UNIVERSIDADE DE SÃO PAULO
FACULDADE DE ECONOMIA, ADMINISTRAÇÃO E CONTABILIDADE
DEPARTAMENTO DE ADMINISTRAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO

UM ESTUDO SOBRE A RELAÇÃO ENTRE QUALIDADE E ARQUITETURA DE SOFTWARE

Mauricio Tsuruta

Orientador: Prof. Dr. Hiroo Takaoka

São Paulo
2010

Prof. Dr. João Grandino Rodas
Reitor da Universidade de São Paulo

Prof. Dr. Reinaldo Guerreiro
Diretor da Faculdade de Economia, Administração e Contabilidade

Prof. Dr. Adalberto Américo Fischmann
Chefe do Departamento de Administração

Prof. Dr. Lindolfo Galvão de Albuquerque
Coordenador do Programa de Pós-Graduação em Administração

MAURICIO TSURUTA

UM ESTUDO SOBRE A RELAÇÃO ENTRE QUALIDADE E ARQUITETURA DE SOFTWARE

Dissertação apresentada ao Departamento de Administração da Faculdade de Economia, Administração e Contabilidade da Universidade de São Paulo como requisito para a obtenção do título de Mestre em Administração.

Orientador: Prof. Dr. Hiroo Takaoka

Versão corrigida

(versão original disponível na Unidade que aloja o Programa)

SÃO PAULO

2010

FICHA CATALOGRÁFICA
Elaborada pela Seção de Processamento Técnico do SBD/FEA/USP

Tsuruta, Mauricio
Um estudo sobre a relação entre qualidade e arquitetura de software / Mauricio Tsuruta. -- São Paulo, 2010.
131 p.

Dissertação (Mestrado) – Universidade de São Paulo, 2010.
Orientador: Hiroo Takaoka.

1. Qualidade de software 2. Arquitetura de software 3. Processo de software 4. Algoritmos genéticos I. Universidade de São Paulo. Faculdade de Economia, Administração e Contabilidade II. Título.

CDD – 005.12

Aos meus familiares.

Agradeço ao meu orientador Prof. Dr. Hiroo Takaoka por todo o apoio e orientação. Suas contribuições na "arquitetura do trabalho" representam suas contribuições para a "qualidade do trabalho".

Ao Prof. Dr. Antonio Geraldo da Rocha Vidal e ao Prof. Dr. Wilson Massashiro Yonezawa pelas valiosas contribuições feitas no exame de qualificação.

E à Lícia Mutsuko Abe por todo o suporte e ajuda durante a execução deste trabalho.

“Architecture, of all the arts,
is the one which acts the most slowly,
but the most surely, on the soul.”
Ernest Dimnet

RESUMO

Diversos setores da economia tem alto grau de dependência de sistemas computacionais: telecomunicação, financeiro, infraestrutura, industrial dentre outros. Desta forma, a qualidade do software contido nestes sistemas é um ítem importante para o bom desempenho destes setores. A arquitetura de software é considerada fator determinante para a qualidade de software. Este trabalho estuda a maneira pela qual a arquitetura de software determina a qualidade do software produzido e as possibilidades de se obter os atributos de qualidade desejados através da especificação de uma arquitetura de software apropriada. O método de pesquisa se fundamenta na revisão da literatura e quatro abordagens para a especificação da arquitetura de software são consideradas: clássica, orientada a objetos, orientada a atributos e orientada a busca. A abordagem orientada a busca é um campo de estudo relativamente recente e os avanços realizados são reportados dentro da área de conhecimento denominada de Search Based Software Engineering. Esta área de conhecimento utiliza técnicas meta-heurísticas para achar boas soluções para os problemas encontrados na Engenharia de Software. Uma das técnicas meta-heurísticas mais utilizadas, o algorítmo genético, é usada em uma aplicação cujo processo de design segue a abordagem orientada a busca.

ABSTRACT

Many sectors of economy depend highly on computing systems: telecommunication, finance, infrastructure, industrial, and others. Thus, the quality of software in these systems is an important item to achieve good performance in these sectors. The software architecture is considered one of the main factors that shape the software quality. This work studies the way software architecture determines the software quality and the possibilities to obtain the desired software quality attributes through specifying appropriate software architecture. The research method is based upon literature review and four approaches to software architecture design process are considered: classic, object oriented, attribute oriented and search oriented. The search oriented approach to software architecture design process is a relatively new field of study and advances are reported in a knowledge area called Search Based Software Engineering. This knowledge area uses metaheuristics techniques to find good solutions to problems found in software engineering. One of the metaheuristic technique most frequently used, the genetic algorithm, is used in an application that follows the search based approach.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABAS: Attribute-Based Architectural Styles
ACDM: Architecture Centric Design Method
ATAM: Architecture Tradeoff Analysis Method
B2B: Business-to-Business
B2C: Business-to-Consumer
BPM: Business Process Management
C2C: Consumer-to-Consumer
CBAM: Cost Benefit Analysis Method
CORBA: Common Object Request Broker Architecture
CRM: Customer Relationship Management
DSS: Decision Support System
EIS: Executive Information System
ERP: Enterprise Resource Planning
ESS: Executive Support System
IEEE: Institute of Electrical and Electronic Engineers
INCOSE: International Council on Systems Engineering
ISO: International Standards Organization
MDA: Model Driven Architecture
MVC: Model View Controller
OMG: Object Management Group
ORB: Object Request Broker
SaaS: Software as a Service
SE: Systems Engineering
SEI: Software Engineering Institute
SEO: Search Engine Optimization
SOA: Service Oriented Architecture
UML: Unified Modeling Language

LISTA DE QUADROS

LISTA DAS DEMAIS ILUSTRAÇÕES

1INTRODUÇÃO

Este trabalho apresenta um estudo sobre a relação entre qualidade e arquitetura de software. A importância do tema, o objetivo do estudo, o contexto do estudo e a estrutura do trabalho são apresentados neste capítulo.

