada tarefa (latency), quanto em relação a quantidade de dados que um sistema é capaz de produzir em uma determinada unidade de tempo (throughput), ou sob a perspectiva da gerência de recursos pela aplicação (resource management). Assim, atributos de qualidade não podem ser considerados auto descritivos e necessitam de artifícios que ajudem a contextualizá-los durante a definição de um problema ou mesmo durante a classificação e seleção de um padrão de software.

3.1.3 Parâmetros de atributos de qualidade

Para a especificação e avaliação de uma arquitetura de software em termos da aderência a requisitos de qualidade, tais requisitos precisam ser expressos em termos que sejam mensuráveis ou ao menos observáveis (Klein et al., 1999). A subjetividade

inerente aos termos que denotam atributos de qualidade demanda artifícios que permitam defini-los melhor durante a contextualização de um problema. Barbacci et al. (2003) utilizam cenários para descrever concretamente os atributos de qualidade que são importantes para um determinado sistema e quais as respostas desejadas para cada atributo de qualidade. Os cenários são capazes de expressar instâncias particulares de características de qualidade desejáveis em um sistema (Kazman et al., 1996). A principal vantagem de se utilizar cenários para contextualizar atributos de qualidade é a possibilidade de avaliar de forma concreta se o sistema contém um atributo de qualidade ou não. Porém a criação de cenários é realizada utilizando-se um problema de projeto concreto, o que não serve para uma abordagem que propõe classificar e recomendar soluções descritas em um nível de abstração superior. Precisa-se reduzir o nível de abstração dos atributos de qualidade, utilizando-se, porém um meio-termo que evite as especificidades dos cenários.

Objetivando representar atributos de qualidade com maior nível de detalhes Soares et al. (2012) propôs o conceito de parâmetros de atributo de qualidade que são parâmetros ou subrequisitos pelos quais atributos de qualidade podem ser especificados, julgados e medidos.

O atributo de qualidade segurança, por exemplo, pode ser analisado sob vários pontos de vistas:

nonrepudiation: a propriedade de a ocorrência de uma transação não poder ser negada por nenhuma das partes envolvidas na mesma;

data integrity: a propriedade de garantir que os dados são entregues como combinado pelas partes;

confidentiality: a propriedade dos dados ou serviços serem protegidos de acesso não autorizado;

auditing: a propriedade de armazenar as atividades realizadas no sistema em níveis suficientes para que as mesmas possam ser desfeitas ou refeitas;

assurance: a propriedade de garantir que as partes envolvidas em uma transação são quem elas dizem ser.

A presença de uma subcaracterística destas em um sistema de software indica o atendimento parcial ao referido atributo de qualidade. O refinamento de atributos de qualidade utilizando parâmetros de atributo de qualidade ameniza os problemas com

vocábulos ambíguos e ainda permite classificar e selecionar padrões com maior nível de granularidade.

3.1.4 Táticas arquiteturais

Segundo Bass, Clements e Kazman (2003), padrões para arquitetura utilizam estratégias arquiteturais para conduzir um problema de projeto a uma solução adequada. Estas estratégias são comumente chamadas de táticas arquiteturais. As táticas arquiteturais são subterfúgios comumente empregados na construção de uma arquitetura de software para possibilitar o atendimento a um determinado conjunto de requisitos e estão diretamente relacionadas ao alcance de metas de qualidade desejadas durante o desenvolvimento de um projeto de software. Garland e Anthony (2003) definem táticas como decisões de projeto que ajudam no alcance de um requisito em particular.

Segundo Kim et al. (2009), táticas arquiteturais são "blocos de construção" reusáveis que contêm soluções arquiteturais genéricas para problemas relacionados a atributos de qualidade, sendo que muitas destas táticas estão documentadas na literatura incluindo o relacionamento das mesmas com requisitos de qualidade. Alguns trabalhos inclusive formalizam abordagens de utilização de táticas na construção de arquiteturas de software. Kim et al. (2009) formalizaram táticas de arquitetura utilizando UML e forjaram um mecanismo de composição de táticas para construir arquiteturas que atendam aos requisitos não funcionais especificados. As táticas diferem-se de padrões pelo fato de não capturarem uma implementação em particular da solução de um problema de projeto, mas restringirem-se a disponibilizar um guia sobre como lidar com o problema. Segundo Harrison, Avgeriou e Zdun (2010), a implementação de táticas se assemelha com a implementação de funcionalidades: cada tática possui um desenho, onde geralmente é decomposta em componentes, conectores entre os componentes e um comportamento requerido. Segue então que a estrutura e o comportamento de uma tática afeta diretamente a estrutura e o comportamento do sistema.

O Quadro 3.3 ilustra alguns exemplos de táticas e explicita como elas podem ser categorizadas de acordo com os atributos de qualidade com que se relacionam.

As táticas arquiteturais são utilizadas neste trabalho para reforçar a capacidade de extrair conhecimento da abordagem. Espera-se que caso a descrição de um padrão deixe explícito que aplica conceitos de concorrência para resolver um problema, ou mesmo, resolva um problema decorrente da aplicação de concorrência em um sistema, contribua para uma arquitetura capaz de suportar funcionalidades com baixo tempo de resposta ou mesmo uma melhor utilização dos recursos computacionais disponíveis. Esta premissa é apoiada em trabalhos de autores que descrevem padrões e suas variantes em termos de táticas arquiteturais (Bachmann, Bass, Nord, 2007) e trabalhos que mapeiam o relacionamento entre várias táticas arquiteturais e requisitos de qualidade (Bass, Clements e Kazman, 2003). Bachmann, Bass e Nord (2007) relaciona táticas arquiteturais diretamente com parâmetros de atributo de qualidade. De fato, táticas disponibilizam meios arquiteturais de ajustar estes parâmetros de forma a impactar positivamente ou negativamente em atributos de qualidade.

Quadro 3.3. Exemplos de táticas arquiteturais

Nome da Tática	Atributo de qualidade	Descrição
Exceptions	Availability	Trata-se de uma tática que previne falhas no software através do reconhecimento e tratamento destas falhas. Impacta diretamente no nível de disponibilidade do software uma vez que aumenta a quantidade de tempo de uso do software sem que o mesmo falhe.
Ping/Echo	Availability	Trata-se também de uma tática de disponibilidade utilizada para checar se um componente está disponível através do envio de mensagens de ping para o componente e esperar alguma resposta do mesmo em um determinado intervalo de tempo.
Semantic Coherence	Modifiability	Trata-se de uma tática para melhorar a modificabilidade de um componente arquitetural através de uso de mecanismos como herança e polimorfismo.
ID/Password	Security	Trata-se de uma tática de segurança utilizada para garantir a autenticidade de usuários através da utilização de identificadores de usuários e senhas.
Concurrency	Performance	Concorrência é uma tática comumente utilizada com o objetivo de otimizar a utilização dos recursos disponíveis e com isso diminuir o tempo de resposta do sistema.

3.1.5 Sinônimos

Identificar e agrupar conceitos semelhantes é essencial para que informação de qualidade seja extraída. Um atributo de qualidade, parâmetro de atributo de qualidade ou mesmo tática arquitetural pode ser mencionada de inúmeras maneiras.

O termo performance por exemplo pode ser substituído por efficiency sem maiores prejuízos. Ainda que dois termos não sejam sinônimos, eles podem estar muito próximos quanto à ideia que representam e podem ser considerados equivalentes em determinados contextos. Objetivando atender a esta demanda, a base de conhecimento comporta a representação da relação sinônimo (termochaveA, termochaveB) de forma que seja possível mapear ocorrências de termos diferentes em um mesmo cluster.

3.2 Povoamento da base de conhecimento

Uma vez criado o modelo conceitual utilizado na abordagem, é necessário povoar a base de conhecimento com termos que denotem atributos de qualidade, parâmetros de atributo de qualidade e táticas de arquitetura. O trabalho de busca de termos chave para povoar a base de dados seguiu duas frentes:

i) revisão de literatura em engenharia de requisitos e arquitetura de software;
ii) procura de sinônimos dos termos levantados em bases de dados lexicográficas tais como WordNet, Thesaurus e Merriam Webster;

A revisão de literatura em engenharia de requisitos teve como objetivo encontrar trabalhos que catalogavam requisitos não funcionais ou apresentavam modelos de avaliação de qualidade de software. Por outro lado, a revisão em arquitetura de software concentrou-se em identificar trabalhos que promoviam o entendimento da relação entre requisitos não funcionais e arquitetura, articulavam abordagens para avaliação do grau de satisfação de uma arquitetura quanto aos requisitos de um sistema ou ainda identificavam elementos do domínio da solução que poderiam ser utilizados na extração de conhecimento.

O Quadro 3.4 apresenta os trabalhos encontrados na literatura e que constituíram a principal fonte de termos chave para a base de conhecimento. O processo de mineração de termos é composto pelos seguintes passos:

i. identificação do modelo conceitual utilizado pelo trabalho;
ii. identificação das relações de equivalência entre os conceitos do trabalho com a abordagem proposta nesta dissertação;
iii. identificação de termos chave e inserção na base de conhecimento;

Quadro 3.4. Listagem de trabalhos utilizados como fontes de termos chave e as características destes trabalhos

#	Título do Trabalho	Características do trabalho
1	Architectural Tradeoff Analysis Method (Kazman, Klein e Clements, 2000)	Este trabalho mapeia consequências de decisões arquiteturais sobre requisitos de atributos de qualidade e identifica tradeoffs a serem realizados entre atributos de qualidade. Consideram a utilização de estilos arquiteturais como determinantes para a presença de atributos de qualidade na arquitetura. Elementos do modelo conceitual proposto pertinentes ao presente trabalho: **Metas de qualidade:** equivalente ao conceito de atributo de qualidade como definido neste trabalho
2	Factors in Software Quality (Mccall, Richards e Walters, 1977)	Relatório técnico criado pela força aérea estadunidense e centro de desenvolvimento aéreo de Roma para provisionar padrões e apoio técnico para a aquisição de software. Tanto questões da especificação quanto da avaliação da qualidade de software são tratadas neste relatório. Elementos do modelo conceitual proposto: **Fatores**: condição ou característica que contribui ativamente para a qualidade do software. Equivalentes aos atributos de qualidade. Exemplos: correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, interoperability. **Critérios**: atributos do software ou processo de produção do software pelos quais os fatores podem ser julgados e definidos, ou ainda, características do software que caso presentes contribuem de alguma forma para algum fator de qualidade. Equivale ao conceito de parâmetro de atributo de qualidade. Exemplos: traceability, completeness, consistency, accuracy, error tolerance, simplicity, modularity. **Métricas:** medida quantitativa de um atributo do software relacionado a um ou mais fatores de qualidade. Este trabalho define também alguns tradeoffs comuns entre fatores de qualidade: maintainability x efficiency, integrity x efficiency, interoperability x integrity. Estes tradeoffs foram utilizados para validar a abordagem proposta neste trabalho.
3	Quantitative Evaluation of Software Quality (Boehm, Brown e Lipow, 1976)	Apresenta um framework conceitual de qualidade de software. Elementos do modelo conceitual proposto: **Características de qualidade**: equivale ao conceito de atributo de qualidade. Foram organizadas em uma estrutura hierárquica de forma que características em níveis intermediários formam um conjunto de características necessárias para que um software apresente uma característica do nível superior em sua totalidade. **Métrica**: medida da extensão ou grau no qual um produto possui ou exibe uma certa característica de qualidade.