1.1 Importância do tema

A maioria das empresas depende em maior ou menor grau de um sistema de software para auxiliar sua operação. Portanto, o sistema de software exerce influência sobre a operação das empresas. A utilização de um sistema de software de alta qualidade tende a melhorar a operação de uma empresa assim como a utilização de um sistema de software de baixa qualidade tende a piorar a operação da mesma. Conseqüentemente, a qualidade de um sistema de software, ou de forma resumida, a qualidade de software se torna uma questão importante para as empresas.

Qualidade de software usualmente está associada à idéia de um software que apresenta nenhum ou poucos defeitos. Está idéia refere-se a implementação correta dos chamados requerimentos funcionais. Neste trabalho a ênfase é dada para a importância dos requerimentos não-funcionais. Em função da dinâmica do mundo dos negócios, os sistemas de software precisam se adaptar. Essa capacidade de adaptação ou modificação é um exemplo de um requerimento não-funcional muito importante para a maioria dos softwares.

A qualidade de software é determinada pelo produto de software o qual é resultado de um processo denominado de processo de software. A literatura reporta que um produto intermediário deste processo, a arquitetura de software, é um fator significativo para a determinação da qualidade de software. Desta forma, a compreensão da relação entre este fator, arquitetura de software, e a qualidade de software é importante para obter um software de alta qualidade ao final do processo de software.

A ilustração 1 apresenta o conceito desenvolvido nos dois parágrafos anteriores.

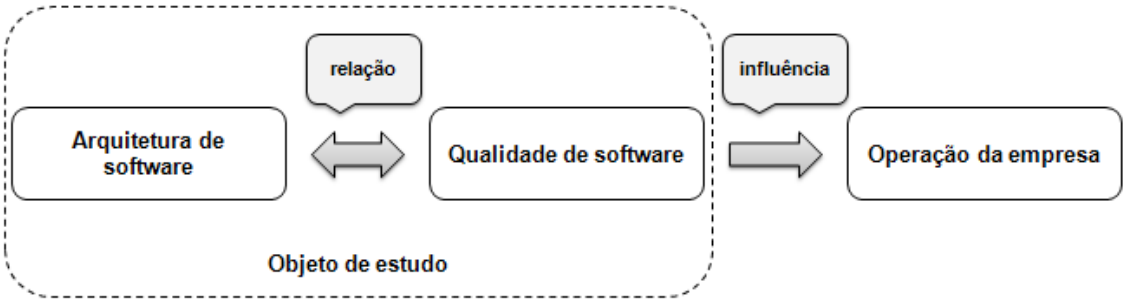

Ilustração 1 - Importância do tema.

Stair (2006) apresenta diversas aplicações de negócios de sistemas de informação. A ilustração 2 apresenta uma possível organização dos sistemas de informação baseada em algumas das categorias descritas por Stair (2006):

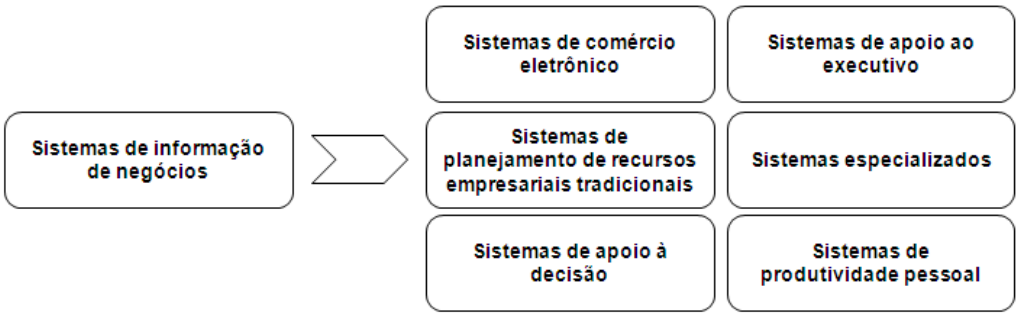

Ilustração 2 - Classificação de sistemas de informação de negócios.
Fonte: AUTOR, 2010, baseado em algumas das categorias descritas por Stair (2006).

Uma descrição de cada uma das seis categorias apresentadas na ilustração 2 é apresentada a seguir:

(1) Sistemas de comércio eletrônico

São sistemas que implementam os requerimentos necessários para se realizar comércio (transações) eletronicamente. As modalidades de negócio são usualmente classificadas de acordo com o tipo das partes envolvidas na transação:

- comércio eletrônico de negócio-para-negócio (B2B).
- comércio eletrônico de negócio-para-consumidor (B2C).
- comércio eletrônico de consumidor-para-consumidor (C2C).

(2) Sistemas de planejamento de recursos empresariais tradicionais (ERP)

São sistemas que implementam os requerimentos necessários para se realizar as atividades do dia-a-dia da empresa. Em função do tipo de atividade do dia-a-dia implementado, os sistemas de planejamento de recursos empresarias podem ser enquadrados em diversas categorias. Algumas destas categorias[1] são:

- Sistema de entrada de pedidos.
- Sistema de planejamento de entregas.
- Sistema de controle de estoque.
- Sistema de administração de relacionamento com clientes (CRM).
- Sistema de roteamento.
- Sistema de contas a receber.

(3) Sistemas de apoio à decisão (DSS)

São sistemas que auxiliam na resolução de problemas pouco estruturados. Os problemas pouco estruturados são caracterizados por apresentar dados difíceis de serem manipulados e/ou obtidos, e cujos relacionamentos não são claros e bem definidos. Stair (2006, p. 395) apresenta algumas das funções desempenhadas pelos sistemas de apoio à decisão:

- lidar com grandes volumes de dados, provenientes de fontes diversas.
- apresentar flexibilidade na elaboração e apresentação de relatórios.
- oferecer tanto orientação gráfica quanto textual.
- permitir análises detalhadas.
- fornecer apoio para a utilização de modelos de otimização e simulação.

(4) Sistemas de apoio ao executivo (ESS)

São sistemas voltados aos executivos de alto escalão que com freqüência precisam de auxílio especializado para tomar decisões estratégicas. São também conhecidos como sistemas de informação executiva (EIS). Algumas das características das decisões executivas auxiliadas por estes sistemas são apresentadas por Stair (2006, p. 407):

- apoio à definição de uma visão geral.
- apoio ao planejamento estratégico.
- apoio ao controle estratégico.

---

[1] Uma breve descrição de cada uma destas categorias é apresentada no glossário.

- auxílio na gestão de crises.

(5) Sistemas especializados

Stair (2006, p. 417-443) utiliza essa categoria para agrupar os sistemas de inteligência artificial, os sistemas de realidade virtual e outros sistemas especializados. Os sistemas de inteligência artificial por seu amplo campo de atuação são ainda subdivididos em sistemas de robótica, sistemas de visão, sistemas de processamento de linguagem natural, sistemas de aprendizado, sistemas de redes neurais e sistemas especialistas[2].

(6) Sistemas de produtividade pessoal

São sistemas que auxiliam as pessoas em suas atividades individuais seja no trabalho, na escola ou no lar. Fazem parte desta categoria os softwares de processamento de texto, planilha eletrônica, banco de dados, apresentação, gerenciamento de projetos e gerenciamento financeiro.

As seis categorias descritas permitem compreender a forte dependência que a maioria das empresas tem dos sistemas de software. Um fator que acentua esta dependência é o desenvolvimento da internet. A internet aumenta a importância dos sistemas de software para as empresas: (1) ao possibilitar a criação de novos modelos de negócios e (2) ao possibilitar a adição da característica de acesso a partir de qualquer dispositivo que possua um navegador aos sistemas existentes:

(1) Exemplos de novos modelos de negócios, muitos dos quais enquadrados na categoria conhecida como SaaS (*Software as a Service*), são: Google AdSense, Google AdWords, Amazon Elastic Compute Cloud (EC2), Wikipedia, Blogger, MySpace, YouTube, flickr, eBay, Google Apps[3], Google Maps, Microsoft Windows Live[4] e Microsoft Office Live Small Business.

---

[2] Uma breve descrição destes tipos de sistemas é apresentada no glossário.

[3] O Google Apps é uma coleção de serviços de software. Entre os principais serviços pode-se mencionar o Gmail, Google Talk, Google Calendar, Google Docs e Google Sites.

[4] O Microsoft Windows Live também é composto por um conjunto de serviços de software: Windows Live Messenger, Windows Live Hotmail, Windows Live Calendar, Windows Live Spaces e Windows Live ID.

(2) Exemplos de categorias de software que foram adaptadas ou reescritas para a internet são: sistemas de comércio eletrônico, sistemas de banco de dados, sistemas de gestão empresarial (ERP) e sistemas de gerenciamento de relacionamento com o cliente (CRM).

Esta primeira parte do capítulo 1 apresentou a importância do tema: a relação entre qualidade e arquitetura de software. Um atributo de qualidade, capacidade de modificação, foi ressaltado devido a sua importância para as empresas. Um efeito da internet é aumentar a importância deste atributo através da introdução de um ambiente que exige das empresas mudanças em períodos de tempo cada vez menores.

1.2 Objetivo do estudo

O objetivo deste trabalho é analisar a relação entre qualidade de software e arquitetura de software. Este objetivo principal foi decomposto em três objetivos específicos:

(1) Conceituar qualidade de software e pesquisar formas de mensuração.
(2) Conceituar arquitetura de software e pesquisar formas de representação.
(3) Analisar as formas pelas quais a arquitetura de software pode ser especificada dada a qualidade de software desejada.