#	Título do Trabalho	Características do trabalho
4	Attributed-Based Architectural Styles (ABAS) (Klein et al., 1999)	Não se trata de um modelo de qualidade, mas de um framework de especificação de estilos arquiteturais em termos de atributos de qualidade. Elementos pertinentes propostos no modelo conceitual: **Medidas de atributo de qualidade**: definidos como uma especificação do problema, mas em termos pertinentes a aspectos mensuráveis do modelo de atributo de qualidade. Aproxima do conceito de parâmetro de atributo de qualidade como definido neste trabalho. **Parâmetros de atributo de qualidade**: Utiliza também o conceito de "parâmetro de atributo de qualidade" em sua terminologia, porém em uma acepção de decisão arquitetural, propriedade da arquitetura ou mecanismos arquiteturais, diferentemente do presente trabalho que entende parâmetros de atributo de qualidade como parâmetros pelos quais atributos de qualidade podem ser julgados. Latency por exemplo é considerado um parâmetro de atributo de qualidade no presente trabalho e não se trata de uma propriedade da arquitetura e muito menos ainda uma propriedade ajustável da arquitetura. No caso, decisões arquiteturais, sejam elas na forma de táticas ou mesmo padrões são as propriedades ajustáveis da arquitetura.
5	Seleção de padrões para arquitetura de software: uma abordagem baseada em atributos de qualidade (Soares et al., 2012)	Apresenta um modelo conceitual para guiar a seleção de padrões. Elementos do modelo conceitual proposto: **Atributos de qualidade:** equivale ao conceito de atributo de qualidade utilizado neste trabalho; **Parâmetro de atributo de qualidade**: origem do conceito utilizado no neste trabalho. Construiu uma base inicial de atributos de qualidade e parâmetros de atributo de qualidade, os quais foram incluídos na base de dados do presente trabalho.
6	Software architecture in practice: Second Edition (Bass, Clements e Kazman, 2003)	Um dos livros mais conhecidos de arquitetura de software, coloca atributos de qualidade como elemento central do processo de construção de arquiteturas de qualidade. Elementos do modelo conceitual proposto: **Atributos de qualidade** **Preocupações (concerns)**: Equivale ao conceito de parâmetro de atributo de qualidade **Táticas arquiteturais**: Mesma acepção utilizada neste trabalho.
7	Essential Software Architecture (Gorton, 2011)	Livro sobre arquitetura de software que lista os **principais atributos de qualidade.**
8	Attribute Driven Design (Wojcik et al., 2006)	Apresenta um método de desenhar arquiteturas tomando por base atributos de qualidade. Lista **atributos de qualidade** importantes.
9	CISQ (CISQ, 2011)	Consórcio idealizado para melhoria de qualidade de software. Apresenta uma listagem **de atributos de qualidade** desejáveis em um software.

#	Título do Trabalho	Características do trabalho
10	IEEE Standard for a Software Quality Metrics Methodology (IEEE, 1998)	Apresenta uma metodologia para estabelecer requisitos de qualidade, identifica-los, implementá-los, analisá-los e validar a qualidade do processo e do produto de software. Estabelece uma listagem de **atributos de qualidade**.
11	ISO – IEC (ISO/IEC, 2001)	Propõe um framework de avaliação da qualidade de software embasado em um modelo de qualidade. Elementos pertinentes propostos no modelo conceitual: **Características de qualidade**: conjunto de atributos do produto de software pelo qual sua qualidade é descrita e avaliada. Podem ser refinadas em subcaracterísticas de qualidade.

Bases de dados com léxicos da língua inglesa como WordNet[3], Thesaurus.com[4] e MerriamWebster[5] foram utilizadas para identificar os sinônimos dos termos encontrados na literatura. Após analisar os termos, foram incluídos na base apenas aqueles com encaixe no contexto de requisitos e arquitetura de software.

Durante a procura por sinônimos os termos foram divididos em dois grupos: termos simples e termos compostos. Os termos simples são aqueles compostos por uma única unidade lexicográfica, como: performance, latency, security, etc. Os termos compostos são aqueles compostos por mais que uma unidade lexicográfica, tais como time limit, multiple user interface, etc.

Os termos simples foram submetidos às referidas bases de dados lexicográficas e os sinônimos encontrados foram adicionados à base de termos. Os termos compostos por sua vez foram decompostos em tokens que representavam cada unidade lexicográfica do termo. Cada token foi submetido individualmente às bases de dados lexicográficas e os resultados combinados de maneira a obter sinônimos para o termo composto. O Quadro 3.5 exemplifica o processo de obtenção de sinônimos utilizado para povoar a base de dados.

[3] http://wordnet.princeton.edu/

[4] http://thesaurus.com/

[5] http://www.merriam-webster.com/

Quadro 3.5. Obtenção de termos chave sinônimos para o termo chave attack resistance

Termo composto	Unidades lexicais	Sinônimos unidades lexicais	Sinônimos termo composto
attack resistance	attack	intrusion, invasion, encroachment, violation, usurpation, trespass	intrusion resistance, invasion resistance, encroachment resistance, violation resistance, usurpation resistance, trespass resistance, attack immunity, intrusion immunity, invasion immunity, encroachment immunity, violation immunity, usurpation immunity, trespass immunity.
	resistance	immunity	

Não faz parte do escopo deste trabalho criar uma base de dados definitiva. Espera-se que a base de dados criada possa ser expandida conforme novas táticas ou requisitos de qualidade sejam identificados e organizados na literatura. O critério de inserção e organização de termos na base de dados baseia-se na adequabilidade do termo no contexto arquitetural. Assim, requisitos de qualidade cujos impactos na arquitetura não são relatados na literatura corrente foram omitidos. Termos que apesar de referirem-se a coisas diferentes, convergem caso analisados sob um contexto arquitetura e apontam para um mesmo conjunto de características na arquitetura foram inseridos na base de dados como se fossem equivalentes. Desta forma, ainda que os atributos de qualidade maintainability e modifiability representem conceitos diferentes, uma vez que as características do software que apontam a presença destes atributos de qualidade e mesmo as táticas de qualidade utilizadas para alcançar estes atributos são as mesmas, estes conceitos são representados como equivalentes, ou seja, são representados como sinônimos na base de dados.

Outra característica da base de dados criada é que o relacionamento entre atributos de qualidade não foi representado nem utilizado pela abordagem. Porém, verifica-se que estes relacionamentos podem ser deduzidos da extração de conhecimento, uma vez que os padrões abordam diretamente os tradeoffs realizados durante decisões arquiteturais. O Capítulo 5 reserva uma seção sobre como as descrições dos padrões refletem os tradeoffs clássicos entre atributos de qualidade já identificados na literatura.

A Figura 3.2 mostra um fragmento da hierarquia gerada. O ponto de partida é o atributo de qualidade security, o qual é desmembrado em seus sinônimos e parâmetros de atributo de qualidade. Os parâmetros de atributo de qualidade por sua vez são desmembrados em seus sinônimos e táticas pelas quais podem ser alcançados e por último, uma listagem de sinônimos é exibida para cada tática descrita. Os apêndices contidos no final desta dissertação (Apêndices B, C e D) apresentam o conjunto de atributos de qualidade, parâmetros de atributos de qualidade, táticas arquiteturais e sinônimos coletados durante o trabalho.

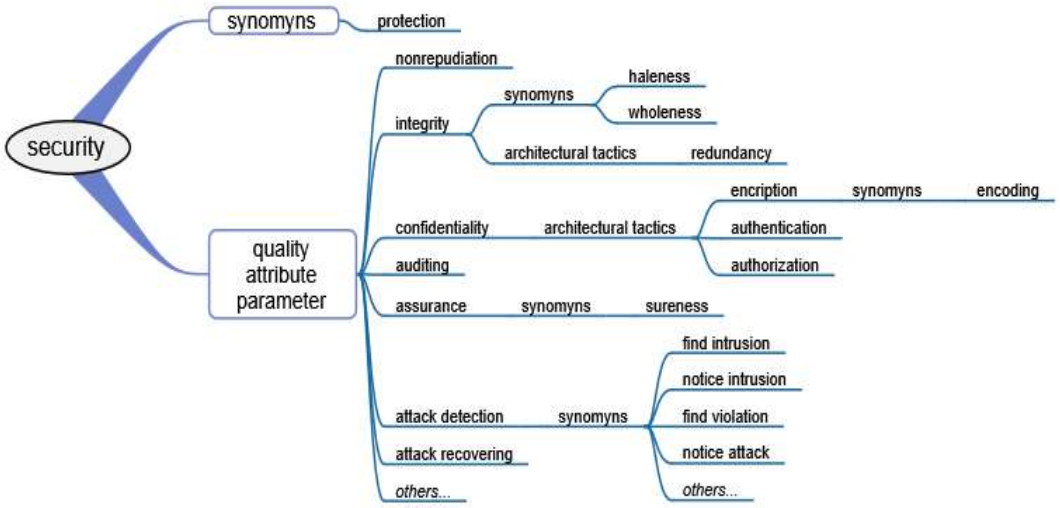

Figura 3.2. Hierarquia de termos chave relacionados ao atributo de qualidade security

4 ABORDAGEM PARA SELEÇÃO DE PADRÕES PARA ARQUITETURA

Este Capítulo apresenta a abordagem criada para facilitar a seleção de padrões para a arquitetura de software. Esta abordagem foi apresentada no Congresso Ibero-Americano em Engenharia de Software – 2014) realizado em Pucón, Chile e publicado nos proceedings da conferência. A abordagem compõe-se de duas fases: a fase de etiquetação dos padrões, responsável por classificar os padrões de acordo com o relacionamento dos mesmos com os termos chave da base de conhecimento e a fase de cálculo das recomendações, responsável por ranquear os padrões de acordo com o grau de afinidade dos mesmos com requisitos de qualidade. (Azevedo, Soares e Braga, 2014).

4.1 Etiquetação de padrões

Os termos chave levantados no capítulo anterior são utilizados para etiquetar os padrões de arquitetura e de projeto. Esta etiquetação é realizada via processamento de linguagem natural: cada termo levantado foi procurado em cada padrão da base de dados e cada ocorrência do termo armazenada juntamente com a seção da ocorrência do termo foi armazenada em uma base de dados.

A procura inicia-se com uma etapa de pré-processamento. As descrições dos padrões e os termos chave são submetidos ao algoritmo de Porter (Porter, 1980), algoritmo este que reduz palavras da língua inglesa a seus radicais. Isto se faz necessário para que a procura não fique sujeita às variações morfológicas de uma palavra. Além disso, palavras consideradas irrelevantes para os propósitos deste trabalho, também conhecidas na literatura de extração de informação como stop words, tais como preposições, conectivos e artigos são excluídas das descrições dos padrões e dos termos chave levantados.

Feito o pré-processamento, realiza-se o cálculo das frequências de cada termo em cada seção de cada padrão. A frequência de um termo chave em uma seção de um padrão é dada pelo somatório das frequências do referido termo chave em cada sentença da seção. A frequência de um termo chave em uma sentença é dada pelo mínimo entre as frequências de cada elemento lexical do termo chave na sentença. A frequência de um elemento lexical em uma sentença é a quantidade de vezes que o elemento lexical ocorre na sentença.

Considerando o algoritmo citado no parágrafo anterior, o termo chave multiple user interface ocorre uma vez na sentença (1) já que cada elemento lexical do termo chave – multiple, user, interface – ocorre uma vez na sentença. O algoritmo proposto considera, portanto que para considerar uma ocorrência de um termo chave em uma sentença, basta que todos os seus elementos lexicais estejam na sentença, independentemente da ordem em que aparecem.

(1) In some rich-client **user interfaces**, **multiple** views of the same data are shown at the same time.