A ilustração 3 apresenta o objetivo do estudo e sua decomposição em três objetivos específicos. Adicionalmente, a ilustração 1 referente à importância do tema foi incorporada para evidenciar os elementos do objeto de estudo, a relação entre eles e a relação do objeto de estudo com a operação da empresa.

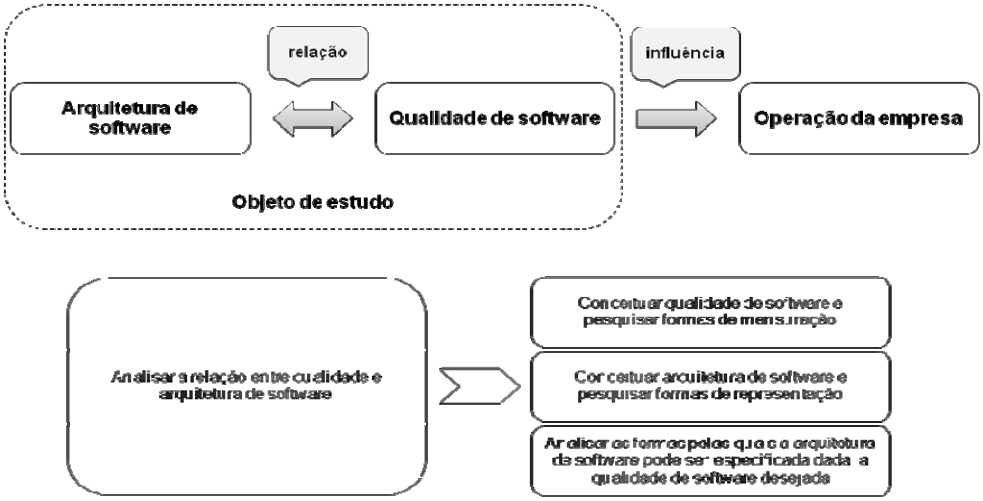

Ilustração 3 - Objetivo do estudo e sua decomposição em três objetivos específicos.

O primeiro objetivo específico consiste em conceituar qualidade de software e pesquisar formas de mensuração. Para alcançar este objetivo três etapas são percorridas: (1) conceituar o termo qualidade, (2) apresentar os modelos de qualidade mais estabelecidos e (3) apresentar formas de mensurar a qualidade de software (métricas de qualidade de software).

O segundo objetivo específico consiste em conceituar arquitetura de software e pesquisar formas de representação. Para atingir este objetivo três etapas são vencidas: (1) conceituar o

termo arquitetura de software, (2) apresentar formas de representar a arquitetura de software e (3) apresentar formas de se alterar a representação de uma arquitetura de software.

O terceiro objetivo específico consiste em analisar as formas pelas quais a arquitetura de software pode ser especificada dada uma qualidade de software desejada. O processo pelo qual uma arquitetura de software é especificada para apresentar a qualidade de software desejada é denominado de processo de design da arquitetura de software. Para alcançar este objetivo específico quatro formas, ou em outras palavras, quatro abordagens são analisadas: (1) a abordagem clássica, (2) a abordagem orientada a objetos, (3) a abordagem orientada a atributos e (4) a abordagem orientada a busca.
Ao término deste terceiro objetivo específico, o objetivo principal de analisar a relação entre qualidade de software e arquitetura de software é concluído.

A contribuição do estudo é a possibilidade de se melhorar a operação de uma empresa através de uma melhoria da qualidade dos sistemas de software que ela utiliza. Uma melhor compreensão da relação entre qualidade e arquitetura de software permite construir sistemas de software cuja arquitetura apresenta a qualidade desejada. A ilustração 4 apresenta a relação entre o objetivo do estudo e a contribuição do estudo. Adicionalmente, a ilustração 1 referente à importância do tema foi incorporada para contextualizar a contribuição do estudo.

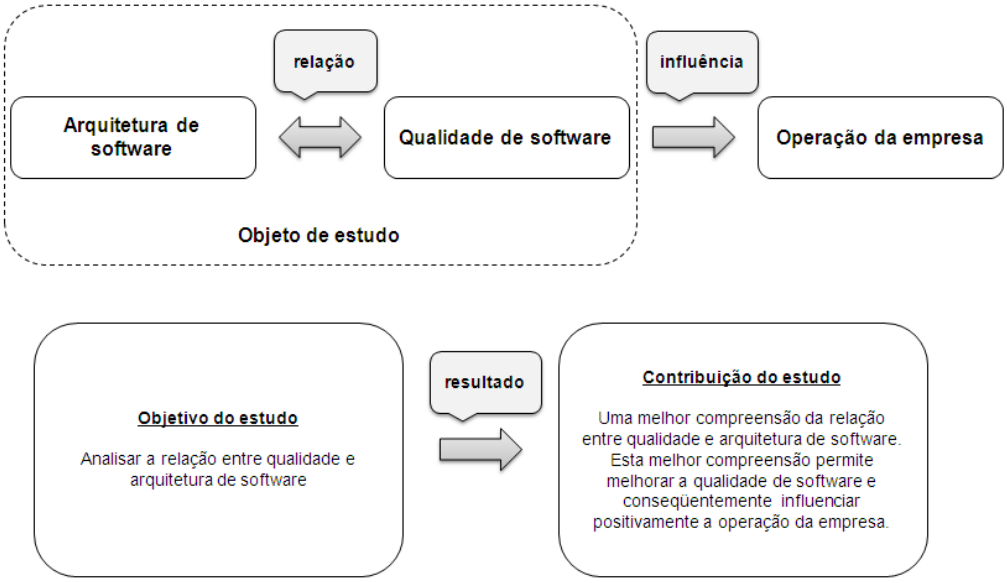

Ilustração 4 - Contribuição do estudo, objetivo do estudo e seu contexto.