A frequência de um termo na seção de um padrão é então armazenada em uma base de dados para ser utilizada posteriormente durante o cálculo de recomendações.

4.2 Cálculos de recomendações

A procura de termos descrita anteriormente permite encontrar o número de ocorrências de cada termo da base presente em cada seção de cada padrão. Esta informação é utilizada para calcular as recomendações. O cálculo das recomendações é realizado em quatro etapas.

Primeira etapa: Cada ocorrência de termo chave em um padrão é submetida à função M(tipo do termo, seção), definida na Tabela 4.1. A função mapeia a ocorrência de um termo t de um determinado tipo (AQ- Atributo de Qualidade, PAQ-Parâmetro de Atributo de Qualidade, TA-Tática Arquitetural) em uma seção s da descrição de um padrão para um fator F que indica o impacto que cada ocorrência do termo apresenta sobre a adequação deste padrão com este termo. O impacto pode ser positivo ou negativo de acordo com a seção em que o termo ocorre.

A ideia por trás da atribuição dos pesos segue duas dimensões: a importância e grau de coesão de cada seção e a proporção em que cada tipo de termo contribui para o atendimento a um padrão.

Um termo que ocorre na seção Name possui relação direta com o padrão e por isso os maiores pesos foram atribuídos a esta seção. As seções Context e Problem concentram-se em explicar o problema e o contexto do mesmo muitas vezes em termos de seus tradeoffs o que implica em alta taxa de falsos positivos, sendo, portanto as seções com menor peso. As seções Strengths e Liabilities concentram-se em aspectos mais relacionados com o relacionamento com atributos de qualidade ou não e por isso possuem pesos intermediários.

A ocorrência de termos que denotam atributos de qualidade indica uma relação direta do padrão com o requisito de qualidade correspondente ao referido atributo e, portanto os termos deste tipo possuem os maiores pesos. Parâmetros de atributos de qualidade indicam atendimento parcial a um determinado atributo de qualidade e portanto possuem pesos inferiores aos pesos atribuídos a ocorrências de termos que são atributos de qualidade. Raciocínio análogo justifica o fato de ocorrências de táticas arquiteturais apresentarem peso menor que ocorrências de parâmetros de atributo de qualidade.

Tabela 4.1. Função de mapeamento M(t, s)

F=M(t, s)	s = name	s = context	s = problem	s = strengths	s = liabilities
t = AQ	4	1	1	2	-2
t = PAQ	2	0,5	0,5	1	-1
t = TA	1	0,25	0,25	0,5	-0,5

Segunda etapa: Os fatores gerados para cada ocorrência são condensados em uma matriz de afinidade parcial A (termo, padrão) onde cada célula representa o somatório dos fatores de impacto gerados para cada ocorrência do termo nos padrões da base. A Figura 4.1 apresenta as afinidades parciais do padrão Model View Controller com os termos da base. O termo flexibility enquadra-se no conceito de atributo de qualidade. Este termo ocorre uma vez na seção problem, uma vez na seção context e uma vez na seção liabilities do padrão Model View Controller. De acordo com a Tabela 4.1, os pesos de ocorrências de atributos de qualidade nas seções problem, context e liabilities de um padrão são respectivamente 1, 1, -2. Aplicando os pesos às ocorrências temos que:

A(termo = flexibility, padrão = Model View Controller) = 1 x 1 + 1 x 1 + 1 x -2 = 0.

			name	problem	context	strengths	liabilities	Fator de afinidade parcial
AQ	flexibl	flexibl	0	1	1	0	1	0
AQ	modifi	modifi	0	1	0	0	0	1
AQ	maintain	maintain	0	1	0	0	0	1
PAQ	multipl view	multipl view	0	0	0	2	0	2
AQ	exchang	exchang	0	0	0	2	0	2
PAQ	compon complex	compon complex	0	0	0	0	1	-1
PAQ	coupl	coupl	0	0	0	0	1	-1
TA	indirect	indirect	0	0	0	0	1	-0.5
AQ	perform	perform	0	0	0	0	1	-2
TA	cach	cach	0	0	0	0	1	-0.5
TA	encapsul	encapsul	0	0	0	0	1	-0.5
AQ	multipl use interface	multipl use interface	0	0	0	0	0	0
AQ	portabl	portabl	0	0	0	0	1	-2
PAQ	output	output	0	0	0	0	1	-1

Figura 4.1. Afinidades parciais entre os termos da base e o padrão *Model View Controller*

Terceira etapa: A afinidade final de um termo t do tipo = (AQ ou PAQ) com um padrão é calculada através do somatório de A (t, p) com as afinidades parciais geradas para todos os termos que estiverem abaixo do termo t na hierarquia de termos. Utilizando a Figura 3.2 para exemplificar, a afinidade final do parâmetro de atributo de qualidade integrity com um padrão p é a soma de A (t=integrity, p) + A (t=wholeness, p) + A (t=haleness, p) + A (t = redundancy, p).

Quarta etapa: Um conjunto C = {t_1, t_2, t_3, $t_{4...}$, t_n} composto por termos que denotam atributos de qualidade e parâmetros de atributo de qualidade é fornecido como entrada para a recomendação de padrões. Esses termos estão entre os requisitos do sistema sendo desenvolvido e são informados pelo usuário da abordagem. Considerando o sistema com os requisitos do conjunto C, o grau de recomendação φ de um padrão **p** da base de conhecimento para o sistema é calculado somando-se as afinidades finais de todos os termos do conjunto C em relação a cada padrão da base de dados, matematicamente:

$$\varphi(C_t, p) = \sum_{i=1}^{n} A(t_i, p)$$

Os padrões que obtiverem maior grau de recomendação com o conjunto de termos são os mais recomendados para o sistema em questão. Especificamente:

1. Os padrões com maior valor positivo de φ são os mais recomendados;
2. Os padrões com menor valor negativo de φ são os menos recomendados;
3. Os padrões com valores nulos ou intermediários de φ são considerados neutros.

A Figura 4.2 apresenta a interface de usuário do protótipo através da qual é possível obter as recomendações de padrões. O usuário pode escolher quantos pares atributos de qualidade / parâmetros de atributo de qualidade achar necessário para obter a recomendação, havendo a possibilidade de omitir o parâmetro de atributo de qualidade e obter recomendações considerando apenas o atributo de qualidade. Para cada par atributo de qualidade / parâmetro atributo de qualidade, será considerado no conjunto C utilizado para obter a recomendação somente o parâmetro de atributo de qualidade caso o mesmo seja especificado, ou somente o atributo de qualidade caso parâmetro de

atributo de qualidade seja omitido. Considerando o exemplo em questão, têm-se o conjunto C = {confidentiality, multiple user interface}.

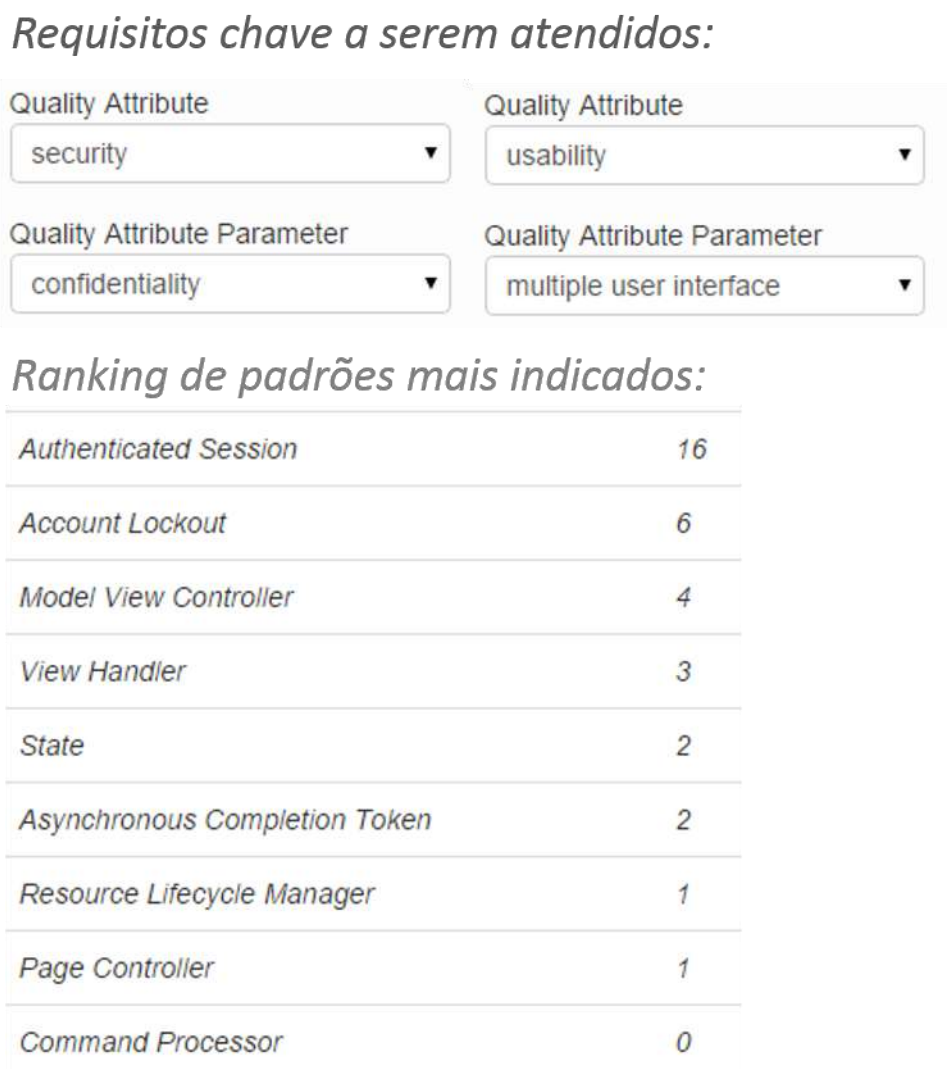

Figura 4.2. Tela de recomendação do protótipo utilizado para avaliar a abordagem

5 AVALIAÇÃO DA ABORDAGEM

Uma vez proposta a abordagem para a recomendação de padrões, o passo seguinte consistiu em verificar a sua efetividade através do desenvolvimento de um protótipo que implementa a abordagem.

O protótipo é composto por dois módulos: módulo de procura e módulo de recomendação. O módulo de procura é responsável por extrair informação a partir da descrição textual de padrões baseada na utilização de termos chave. O módulo de recomendação é responsável por traduzir os resultados da procura em recomendações baseadas nos requisitos da aplicação, informados por meio de um conjunto $C = \{t_1, t_2, t_3, t_4, ..., t_n\}$, onde cada t_i é um termo que representa um requisito não funcional da aplicação.

A validação foi realizada segundo três perspectivas: capacidade da abordagem de extração de conhecimento representar os padrões em termos de suas características de qualidade; capacidade da abordagem de recomendar padrões apropriados a cada problema de software e capacidade da abordagem de identificar tradeoffs comuns entre atributos de qualidade ou identificar mecanismos de resolução de conflitos entre os mesmos.

5.1 Avaliação sob a perspectiva da extração de conhecimento

Os padrões descritos em (Gamma et al., 1995) e os descritos nas duas primeiras edições do POSA (Buschmann et al., 1996) e (Schmidt et al., 2000) além de alguns descritos no terceiro volume (Kircher e Jain, 2004), no livro sobre padrões para segurança (Kienzle et al., 2006) e no MSDN – Microsoft Development Network (Microsoft, 2014) compõem o conjunto de 60 padrões submetidos a procura por termos chave. Uma listagem com os referidos padrões pode ser vista no Apêndice A. A Tabela 5.1 apresenta a quantidade de padrões de cada fonte utilizados nos testes da prova de conceito.

Tabela 5.1. Quantidade de padrões utilizados no teste da prova de conceito de cada fonte consultada

Fonte	Número de Padrões
GAMMA (Gamma et al., 1995)	22
SECURITY (Kienzle et al., 2006)	2
MSDN (Microsoft, 2014)	1
POSA1 (Buschmann et al., 1996)	15
POSA2 (Schmidt et al., 2000)	17
POSA3 (Kircher e Jain, 2004)	3

O livro do (Gamma et al., 1995) descreve somente padrões de projeto, enquanto os livros da série POSA descrevem tanto padrões arquiteturais, quanto padrões de projeto. Até o segundo volume do POSA, os autores se preocuparam em deixar explícito em cada padrão se o mesmo tratava-se de um padrão arquitetural ou de projeto. A partir do terceiro volume esta preocupação entrou em desuso e os autores passaram a não deixar explícito o tipo de padrão sendo descrito. O catálogo de padrões para segurança também se abstêm desta preocupação. A Tabela 5.2 mostra a quantidade de padrões contida na base de dados durante os testes realizados para cada tipo de padrão.

Tabela 5.2. Quantidade de padrões por tipo de padrão

Tipo do Padrão	Número de padrões
Arquitetural	13
Projeto	41
Indefinido	6

Pouco mais de duas centenas de termos chave foram utilizados nos testes sendo a grande maioria sinônimos de outros termos chave como mostra a Tabela 5.3.

Tabela 5.3. Quantidade de termos na base de conhecimento estratificado por tipo de termo

Tipo Termo	Quantidade de Termos
Atributo de qualidade	10
Parâmetros de atributo de qualidade	32
Sinônimo	140
Tática arquitetural	42
Total	223

Uma vez executada a procura, constatou-se que a descrição dos padrões continha ocorrências de atributos de qualidade, parâmetros de atributo de qualidade e táticas arquiteturais em uma taxa de 12 ocorrências por padrão, sendo a maioria delas ocorrências de atributos de qualidade.

Outro ponto interessante é que ainda que os atributos de qualidade correspondam a menos de 10% dos termos da base, mais de 30% das ocorrências são de atributos de qualidade. De fato, uma análise superficial de descrições dos padrões deixa claro o quanto os padrões são descritos em termos de seus benefícios e malefícios aos atributos de qualidade, especialmente quando se trata da descrição de um padrão arquitetural como pode ser observado na Tabela 5.4.

Tabela 5.4. Análise de resultados por tipo de padrão

Tipo do padrão	Quantidade de padrões	Media de ocorrência de termos chave por padrão
Indefinido	6	28.3333
Arquitetural	13	25.8462
Padrão	40	10.625
Idiom	1	5

A probabilidade de um termo chave ocorrer em um padrão de arquitetura é pelo menos duas vezes maior que em um padrão de projeto o que é razoável já que como dito anteriormente, um dos papéis do projeto de arquitetura de software é operacionalizar atributos de qualidade, sendo que esta operacionalização pode ser realizada através da aplicação de padrões arquiteturais e em menor escala, através da aplicação de padrões de projeto se aplicados sob uma perspectiva arquitetural.

O elevado número de ocorrências de termos chave nos padrões não classificados entre padrões de projeto ou arquiteturais pelos seus autores deve-se a dois fatos: metade dos padrões indefinidos são padrões restritos para o domínio segurança retirados do catálogo criado por (Kienzle et al., 2006). Os padrões deste catálogo são descritos utilizando template próprio, que preconiza a exposição dos atributos de qualidade afetados pelo padrão. Os outros três padrões foram retirados do terceiro volume do POSA criado em 2005, época em que a importância de relacionar decisões arquiteturais e atributos de qualidade já havia sido destacada como tendência de pesquisa, podendo isto ter influenciado para que as descrições de padrões evidenciassem este tipo de relação de forma mais enérgica. Algum tempo depois a inserção dos padrões na base de conhecimento, constatou-se que um dos padrões tratava-se de um idiom: tipo de padrão de baixo nível que descreve como implementar aspectos locais e particulares de componentes ou relacionamento entre os mesmos usando recursos de uma linguagem de programação específica. A expectativa quanto a este tipo de padrão é que haja pouca ou nenhuma relação com atributos de qualidade já que a preocupação de um idiom é estritamente funcional.

De fato, apesar da amostra insignificante de idioms na base de dados, o fato de a ocorrência média de termos chave ser cinco vezes menor que em um padrão arquitetural

corrobora o fato de que quanto menor o nível de abstração de um padrão menor a relação dos mesmos com requisitos de atributo de qualidade.

Como justificado anteriormente, padrões de projeto foram inseridos na base de dados devido ao fato de estes poderem ser aplicados sob uma perspectiva arquitetural. Há inclusive relatos na literatura de aplicações arquiteturais de padrões de projeto como o Interpreter, Observer, Composite e Mediator (Avgeriou e Zdun, 2005), (Giesecke, Hasselbring e Riebisch, 2007). O gráfico da Figura 5.1 mostra quantas ocorrências de termos chave possui cada padrão descrito em (Gamma et al., 1995).

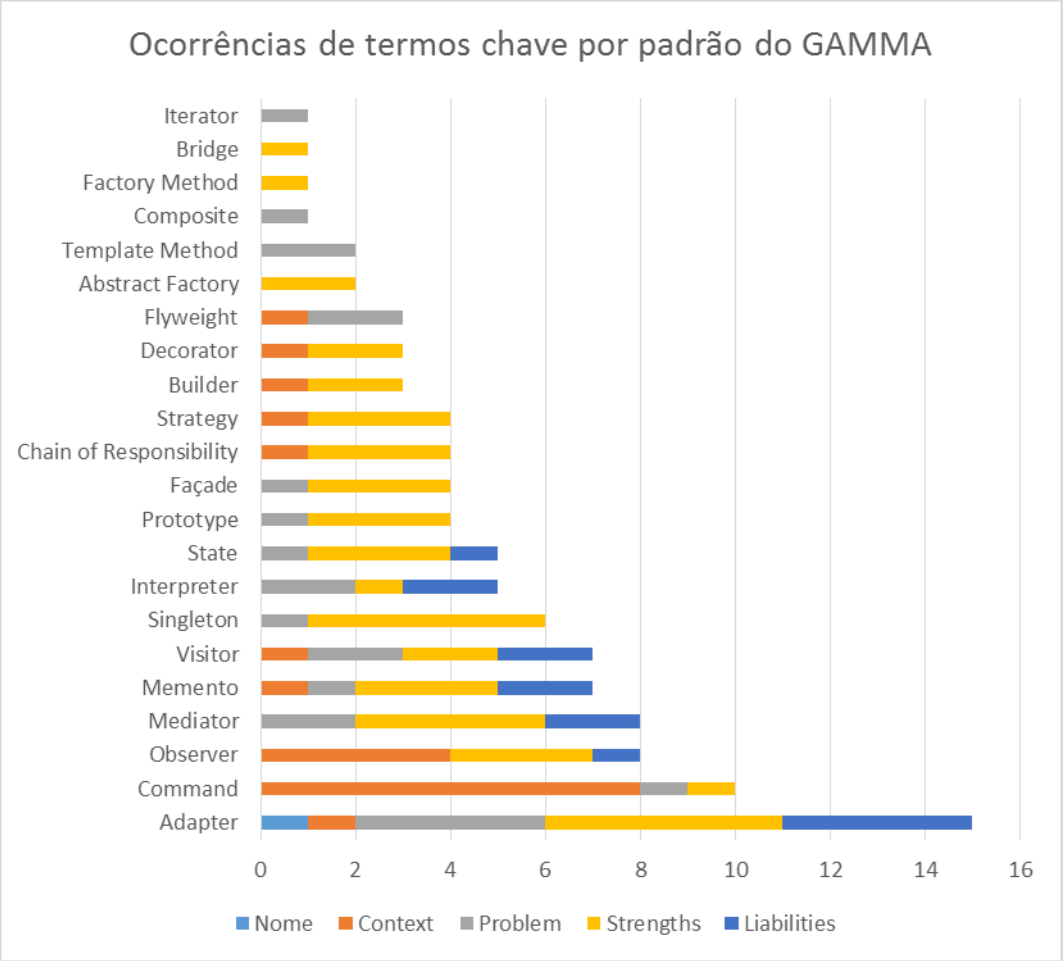

Figura 5.1. Frequência de termos chave em padrões de projeto

Os extremos do gráfico reforçam os argumentos apresentados anteriormente quanto à categorização de padrões em padrões de projeto ou arquiteturais estar relacionada a uma questão de perspectiva. À exceção do padrão Composite, os padrões com relatos de aplicação sob uma perspectiva arquitetural são os mesmos que apresentam as maiores quantidades de ocorrência de termos chave dentre os padrões da base. Por outro lado, o padrão Iterator, categorizado como padrão de projeto, mas cujo

nível de abstração é baixo a ponto de ter construções específicas para linguagens de programação como Java e C++, apresenta uma das menores quantidades de ocorrência termos.

Convém analisar também o comportamento da procura quanto às seções dos padrões onde a procura é realizada. Como esperado, as seções de benefícios e fraquezas são as com maior frequência de termos chave.

5.2 Avaliação sob a perspectiva das recomendações

Os impactos de padrões arquiteturais descritos em (Buschmann et al., 1996) sobre um determinado conjunto de atributos de qualidade foram analisados manualmente por Harrison e Avgeriou em (Harrison, Avgeriou, 2007) e os resultados obtidos foram comparados com aqueles obtidos pelo protótipo que implementa a abordagem proposta.

O experimento consistiu das seguintes etapas: para cada atributo de qualidade a analisado em (Harrison e Avgeriou, 2007) determinou-se um conjunto teste $C_T = \{a\}$; o conjunto C_T foi utilizado como entrada para a prova de conceito que determinou quais os padrões mais recomendados, menos aconselhados e indiferentes para um sistema onde C_T represente os requisitos não funcionais mais importantes. Os padrões não analisados por Harrison e Avgeriou foram excluídos do conjunto solução, restringindo o experimento aos oito padrões e seis atributos de qualidade analisados em (Harrison e Avgeriou, 2007).

Os resultados obtidos são apresentados na Tabela 5.5, onde cada célula apresenta uma dupla PC/HA em que PC representa o resultado obtido pela prova de conceito e HA o resultado obtido manualmente por Harrison e Avgeriou. As células destacadas em cinza indicam onde os resultados obtidos automaticamente pela prova de conceito e manualmente por Harrison e Avgeriou divergem. As letras N, S e L denotam relações de neutralidade (neutral), benefício (strength) e fraqueza (liability) entre padrões e termos. Estes valores indicam respectivamente que a aplicação do padrão "p" é recomendada, não recomendada, e indiferente em um sistema em que *"a"* seja o único atributo de qualidade desejado. O símbolo (*) indica que a aplicação do padrão possui impactos tanto positivos quanto negativos sobre o atributo de qualidade.