Esta segunda parte do capítulo 1 apresentou o objetivo do trabalho e sua decomposição em três objetivos específicos. A contribuição do estudo foi contextualizada em termos do benefício que um melhor entendimento do objeto de estudo pode trazer para as operações das empresas.

1.3 Contexto do estudo

O contexto do estudo apresenta: (1) a perspectiva sob a qual a qualidade de software é considerada e (2) a posição que o tema arquitetura de software assume dentro de uma área de conhecimento.

(1) a perspectiva sob a qual a qualidade de software é considerada.

A qualidade de software pode ser influenciada por diversos fatores. Desta forma, a análise da influência de um fator sobre a qualidade de software pode ser vista como uma análise parcial. Cada uma das possíveis análises parciais assume, portanto, um ponto de vista, ou em outras palavras, uma perspectiva da qualidade de software.

Jalote (2008, p.9) considera que os três principais fatores que influenciam a qualidade e produtividade de software são: pessoas, processos e tecnologia. A produtividade de software é também um tema importante pois aborda a questão de como produzir software com uma determinada qualidade a um custo e/ou prazo menor.

Neste trabalho, a perspectiva sob a qual a qualidade de software é considerada é a perspectiva de processos conforme apresenta o diagrama da ilustração 5.

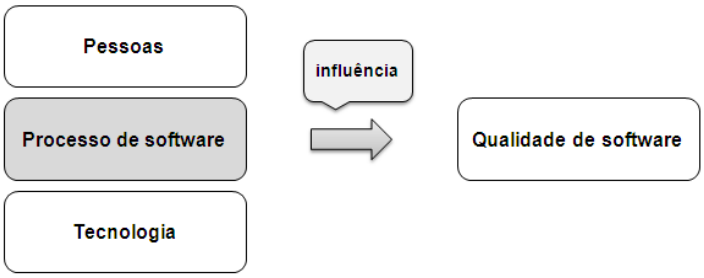

Ilustração 5 - Perspectiva sob a qual a qualidade de software é considerada.
Fonte: AUTOR, 2010, baseado nas considerações de Jalote (2008, p. 9).

(2) a posição que o tema arquitetura de software assume dentro de uma área de conhecimento.

Adotando-se a engenharia de software como uma área de conhecimento, a posição que o tema arquitetura de software assume dentro desta área de conhecimento á apresentado a seguir.

O software pode ser visto como um produto e, portanto, pode ser visto como o resultado de um processo. O processo que tem como resultado o produto software é denominado de processo de software.

Jalote (2008, p. 13) decompõe o processo de software em dois processos: (a) o processo de gerenciamento de processo de software e (b) o processo de engenharia de produto de software.

(a) O processo de gerenciamento de processo de software é responsável por gerenciar e otimizar o próprio processo de software e pode, portanto, ser visto como um metaprocesso.

(b) O processo de engenharia de produto de software é responsável pela construção do produto e pode ser decomposto em três processos: (b1) o processo de gerenciamento de configuração de software, (b2) o processo de gerenciamento de projeto de software e (b3) o processo de desenvolvimento de software.

(b1) O processo de gerenciamento de configuração de software é responsável por gerenciar as alterações de software ao longo de seu desenvolvimento. Um exemplo de alteração de software é a alteração do código-fonte. Para este exemplo, usualmente uma ferramenta de versionamento é utilizada no processo.

(b2) O processo de gerenciamento de projeto de software é responsável por gerenciar o projeto de software. A construção do produto software pode ser visto como um projeto e técnicas clássicas de gerenciamento de projetos podem ser utilizadas para se gerenciar os aspectos de qualidade, custo e prazo deste projeto.

(b3) O processo de desenvolvimento de software é responsável pela construção do produto software propriamente e pode ser considerado o núcleo do processo de engenharia de produto.

O tema arquitetura de software é tratado e, portanto posicionado, dentro do processo de desenvolvimento de software. A ilustração 6 apresenta a relação entre os processos descritos e destacando o caminho percorrido até se chegar ao processo de desenvolvimento de software.

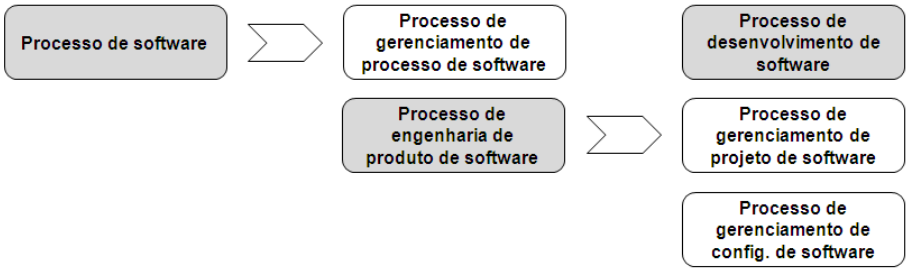

Ilustração 6 - Decomposição do processo de software
Fonte: Adaptado de JALOTE (2008, p. 13).