Tabela 5.5. Comparação entre os resultados obtidos através do protótipo da abordagem e os resultados obtidos por Harrison e Avgeriou

	Usability	Security	Maintainability	Performance	Reliability	Portability
Layers	N/N	N/S	S/S	L/L	L/S	S/S
Pipes and Filters	N/L	N/L	S/S	S/*	L/L	S/S
Blackboard	N/N	N/L	S/S	L/L	N/N	N/N
MVC	S/S	N/N	L/L	L/L	N/N	L/L
PAC	S/S	N/N	S/S	L/L	N/N	N/S
Microkernel	N/N	N/N	S/S	L/L	S/S	S/S
Reflection	N/N	N/N	S/S	L/L	N/L	N/S
Broker	N/S	N/S	S/S	L/N	L/L	S/S

Os resultados foram comparados de forma a obter medidas de similaridade, precisão e recall, métricas comumente utilizadas para avaliar trabalhos relacionados a extração de informação. Seguem as definições destas medidas adaptadas para o contexto deste trabalho:

Similaridade. A razão entre a quantidade de recomendações da prova de conceito que coincidem com os resultados obtidos por Harrison e Avgeriou (2007) e o total de recomendações analisadas.

Precisão. A precisão é calculada segundo dois pontos de vista neste trabalho: (i) capacidade da prova de conceito recomendar a aplicação de um padrão corretamente, a qual denominamos precisão de recomendação; (ii) capacidade da prova de conceito 'desaconselhar' a aplicação de um padrão de forma correta, a qual denominamos precisão de desaconselhamento.

Recall. Assim como a precisão, o recall é calculado segundo dois pontos de vista: (i) capacidade da prova de conceito recomendar os padrões que devem ser recomendados dado um atributo de qualidade, a qual denominamos recall de recomendação; (ii) capacidade da prova de conceito 'desaconselhar' a aplicação dos padrões que devem ser desaconselhados dado um atributo de qualidade, a qual denominamos recall de desaconselhamento.

A Tabela 5.5 permite inferir que a similaridade entre as classificações obtidas por (Harrison e Avgeriou, 2007) e pela ferramenta apresentada por este trabalho foi de 77%. Ao analisar a ferramenta em termos de precisão de recomendação encontrou-se que a prova de conceito recomenda padrões com precisão máxima (100%). Por outro lado,

cerca de 30% das recomendações obtidas manualmente por Harrison e Avgeriou (2007) não foram obtidas pela ferramenta, o que confere um recall de recomendação de 70%. A ferramenta não mostrou a mesma eficácia ao desaconselhar o uso de padrões dado um atributo de qualidade, obtendo precisão de desaconselhamento de 83%, sendo a taxa de recuperação ao desaconselhar (recall) da ordem de 71%.

O índice de precisão de desaconselhamento justifica-se em parte devido a menor quantidade de informação sendo processada para identificar esta relação, uma vez que quatro das cinco seções consideradas durante a procura estão atreladas à descrição de relações positivas entre padrões e termos e apenas uma às relações negativas. Outro problema é que a descrição de fraquezas costuma ser feita em termos de tradeoffs a serem realizados ao aplicar o padrão, acarretando na ocorrência de termos que representam fraquezas do padrão em seções não apropriadas. Assim, introduz-se uma fraqueza ao custo de mencionar um benefício do padrão. Esta menção acaba por tornar-se um falso positivo na classificação. Os resultados obtidos para recall ocorrem em boa parte, devido à necessidade de uma análise mais profunda do padrão para identificar algumas relações. Nestes casos, o impacto de um padrão sobre um atributo de qualidade não é consequência direta, mas sim, inferida a partir de outros aspectos deriváveis da aplicação do padrão e, portanto difíceis de identificar via procura de termos.

5.3 Relações entre atributos de qualidade identificadas nas recomendações

Os atributos de qualidade relacionam-se entre si de modo que o atendimento a um requisito pode reforçar o atendimento a outro atributo de qualidade ou mesmo prejudicar o alcance a uma determinada característica de qualidade.

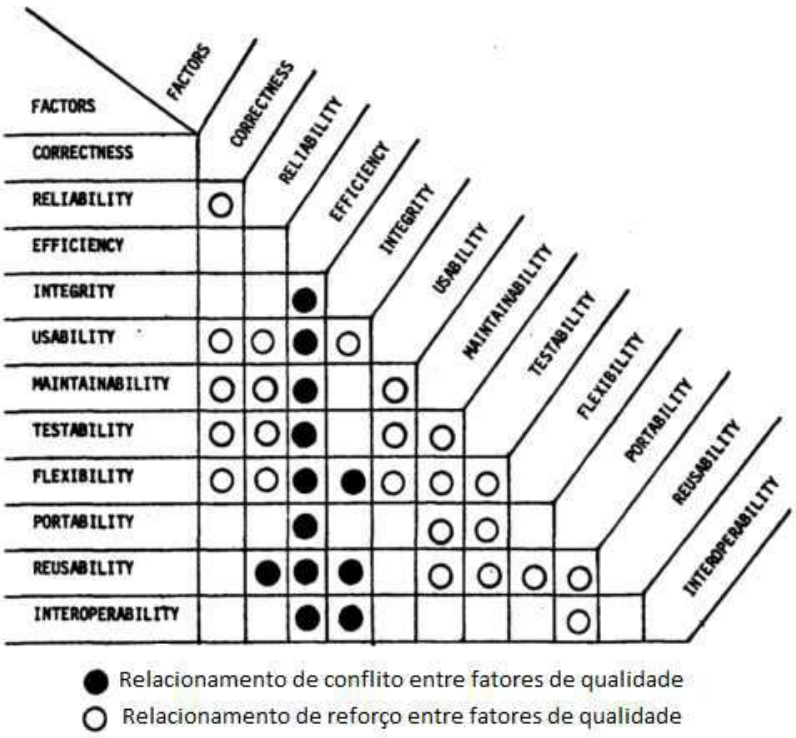

Figura 5.2. Relacionamento entre fatores de qualidade. Retirado de (Mccall, Richards e Walters, 1977)

A Figura 5.2 apresenta os resultados da investigação realizada por Mccall, Richards e Walters (1977) sobre o relacionamento entre fatores de qualidade de software. Este trabalho, além de identificar alguns fatores de qualidade pertinentes durante o desenvolvimento de software, documentou o tipo de relacionamento entre eles, em alguns casos justificando o tipo de relacionamento. Outros autores, como por exemplo Bass, Clements e Kazman (2003) relacionam alguns tradeoffs comuns entre requisitos de qualidade. Uma parte da validação deste trabalho foi verificar se as relações entre atributos de qualidade documentadas na literatura eram capturadas pela abordagem de extração de conhecimento proposta pela abordagem. Outra parte foi identificar casos em que estas relações não coincidiam e detectar o motivo da não coincidência. Neste âmbito, as seguintes possibilidades eram consideradas:

i) A existência de falsos positivos prejudicava a procura fazendo com que o relacionamento entre um atributo de qualidade e um padrão fosse mapeado incorretamente, como fora o caso do padrão Layers com o atributo de qualidade Reliability mostrado na seção anterior;
ii) O padrão atuava na resolução do tradeoff em questão, sugerindo algum meio termo entre os atributos de qualidade;

iii) O relacionamento entre os atributos de qualidade em questão era amenizado por um terceiro atributo de qualidade dominante no contexto de aplicação do padrão sendo analisado;

iv) O tipo de relacionamento de um atributo de qualidade com outro atributo de qualidade era relativo ao parâmetro de atributo de qualidade considerado. Como veremos nas próximas seções, ainda que modificabilidade tenha um relacionamento conflituoso clássico com performance, caso performance seja analisada sob o ponto de vista de gerência de recursos, esta pode ter um relacionamento sinérgico com modificabilidade uma vez que padrões que facilitam a gerência de recursos possibilitam que estes recursos sejam modificados, trocados, ou inseridos em uma arquitetura mais facilmente.

5.3.1 Modificabilidade e Desempenho

Os atributos de qualidade modificabilidade (modifiability) e desempenho (performance) são atributos classicamente conflitantes, como descrito em diversas fontes. Mccall, Richards e Walters (1977) justifica dizendo que otimização de código implica no aumento de complexidade e em acesso direto a recursos o que degrada a capacidade de modificação do software. Por outro lado, aplicar técnicas de modularização e aumento do grau de abstração do código para que o mesmo se torne mais entendível e modificável geralmente implica em overhead de comunicação entre os módulos, resultando em operações menos eficientes.

A prova de conceito identificou que 38 dos 60 padrões utilizados na extração de conhecimento preocupam-se com estes dois atributos de qualidade simultaneamente, o que confirma o forte grau de relacionamento entre os mesmos. O caráter conflitante desta relação se confirma também uma vez que 65% dos padrões que beneficiam um destes atributos de qualidade prejudica o alcance ao outro e vice-versa. Os outros 35% denotam uma relação de reforço entre os dois atributos.

Uma análise detalhada destes padrões fez-se necessária para que esta porcentagem de reforço entre atributos classicamente conflitantes pudesse ser justificada. Apresentamos a seguir um resumo desta análise.

Um dos padrões que apontaram maior taxa de reforço entre os referidos atributos foi o padrão Resource Lifecycle Manager. De fato, a proposta do padrão de facilitar a gerência do ciclo de vida dos recursos utilizados por sistemas intensivos provê meios de manter e modificar a maneira como os mesmos são utilizados o que reflete positivamente no atributo de qualidade modifiability. Além disso, esta mesma gerência de recursos facilitada impacta na capacidade de rearranjá-los de forma a otimizar o uso do mesmos, o que estabelece ligação direta do padrão com o parâmetro de atributo de qualidade resource usage relacionado ao atributo de qualidade performance.

O padrão Pipes and Filters repete o mesmo comportamento. Os benefícios relacionados à modificabilidade advindos da aplicação do padrão são inquestionáveis dado a flexibilidade obtida seja pela facilidade de trocar filtros ou recombiná-los para obter novos resultados. A questão da performance é controversa já que o padrão favorece a execução de tarefas paralelas, mas por outro lado o overhead gerado pela comunicação tende a anular esta contribuição. De fato, ainda que a procura aponte performance como ponto positivo do padrão, a mesma indica um relacionamento negativo do padrão com o parâmetro de atributo de qualidade latency, o que está de acordo com a relação conflituosa de performance com modificabilidade. O que se verifica é que performance do ponto de vista de tempo de resposta diverge de modificabilidade, no entanto a relação é sinérgica se analisada sob a perspectiva de uso eficiente dos recursos, já que possibilita a paralelização de atividades. O padrão Master Slave preconiza dividir o trabalho para conquistar. Possibilita a paralelização de tarefas (ainda que isto implique em aumento de complexidade) e, portanto pode impactar positivamente em performance. Além disso, a existência de um "mestre" propicia a separação do código que realiza os trabalhos daquele que particiona, divide e junta os resultados, aumentando a modificabilidade. O padrão Component Configurator apresenta o mesmo comportamento.

A partir desta análise, percebe-se que o relacionamento entre atributos de qualidade não pode ser generalizado sem considerar as perspectivas sob as quais estes estão sendo analisados. A Figura 5.3 contrapõe o relacionamento entre modificabilidade e performance como comumente descrito na literatura (a) com o relacionamento entre os mesmos considerando os parâmetros de atributo de qualidade de performance considerados neste trabalho (b).

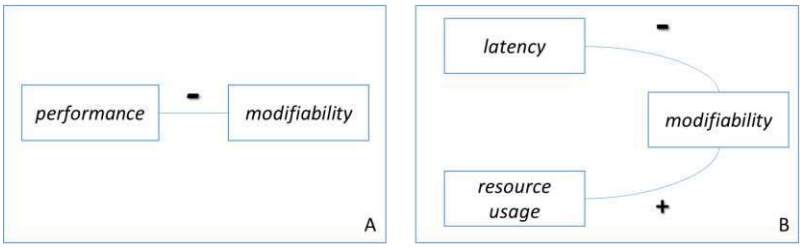

Figura 5.3. Relacionamento entre os atributos de qualidade performance e modificabilidade

No caso do Asynchronous Completion Token a sinergia entre os dois atributos de qualidade advém da baixa complexidade das estruturas de dados envolvidas no padrão, o que as torna fáceis de manter e de análise eficiente. Trata-se de um caso em que a utilização do padrão atua como uma forma de resolver conflitos entre atributos de qualidade.