O processo de desenvolvimento de software pode ser visto como um conjunto de atividades. A forma como o processo de desenvolvimento de software é decomposto em atividades e o que compreende cada uma das atividades varia um pouco de autor para autor mas a estrutura básica é similar.

Por exemplo, Sommerville (2006, p. 64) enumera as seguintes atividades: especificação, design, implementação, validação e evolução. E Casteleyn (2009, p. 58) decompõe o processo nas seguintes atividades: requisitos, design, implementação, teste, implantação, manutenção e evolução. Casteleyn conceitua estas sete atividades que compõem o processo de desenvolvimento de software da seguinte forma:

(1) Requisitos: a atividade de requisitos tem por objetivo compreender o problema e descrevê-los. Considera tanto os requisitos funcionais quanto os requisitos não-funcionais.

(2) Design: a atividade de design tem por objetivo planejar a especificação de uma solução. Esta especificação apresenta o design do software a partir de diferentes visões.

(3) Implementação: a atividade de implementação tem por objetivo criar o código, o banco de dados e os arquivos de configuração a partir da especificação da solução.

(4) Teste: a atividade de teste tem por objetivo verificar se os requisitos funcionais e requisitos não-funcionais foram devidamente implementados.

(5) Implantação: a atividade de implantação tem por objetivo disponibilizar o sistema de software aos usuários finais.

(6) Manutenção: a atividade de manutenção tem por objetivo monitorar e assegurar o funcionamento correto e contínuo do sistema de software.

(7) Evolução: a atividade de evolução tem por objetivo aperfeiçoar e adicionar novos requisitos ao sistema de software.

A ilustração 7 apresenta a decomposição do processo de desenvolvimento de software em sete atividades. A atividade de design está destacada para informar que o tema arquitetura de software é abordado nesta atividade.

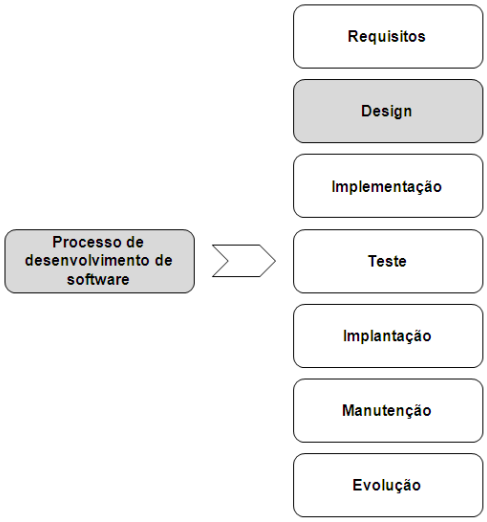

Ilustração 7 - Decomposição do processo de desenvolvimento de software.
Fonte: Adaptado de CASTELEYN (2009, p. 58).

Esta terceira parte do capítulo 1 apresentou a perspectiva de processos como a principal perspectiva considerada para se analisar a relação entre arquitetura e qualidade de software, e identificou a atividade de design do processo de desenvolvimento de software como a atividade a ser analisada pelas quatro abordagens apresentadas no capítulo 5.

1.4 Estrutura do trabalho

Este trabalho está organizado da seguinte forma:

Capítulo 1 – Introdução: apresenta a importância do tema, o objetivo do estudo, o contexto do estudo e a estrutura do trabalho.

Capítulo 2 – Metodologia: apresenta a metodologia do trabalho e a correspondente classificação metodológica.

Capítulo 3 – Qualidade de software: apresenta o conceito de qualidade, os modelos de qualidade e as métricas de qualidade.

Capítulo 4 – Arquitetura de software: apresenta o conceito de arquitetura de software, a representação da arquitetura de software e as táticas de arquitetura de software.

Capítulo 5 – Processo de design da arquitetura de software: apresenta o processo de design da arquitetura de software através de quatro abordagens: abordagem clássica, abordagem orientada a objetos, abordagem orientada a atributos e abordagem orientada a busca.

Capítulo 6 – Aplicação do processo de design da arquitetura de software: apresenta uma aplicação do processo de design da arquitetura de software segundo a abordagem orientada a busca.

Capítulo 7 – Considerações finais: apresenta as considerações finais, uma síntese da contribuição do estudo e sugestões para estudos futuros.

A ilustração 8 apresenta um diagrama que apresenta a relação entre os capítulos centrais do trabalho e os três objetivos específicos. O objetivo da ilustração é destacar de forma visual como o objetivo do estudo está distribuído e é alcançado ao longo da estrutura do trabalho.