O padrão Proxy apresenta a mesma característica de acordo com a procura, sendo a justificativa inclusive elaborada em outras fontes como (Lin, 2009) e ilustrada através da Figura 5.4. No caso, o modelo (a) trata de um projeto simples que permite um cliente acessar diretamente um objeto "pesado" (por exemplo, uma imagem) que pode resultar em baixo tempo de resposta, uma vez que criar um objeto "pesado" é custoso. O modelo (b) introduz um mediador que disponibiliza informações estendidas do objeto real, como por exemplo, o tamanho da imagem. Assim, no caso de o cliente querer apenas ter acesso a informações estendidas, como por exemplo, tamanho, autor ou data de criação da imagem, o cliente acessa este mediador e não gasta tempo criando a imagem para extrair estas informações. Melhora-se a performance ao custo de o cliente precisa lidar com dois objetos de acordo com sua necessidade, o que implica em aumento de complexidade e consequente impacto negativo em modificabilidade. O modelo (c) aplica o padrão Proxy para resolver o conflito entre performance e modificabilidade. Através da abstração do agente mediador e do agente real em um objeto em comum, chamado Subject, a complexidade no cliente é diminuída uma vez que não é mais necessário distinguir o objeto mediador do objeto real. Além disso, performance é mantida uma vez que o cliente ainda pode conseguir informação estendida acessando o objeto proxy.

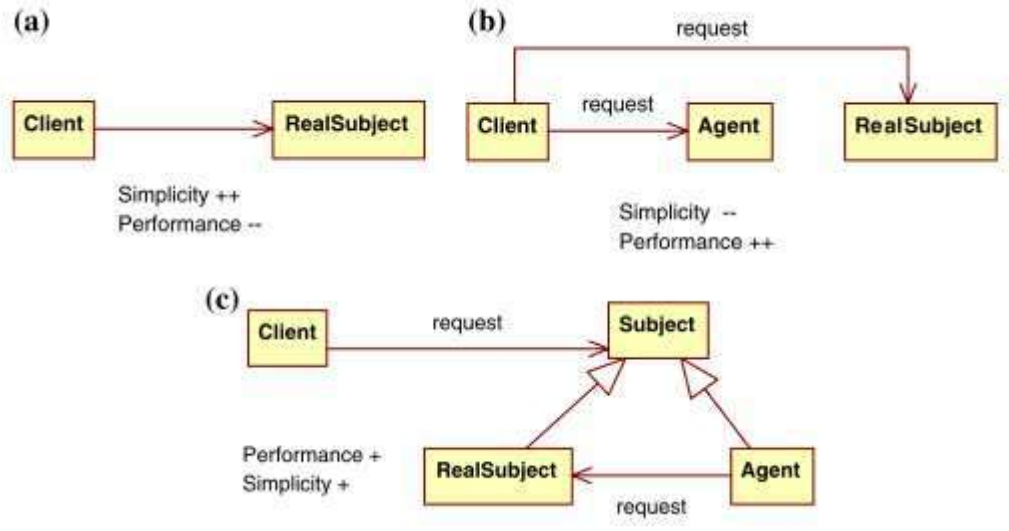

Figura 5.4. Padrão Proxy atuando como resolvedor de conflitos. Retirado de (LIN, 2009)

O padrão Presentation Abstraction Control apresenta um comportamento interessante: o alto grau de modularidade imposto pelo padrão (muitos agentes) torna a arquitetura complexa. Assim a complexidade inerente ao padrão suplanta os benefícios da modularidade proposta fazendo que a abordagem proposta identifique um relacionamento negativo do padrão com modificabilidade. O overhead gerado pela comunicação entre os agentes aumenta o tempo de resposta e impacta negativamente na eficiência. Não é possível, porém conectar os dois atributos neste caso, já que o problema com performance e gerado pela modularização excessiva, que deveria melhorar a modificabilidade do sistema, mas como resulta em aumento da complexidade, acaba impactando negativamente na capacidade de modificar o software.

Uma breve análise dos padrões que repetiam o relacionamento conflituoso entre modificabilidade e performance documentado na literatura indicou as seguintes razões para tal tipo de relacionamento:

Os padrões que utilizam níveis de indireção para tornar o software mais modificável acabam lesando performance devido ao overhead gerado pela comunicação entre estes níveis. Alguns dos padrões com esta característica são Layers, Reflection, Microkernel, Wrapper Facade.

Outros padrões preconizam por deixar transparente a localização dos componentes da arquitetura (preocupação típica de ambientes distribuídos) e acabam por aumentar o tempo de resposta devido à necessidade de executar tarefas de

localização, configuração e inicialização de recursos. Alguns dos padrões que se encaixam nesta característica são os padrões Broker e Client Dispatcher Server.

Alguns padrões atuam de forma a possibilitar que componentes de software possam comunicar com outros componentes sem que isso implique no aumento do acoplamento entre estes componentes. O problema é que as estratégias utilizadas por estes padrões podem acarretar em efeitos cascata caso não sejam implementados e utilizados com atenção, degradando a performance da arquitetura. Enquadram-se neste grupo padrões como o Observer, Interpreter.

5.3.2 Confiança e Desempenho

(Mccall, Richards, Walters, 1977) não identifica nenhum relacionamento entre confiança (reliability) como pode ser observado na Figura 5.2, porém outros autores, como (Bass, Clements e Kazman, 2003) identificaram que atividades que aumentam a confiabilidade de um sistema como manter cópias atualizadas de dados, replicação de componentes, verificação periódica de componentes em falha são atividades que consomem recursos e, portanto afetam negativamente a questão da eficiência.

A prova de conceito identificou 14 padrões em que performance e reliability são preocupações abordadas concomitantemente e constatou que em torno de 79% destes padrões relacionavam-se de maneira inversamente proporcional.

Dentre os padrões que não confirmaram este comportamento, alguns deles devem-se a ocorrência de falsos positivos em suas respectivas descrições: é o caso do padrão Layers em relação a confiança e do padrão Lookup que insere falsos positivos ao discutir justamente a relação conflituosa entre eficiência e confiança em uma das seções do padrão.

Por outro lado, os resultados obtidos pela extração de conhecimento para o padrão Broker são justificáveis ainda que não coincidam com a relação conflituosa proposta na literatura. Este padrão apresenta baixa tolerância a falhas devido à presença de um ponto único de falha o que justifica seu relacionamento negativo com reliability e apresenta baixa performance por preconizar transparência na localização dos componentes. A análise das justificativas dos relacionamentos negativos entre os dois atributos de qualidade com o referido padrão não permite encontrar nenhum ponto de conexão entre elas. Uma hipótese para este relacionamento convergente é que o fato de que a premissa

de que se um padrão beneficia um dos atributos de qualidade, enfraquece o outro ser forte não significa que a premissa de que se um padrão enfraquece um atributo de qualidade então beneficia o outro também é forte.

5.3.3 Testabilidade e Segurança

O atributo de qualidade testabilidade está diretamente relacionado a capacidade de se observar e controlar o ambiente e os estados internos dos algoritmos executados, o que vai de encontro com alguns dos princípios fundamentais de segurança que requer observabilidade e controlabilidade mínimos (Natale et al., 2010).

O único padrão que trata destes dois atributos de qualidade simultaneamente levantado pela prova de conceito está de acordo com este tradeoff. De fato, a descrição do padrão Component Configurator deixa explícito que sua aplicação facilita decompor a aplicação em componentes de maneira que estes componentes possam ser testados independentemente e então serem combinados dentro de processos. Entretanto, deixa claro também que é uma opção menos segura, uma vez que facilita a substituição de componentes por componentes impostores.

5.3.4 Testabilidade e Confiança

Mccall, Richards e Walters (1977) identifica que um software que apresenta um alto grau de testabilidade tende a ser um software confiável. Jimenez, Taj e Weaver, (2005) complementa elucidando que um software que passou por sucessivas fases de teste e verificação e apresenta alto grau de testabilidade é um software de confiança.

Considerado isto, os resultados obtidos pela prova de conceito não estão de acordo com as afirmações obtidas na literatura.

De fato, para dois dos padrões onde identifica-se a relação antagônica entre os atributos de qualidade, isso ocorre devido a ocorrências de falsos positivos nas descrições dos padrões (Layers, Presentation Abstract Control).

No entanto o diagnóstico de relacionamento antagônico entre testabilidade e confiança para outros dois padrões não se deve a problemas na extração de conhecimento e pode ser justificado analiticamente.

O padrão Component Configurator facilita a decomposição do software em componentes e consequentemente o teste independente destes componentes, porém a configuração errada de um componente pode afetar negativamente a execução de outro componente (difícil isolar comportamento).

O padrão arquitetural Broker, pelo fato de facilitar a decomposição da aplicação em serviços impacta positivamente testabilidade, mas por outro lado apresenta baixa tolerância a falhas o que reflete negativamente na confiabilidade.

O padrão Command Processor melhora a testabilidade já que um comando é um ponto ideal de entrada de teste. Além disso, a implementação de mecanismos que aumentam a segurança tais como logging e rollback de operações é facilitada pela implementação deste padrão.

A análise dos padrões em que ocorrem termos relacionados a testabilidade e confiança concomitantemente não permite estabelecer uma relação de causa e efeito entre os dois atributos, nem mesmo nos casos em que o resultado obtido pela extração coincide com o relacionamento apontado na literatura. O texto de Mccall, Richards, Walters (1977) não deixa explícito o raciocínio utilizado inferir o relacionamento de reforço entre os dois atributos de qualidade e uma análise cuidadosa da justificativa dada em (Jimenez, Taj e Weaver, 2005) mostra que um software testado tende a ser um software confiável, o que decerto não pode ser garantido de um ponto de vista arquitetural.

5.3.5 Usabilidade e Disponibilidade

Apesar de nenhum relacionamento entre os atributos de qualidade usabilidade e disponibilidade estar documentado na literatura, ao considerar todos os casos detectados pela ferramenta de extração de conhecimento em que ambos os atributos são preocupações percebe-se que em todos eles estes atributos se reforçam. Com o objetivo de verificar se existe de fato uma relação entre estes dois atributos, analisou-se os padrões em que isto ocorria.

O padrão Command Processor favorece a implementação de táticas de redução e recuperação de falhas e contribui positivamente, portanto com o atributo de qualidade disponibilidade. A possibilidade de implementar comandos de desfazer, repetir, etc., promove usabilidade. Não é possível, no entanto, estabelecer relação explícita entre usabilidade e disponibilidade.

A facilidade de implementar técnicas de recuperação de ambiente de produção de arquiteturas que utilizam o Component Configurator tende a ser alta devido ao controle centralizado de componentes facilitar identificação de componentes falhos e reinicialização dos mesmos. A possibilidade do sistema inicializar, suspender, terminar componentes dinamicamente em tempo de execução facilitada pela aplicação do Component Configurator contribui para aspectos de usabilidade do sistema. O usuário pode, por exemplo, escolher a configuração que lhe for mais conveniente sem que para isso o software tenha que ser recompilado.

O Presentation Abstraction Control é um padrão que preconiza interagir através de agentes especializados em aceitar entradas e mostrar dados ao usuário e, portanto apresenta-se como opção em termos de usabilidade. Além disso, a possiblidade de implementar agentes responsáveis por identificar e tratar erros faz com que o mesmo se apresente como opção em termos de disponibilidade.