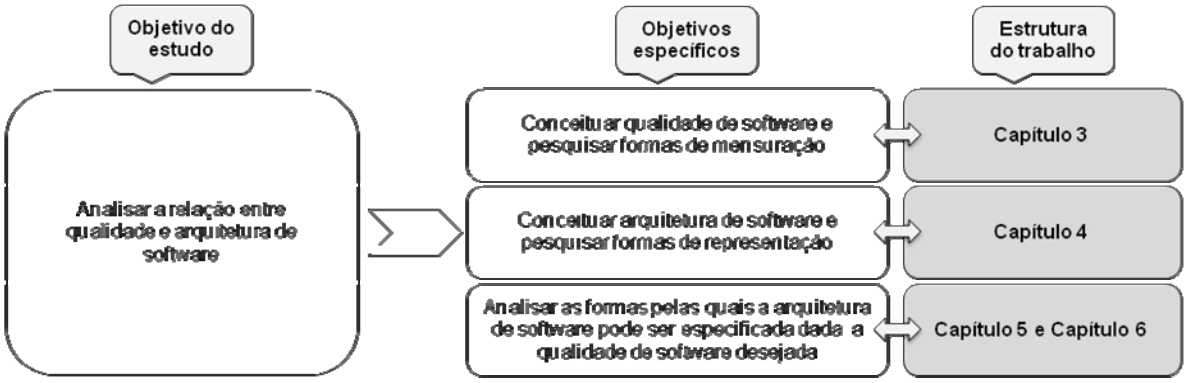

Ilustração 8 - Relação entre a estrutura do trabalho e o objetivo do estudo.

Esta quarta parte do capítulo 1 apresentou a estrutura do trabalho e a forma como ela se relaciona com o objetivo e com os três objetivos específicos do trabalho.

2 METODOLOGIA

Este capítulo de metodologia apresenta:

(1) O método utilizado: tem o objetivo de apresentar o caminho percorrido para se chegar ao objetivo do estudo. As etapas do caminho são representadas por questões fundamentais. Ao todo sete questões fundamentais são formuladas. A localização na estrutura do trabalho onde cada uma das sete questões fundamentais é abordada é indicada.

(2) A classificação metodológica: tem o objetivo de posicionar o estudo dentro de um conjunto de classificações metodológicas usualmente aceito no meio acadêmico e conseqüentemente permitir uma rápida compreensão a respeito da natureza, do objetivo, da abordagem e do procedimento do trabalho.

2.1 O método utilizado

O objetivo de analisar a relação entre qualidade e arquitetura de software é alcançado percorrendo sete etapas (sete questões fundamentais):

(1) O que é qualidade?
(2) O que é qualidade de software?
(3) Como mensurar qualidade de software?
(4) O que é arquitetura de software?
(5) Como representar a arquitetura de software?
(6) Como variar a representação da arquitetura de software?
(7) Como especificar uma arquitetura de software que apresente uma qualidade de software desejada?

Estas sete etapas são vencidas no decorrer dos capítulos 3, 4, 5 e 6. A ilustração 9 apresenta um diagrama que apresenta a correspondência entre os quatro capítulos e as sete etapas.