O padrão Authenticated Session impacta diretamente em usabilidade uma vez que permite que um usuário acessar múltiplas páginas de acesso restringido sem que para isso tenha que autenticar várias vezes. O relacionamento com disponibilidade, no entanto advém de um falso positivo assim como o relacionamento do padrão Lookup com usabilidade.

A disponibilidade provida pelo padrão Account Lockout é baixa devido à possibilidade de um ataque tentar supor a senha de vários usuários e isso resultar no bloqueio destes vários usuários devido ao estouro das tentativas erradas permitidas de suposição de senhas para estes usuários. A usabilidade é impactada pelo mesmo motivo, já que se o usuário erra a senha o limite de vezes imposto ele fica impedido de continuar a tentar.

Analisando estes casos é possível perceber que a não disponibilidade de um sistema, seja por qual motivo for, gera impactos negativos em usabilidade já que o usuário fica impedido de usar o sistema. Esta justificativa é evidente especialmente nos padrões retirados do catálogo de padrões para segurança (Account Lockout e Authenticated Session). Percebe-se também que padrões que promovem iniciativa do próprio sistema quanto a ações necessárias, como por exemplo, reiniciar um servidor, aumentam a usabilidade, uma vez que antecipam uma necessidade do usuário.

6 CONSIDERAÇÕES FINAIS

Este trabalho propôs uma abordagem para selecionar padrões durante o projeto de arquitetura de software apoiada no uso de termos chave e de processamento de linguagem natural. Uma base de conhecimento inicial foi construída utilizando padrões de projeto e de arquitetura e um conjunto de termos chave composto por atributos de qualidade, parâmetros de atributo de qualidade, táticas arquiteturais e termos sinônimos.

Os resultados apontaram que os termos chave levantados são encontrados nas descrições dos padrões, em especial aqueles classificados como padrões arquiteturais ou aqueles que apesar de serem classificados com padrões de projeto apresentam potencial para serem aplicados sob uma ótica arquitetural.

A utilização de processamento de linguagem natural associada ao formato de entrada da abordagem (termos que denotam requisitos não funcionais) facilita a aquisição de conhecimento uma vez que não é necessário um especialista para reescrever o padrão em um formalismo entendido pela abordagem. Isto contribuiu para que a abordagem apresentasse escalabilidade quanto à quantidade de conhecimento considerado ao fazer as recomendações.

O fato de o cálculo do grau de afinidade entre um padrão e um termo considerar não apenas a quantidade de ocorrências do termo no padrão, mas também o contexto das ocorrências contribuiu para a robustez do fator de relacionamento entre um termo chave e um padrão e consequente alto grau de assertividade das recomendações. Contudo é preciso observar que as classificações geradas não possuem o nível máximo de acuidade uma vez que nem sempre o teor da seção onde ocorre um termo chave indica com eficácia a conotação que o termo assume no texto. Analisamos a ferramenta de extração de conhecimento em termos das medidas de precisão e recall parametrizadas por resultados conseguidos manualmente e os índices obtidos foram considerados satisfatórios.

Apresentou-se ainda o comportamento da abordagem sob a ótica do relacionamento entre atributos de qualidade. De fato, a análise destes relacionamentos evidenciou que a abordagem proposta, apesar dos eventuais falsos positivos, via de regra repetia o caráter agregador ou conflituoso já documentado na literatura entre os atributos de qualidade considerados. Em alguns dos casos onde percebeu-se que o comportamento dos atributos de qualidade não era compatível com o comportamento comumente documentado na literatura, análises aprofundadas foram realizadas e constatou-se que o padrão atuava como resolvedor de conflitos, conseguindo conciliar objetivos conflitantes, fato que agregou valor aos resultados alcançados.

6.1 Trabalhos Futuros

Ainda que durante o trabalho uma prova de conceito tenha sido criada no intuito de validar a abordagem, a mesma não possui maturidade suficiente para ser disponibilizada ao usuário final. Assim, a operacionalização da abordagem através da construção de uma ferramenta de recomendação de padrões para a arquitetura, contendo funcionalidades adicionais como a possibilidade de enriquecer a base de conhecimento com novos padrões e termos é uma necessidade que deve ser atendida no futuro.

Outro ponto que pode ser trabalhado futuramente é a matriz de mapeamento das ocorrências de termos chave em pesos apresentada na Tabela 4.1. Ainda que exista um raciocínio justificando as escolhas dos pesos, trata-se de um raciocínio com pouco embasamento científico. Pretende-se utilizar alguma técnica de colaboração que permita que a abordagem apresentada aprenda com os erros. Nesta linha, os pesos da matriz de mapeamento das ocorrências de termos chave seriam ajustados conforme feedback dos usuários da abordagem quanto à assertividade ou não das recomendações.

Ainda que vários requisitos não funcionais sejam considerados na elaboração de uma arquitetura, existe entre eles uma ordem de prioridade com influência direta na tomada de decisão. Isto não foi considerado na concepção da abordagem e é portanto um ponto a ser trabalhado futuramente.

As ocorrências de falsos positivos prejudicam bastante a precisão da procura e constituem também um ponto a ser melhorado através do aperfeiçoamento das técnicas utilizadas na procura de termos chave.

APÊNDICE A – PADRÕES PARA ARQUITETURA

Nome do Padrão	Fonte (listadas na Tabela 5.1)	Tipo
Abstract Factory	GAMMA	Padrão de Projeto
Acceptor-Connector	POSA 2	Padrão de Projeto
Account Lockout	SECURITY	Não definido
Active Object	POSA 2	Padrão de Projeto
Adapter	GAMMA	Padrão de Projeto
Asynchronous Completion Token	POSA 2	Padrão de Projeto
Authenticated Session	SECURITY	Não definido
Blackboard	POSA 1	Padrão Arquitetural
Bridge	GAMMA	Padrão de Projeto
Broker	POSA 1	Padrão Arquitetural
Builder	GAMMA	Padrão de Projeto
Chain of Responsibility	GAMMA	Padrão de Projeto
Client-Dispatcher-Server	POSA 1	Padrão de Projeto
Command	GAMMA	Padrão de Projeto
Command Processor	POSA 1	Padrão de Projeto
Component Configurator	POSA 2	Padrão de Projeto
Composite	GAMMA	Padrão de Projeto
Decorator	GAMMA	Padrão de Projeto
Double-Checked Locking Optimization	POSA 2	Padrão de Projeto
Extension Interface	POSA 2	Padrão de Projeto
Façade	GAMMA	Padrão de Projeto
Factory Method	GAMMA	Padrão de Projeto
Flyweight	GAMMA	Padrão de Projeto
Forwarder-Receiver	POSA 1	Padrão de Projeto
Half Sync Half Async	POSA 2	Padrão Arquitetural
Interceptor	POSA 2	Padrão Arquitetural
Interpreter	GAMMA	Padrão de Projeto
Iterator	GAMMA	Padrão de Projeto

Nome do Padrão	Fonte (listadas na Tabela 5.1)	Tipo
Layers	POSA 1	Padrão Arquitetural
Leader/Followers	POSA 2	Padrão Arquitetural
Leasing	POSA 3	Não definido
Lookup	POSA 3	Não definido
Master Slave	POSA 1	Padrão de Projeto
Mediator	GAMMA	Padrão de Projeto
Memento	GAMMA	Padrão de Projeto
Microkernel	POSA 1	Padrão Arquitetural
Model View Controller	POSA 1	Padrão Arquitetural
Monitor Object	POSA 2	Padrão de Projeto
Observer	GAMMA	Padrão de Projeto
Page Controller	MSDN	Não definido
Pipes and Filters	POSA 1	Padrão Arquitetural
Presentation-Abstraction-Control	POSA 1	Padrão Arquitetural
Proactor	POSA 2	Padrão Arquitetural
Prototype	GAMMA	Padrão de Projeto
Proxy	POSA 1	Padrão de Projeto
Publisher-Subscriber, Observer	POSA 2	Padrão de Projeto
Reactor	POSA 2	Padrão Arquitetural
Reflection	POSA 1	Padrão Arquitetural
Resource Lifecycle Manager	POSA 3	Não definido
Scoped Locking	POSA 2	Idiom
Singleton	GAMMA	Padrão de Projeto
State	GAMMA	Padrão de Projeto
Strategized Locking	POSA 2	Padrão de Projeto
Strategy	GAMMA	Padrão de Projeto
Template Method	GAMMA	Padrão de Projeto
Thread-Safe Interface	POSA 2	Padrão de Projeto
View Handler	POSA 1	Padrão de Projeto
Visitor	GAMMA	Padrão de Projeto
Whole Part	POSA 1	Padrão de Projeto
Wrapper Façade	POSA 2	Padrão de Projeto

APÊNDICE B – ATRIBUTOS DE QUALIDADE

Atributo de Qualidade	Termos equivalentes
Analyzability	
Availability	
Extensibility	
Maintainability	changeability, exchangeability, flexibility, modifiability
Performance	efficiency
Portability	instalability, co-existence, replaceability
Reliability	
Security	protection
Testability	
Usability	

APÊNDICE C – PARAMÊTRO DE ATRIBUTO DE QUALIDADE

PAQ	AQ	Termos equivalentes
data integrity	security	
adaptability	portability	
assurance	security	confidence, sureness
attack detection	security	notice trespass, detect encroachment, discover intrusion, discover violation, notice violation, notice attack, find usurpation, find invasion, detect usurpation, detect invasion, discover trespass, detect trespass, find encroachment, notice encroachment,...
attack recovering	security	intrusion recuperation, usurpation recuperation, encroachment recovering, usurpation recovering, trespass recovering, invasion recuperation, invasion recovering, trespass recuperation, encroachment recuperation, intrusion recovering, attack recuperation,...
attack resistance	security	encroachment immunity, trespass resistance, invasion resistance, attack immunity, invasion immunity, trespass immunity, intrusion immunity, violation immunity, intrusion resistance, violation resistance, encroachment resistance, usurpation immunity
auditing	security	
binding time	maintainability	time to deploy

PAQ	AQ	Termos equivalentes
cohesion	maintainability	
component complexity	analyzability, maintainability, testability	
component volatility	extensibility, maintainability	modifications cost
confidentiality	security	
coupling	extensibility, maintainability	
deadline	performance	time limit, due date, resource arbitration, time frame, window of time
failure rate	availability	mean time failure, fault prevention, failure pace, number of defects
fault detection	Availability	failure detection, mistake find accuracy, mistake detection, catching wrong components, catching failed components, defect discover accuracy, fault detect accuracy, defect find accuracy, error find accuracy, failure discover accuracy, defect detect accuracy.
fault tolerance	availability, reliability	
latency	performance	overhead, delay, time response, retardation, reaction time, slowdown, lag
learnability	usability	
maturity	reliability	
modificable user interface	usability	
multiple user interface	usability	multiple views
nonrepudiation	security	
recoverability	availability, reliability	mean time to recover, error handling
resource usage	performance	resource management, resource arbitration, resource utilization
system initiative	usability	
throughput	performance	output, rate processing

PAQ	AQ	Termos equivalentes
user error avoidance	usability	
user initiative	usability	undo command
user interaction	usability	

APÊNDICE D – TÁTICAS ARQUITETURAIS

Nome Tática	**PAQ relacionado**
abstract services	cohesion
aggregate command	user initiative
authenticate	assurance
authorize	data integrity
bound execution time	resource usage
communication path	coupling
configuration files	binding time
event rate	latency
indirection	component volatility, component volatility
intermediate	coupling
limit access	attack resistance
raise exceptions	fault detection
redundancy	attack recovering, confidentiality, nonrepudiation
restarting component	recoverability
resynchronization	recoverability
transactions	failure rate

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, C.; ISHIKAWA, S.; SILVERSTEIN, M.; JACOBSON, M.; FIKSDAHL-KING, I.; ANGEL, S. **A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure)**. Later printing. Oxford University Press, 1977.

BABAR, M. A. Scenarios, Quality Attributes, and Patterns: Capturing and Using Their Synergistic Relationships for Product Line Architectures. **11th Asia-Pacific Software Engineering Conference** p. 574–578, 2004.

BACHMANN, F.; BASS, L.; NORD, R. Modifiability Tactics (CMU/SEI-2007-TR-002). **Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University,** 2007.

BARBACCI, M. R.; ELLISON, R. J.; LATTANZE, A. J.; STAFFORD, J. A.; WEINSTOCK, C. B.; WOOD, W. G. **Quality Attribute Workshops QAWs - Third Edition,** 2003.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice, Second Edition**. Addison Wesley, 2003.

BENGTSSON, P.; LASSING, N.; BOSCH, J.; VAN VLIET, H. Architecture-Level Modifiability Analysis (ALMA). **Journal of Systems and Software** v. 69, n. 1-2, p. 129–147, 2004.

BIRUKOU, A. A Survey of Existing Approaches for Pattern Search and Selection. In: **Proceedings of the 15th European Conference on Pattern Languages of Programs**. EuroPLoP ’10. New York, NY, USA: ACM, 2010.

BIRUKOU, A.; WEISS, M. Service for Selecting Patterns. In: **Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP’09),** 2009.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M. Quantitative Evaluation of Software Quality. In: **Proceedings of the 2Nd International Conference on Software Engineering**. ICSE ’76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976.

BOEHM, B. W. **Software Engineering Economics**. 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **A System of Patterns: Pattern-Oriented Software Architecture**. **Network**. Wiley, 1996. 476p.

CHUNG, L.; COOPER, K.; YI, A. Developing Adaptable Software Architectures Using Design Patterns: An NFR Approach. **Computer Standards & Interfaces** v. 25, n. 3, p. 253–260, 2003.

CISQ. Software Structural Quality Characteristics. **Consortium for IT Software Quality - CMU/SEI,** 2011.

CLEMENTS, P.; SHAW, M. 25th-Anniversary Top Picks "The Golden Age of Software Architecture" Revisited. **IEEE Software** p. 70–72, 2009.

DROMEY, R. G. Cornering the Chimera. **IEEE Softw.** v. 13, n. January, p. 33–43, 1996.

FALESSI, D.; CANTONE, G.; KAZMAN, R.; KRUCHTEN, P. Decision-Making Techniques for Software Architecture Design. **ACM Computing Surveys** v. 43, n. 4, p. 1–28, 2011.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

GARLAND, J.; ANTHONY, R. **Large-Scale Software Architecture: A Practical Guide Using UML**. Ed. John Wiley & Sons2003.

GIESECKE, S.; HASSELBRING, W.; RIEBISCH, M. Classifying Architectural Constraints as a Basis for Software Quality Assesment. **Advanced Engineering Informatics** v. 21, p. 169–179, 2007.

GOMES, P.; PEREIRA, F. C.; PAIVA, P.; SECO, N.; CARREIRO, P.; FERREIRA, J. L.; BENTO, C. Using CBR for Automation of Software Design Patterns. In: **Proceedings of the 6th European Conference on Advances in Case-Based Reasoning**. ECCBR '02. London, UK, UK: Springer-Verlag, 2002.

GORTON, I. **Essential Software Architecture**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

GROSS, D.; YU, E. From Non-Functional Requirements to Design through Patterns. **Requirements Engineering** v. 6, p. 18–36, 2000.

GUÉHÉNEUC, Y.; MUSTAPHA, R. A Simple Recommender System for Design Patterns. In: **Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories** p. 1–2, 2007.

HARRISON, N. B.; AVGERIOU, P. Leveraging Architecture Patterns to Satisfy Quality Attributes. **Lecture Notes in Computer Science** v. 4758,. Lecture Notes in Computer Science p. 263–270, 2007.

HARTER, D. E.; KRISHNAN, M. S.; SLAUGHTER, S. A. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. **Management Science** v. 46, n. 4, p. 451–466, 2000.

HASHEMINEJAD, S. M. H.; JALILI, S. Design Patterns Selection: An Automatic Two-Phase Method. **Journal of Systems and Software** v. 85, n. 2, p. 408–424, 2012.

HENNINGER, S.; CORRÊA, V. Software Pattern Communities: Current Practices and Challenges. In: **Proceedings of the 14th Conference on Pattern Languages of Programs**. PLOP '07. ACM, 2007.

IEEE. IEEE Standard for a Software Quality Metrics Methodology. **IEEE Std 1061-1998,** 1998.

IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. **IEEE Std 1471-2000** n. IEEE-Std-1471-2000, p. i–23, 2000.

ISO/IEC. **ISO/IEC 9126. Software Engineering -- Product Quality**. ISO/IEC, 2001.

JIMENEZ, G.; TAJ, S.; WEAVER, J. Design For Testability. In: **Proceedings of the 9th Annual NCIIA Conference,** 2005.

KAZMAN, R.; ABOWD, G.; BASS, L.; CLEMENTS, P. Scenario-Based Analysis of Software Architecture. **IEEE Softw.** v. 13, n. 6, p. 47–55, 1996.

KAZMAN, R.; KLEIN, M.; CLEMENTS, P. ATAM : Method for Architecture Evaluation (CMU/SEI-2000-TR-004). **Software Engineering Institute, Carnegie Mellon University** n. August,2000.

KHOURY, P. EL; MOKHTARI, A.; COQUERY, E.; HACID, M.-S. An Ontological Interface for Software Developers to Select Security Patterns. In: **19th International Conference on Database and Expert Systems Applications**. IEEE, 2008.

KIENZLE, D. M.; ELDER, M. C.; D, P.; TYREE, D.; EDWARDS-HEWITT, J. Security Patterns Repository, Version 1.0, **http://www.scrypt.net/ celer/securitypatterns/repository.pdf** 2006.

KIM, S.; KIM, D.; LU, L.; PARK, S. The Journal of Systems and Software Quality-Driven Architecture Development Using Architectural Tactics. **The Journal of Systems & Software** v. 82, n. 8, p. 1211–1231, 2009.

KIRCHER, M.; JAIN, P. **Pattern-Oriented Software Architecture, Volume 3: Patterns for Resource Management**. Chichester, UK: Wiley, 2004.

KAZMAN, R.; OZKAYA, I.; KLEIN, M. Quality-Attribute Based Economic Valuation of Architectural Patterns. In: **First International Workshop on the Economics of Software and Computation, ESC'07**. 1st International Workshop on the Economics of Software and Computation, ESC'07. Software Engineering Institute, Carnegie Mellon University, .

KLEIN, M.; KAZMAN, R.; BASS, L.; CARRIERE, J.; BARBACCI, M.; LIPSON, H. Attribute-Based Architecture Styles. In: **Software Engineering Institute, Carnegie Mellon University,** 1999. 1–20p.

LIN, N. H. J. K. C. Object-Oriented Design : A Goal-Driven and Pattern-Based Approach. **Software and Systems Modeling** v. 8, n. 1, p. 67–84, 2009.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. **Factors in Software Quality. Volume I: Concepts and Definitions of Software Quality**. AD A049. General Electric, 1977.

MENASCE, D. A.; SOUSA, J. P.; MALEK, S.; GOMAA, H. Qos Architectural Patterns for Self-Architecting Software Systems. In: **Proceedings of the 7th International Conference on Autonomic Computing**. ICAC '10. New York, NY, USA: ACM, 2010.

MICROSOFT. MSDN: Patterns and Practices. **Microsoft,** 2014.

MUSSBACHER, G.; WEISS, M.; AMYOT, D. Formalizing Architectural Patterns with the Goal-Oriented Requirement Language. **Nordic Pattern Languages of Programs Conference** p. 1–24, 2006.

NATALE, G. DI; DOULCIER, M.; FLOTTES, M.-L.; ROUZEYRE, B. Self-Test Techniques for Crypto-Devices. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems** v. 18, n. 2, p. 329–333, 2010.

PALMA, F.; FARZIN, H. Recommendation System for Design Patterns in Software Development : An DPR Overview. In: **Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering,** 2012.

PEARSON, S.; SHEN, Y. Context-Aware Privacy Design Pattern Selection. In: **Proceedings of the 7th International Conference on Trust, Privacy and Security in Digital Business**. TrustBus'10. Berlin, Heidelberg: Springer-Verlag, 2010.

PORTER, M. F. An Algorithm for Suffix Stripping. **Program: Electronic Library & Information Systems** v. 40, n. 3, p. 211–218, 1980.

SAHLY, E. M.; SALLABI, O. M. Design Pattern Selection: A Solution Strategy Method. **2012 International Conference on Computer Systems and Industrial Informatics** p. 1–6, 2012.

SCHMIDT, D. C.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. **Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects**. 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2000.

SHAW, M. The Coming-of-Age of Software Architecture Research. In: **Proceedings of the 23rd International Conference on Software Engineering.** EEE Computer Society, 2001.

SHAW, M.; CLEMENTS, P. The Golden Age of Software Architecture. **IEEE Softw.** v. 23, n. 2, p. 31–39, 2006.

SOARES, L. S.; PRICE, R. T.; PIMENTA, M. S.; BRAGA, J. L. Selecting Architectural Patterns through a Knowledge-Based Approach. In: **Proceedings of IADIS Applied Computing 2011**. Rio de Janeiro, 2011.

SOARES, L. S.; PRICE, R. T.; PIMENTA, M. S.; LUIS, J. Seleção de Padrões Para a Arquitetura de Software : Abordagem Baseada Em Parâmetros de Atributo de Qualidade. In: **Proceedings of the 9th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2012)**. Natal, RN, 2012.

SOMMERVILLE, I. **Software Engineering (9th Edition)**. Addison Wesley; 9 edition, 2010. 792p.

KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The Past, Present, and Future for Software Architecture. **Software, IEEE** v. 23, n. 2, p. 22–30, 2006.

SURESH, S. S.; NAIDU, P. M. M.; KIRAN, S. A. (Methodology , Data Model and Algorithms)2011.

THIEL, S. A Framework to Improve the Architecture Quality of Software-Intensive Systems. Universität Duisburg-Essen, 2005.

WANG, J.; SONG, Y.-T.; CHUNG, L. From Software Architecture to Design Patterns: A Case Study of an NFR Approach. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on**2005.

WOJCIK, R.; BACHMMAN, F.; BASS, L.; CLEMENTS, P.; MERSON, P.; NORD, R.; WOOD, W. Attribute-Driven Design (ADD), Version 2 . 0. **(CMU/SEI-2006-TR-023)** n. November,2006.

WOLF, A. L.; LABORATORIES, T. B.; HILL, M. Foundations for the Study of Software Architecture. **Software Engineering Notes** v. 17, n. 4,1992.

XAVIER, J.; WERNER, C. M. L.; TRAVASSOS, G. H. Uma Abordagem Para a Seleção de Padrões Arquiteturais Baseada Em Características de Qualidade. In: **XVI Simpósio Brasileiro de Engenharia de Software (XVI SBES).** Gramado, RS, Brasil, 2002.

ZDUN, U. Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. **Softw. Pract. Exper.** v. 37, n. 9, p. 983–1016, 2007.