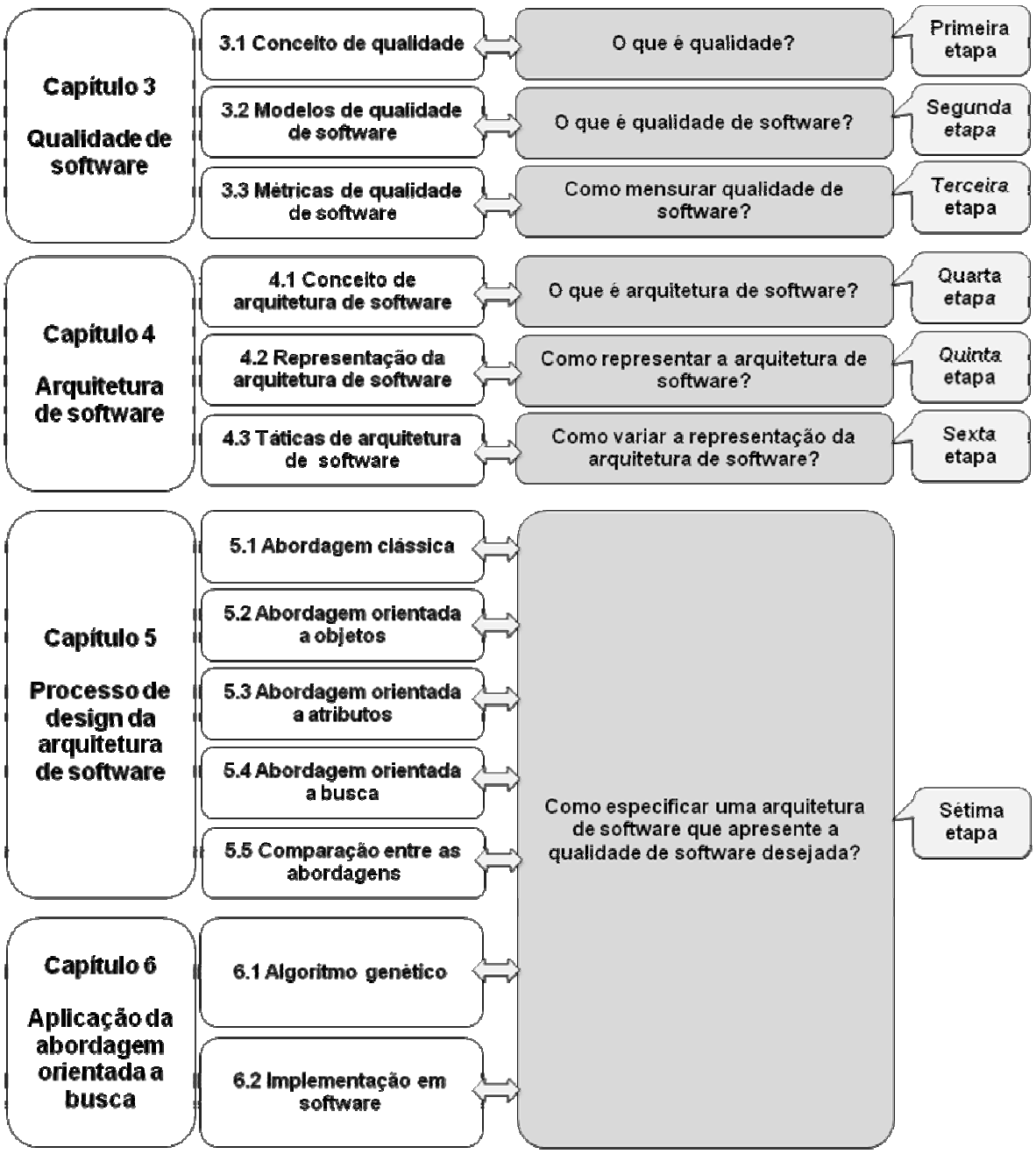

Ilustração 9 - Relação entre a estrutura do trabalho e o método utilizado.

Esta primeira parte do capítulo 2 apresentou o método utilizado. O caminho para se alcançar o objetivo do estudo foi dividido em sete questões ou etapas. Esta parte se encerra apresentando o local da estrutura do trabalho onde estas sete questões ou etapas são abordadas.

2.2 Classificação metodológica

Um estudo pode ser enquadrado em diversas classificações metodológicas. Cada classificação metodológica observa o estudo segundo uma perspectiva. Este estudo será classificado observando-se quatro perspectivas: (1) natureza, (2) objetivo, (3) abordagem e (4) procedimento.

(1) Natureza

Quanto à natureza a pesquisa pode ser classificada como pesquisa básica (ou teórica) ou pesquisa aplicada. A pesquisa básica enfatiza mais a construção e o aprimoramento de teorias e menos um objetivo comercial. A pesquisa aplicada enfatiza a resolução de um problema prático de natureza comercial. Os tópicos seguintes descrevem a visão de alguns autores a respeito dos conceitos de pesquisa básica e aplicada.

(1.1) Pesquisa básica

- "[...] objetiva gerar conhecimentos novos úteis para o avanço da ciência sem aplicação prática prevista. Envolve verdades e interesses universais." (SILVA, 2001, p.20).
- "O conhecimento teórico adequado acarreta rigor conceitual, análise acurada, desempenho lógico, argumentação diversificada, capacidade explicativa." (DEMO, 1994, p. 36).
- "[...] dedicada a reconstruir teoria, conceitos, idéias, ideologias, polêmicas, tendo em vista, em termos imediatos, aprimorar fundamentos teóricos." (DEMO, 2000, p. 20).
- "[...] compreende projetos de pesquisa que representam uma investigação original, com vistas ao avanço do conhecimento científico, e que não têm objetivos comerciais específicos." National Science Foundation.

(1.2) Pesquisa aplicada

- "[...] objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos. Envolve verdades e interesses locais." (SILVA, 2001, p.20).

Este estudo é enquadrado em pesquisa aplicada pois enfatiza a resolução de um problema prático de natureza comercial.

(2) Objetivo

Quanto ao objetivo a pesquisa pode ser classificada como pesquisa exploratória, pesquisa descritiva ou pesquisa explicativa. Esta classificação tem relação com o grau de conhecimento existente da área de conhecimento. A pesquisa exploratória enfatiza a descoberta de novas informações que contribuam para o desenvolvimento do tema. A pesquisa descritiva enfatiza a descrição do tema ou do fenômeno em estudo. A pesquisa explicativa pressupõe um grau de conhecimento maior sobre o tema e enfatiza a explicação do tema ou do fenômeno em estudo. Os tópicos seguintes descrevem a visão de alguns autores a respeito dos conceitos de pesquisa exploratória, descritiva e explicativa.

(2.1) Pesquisa exploratória

- “[...] visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. Envolve levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; análise de exemplos que estimulem a compreensão. Assume, em geral, as formas de Pesquisas Bibliográficas e Estudos de Caso.” (GIL, 1991).
- "[...] é o primeiro passo de qualquer trabalho científico. É também denominada pesquisa bibliográfica. Proporciona maiores informações sobre o tema que o pesquisador pretende abordar; auxilia-o a delimitá-lo; ajuda-o a definir seus objetivos e a formular suas hipóteses de trabalho e também a descobrir uma forma original de desenvolver seu assunto." (CIRIBELLI, 2003).

(2.2) Pesquisa descritiva

- “[...] visa descrever as características de determinada população ou fenômeno ou o estabelecimento de relações entre variáveis. Envolve o uso de técnicas padronizadas de coleta de dados: questionário e observação sistemática. Assume, em geral, a forma de Levantamento.” (GIL, 1991).

- "[...] os fatos são observados, registrados, analisados, classificados e interpretados sem que o pesquisador interfira neles.” (CIRIBELLI, 2003).

(2.3) Pesquisa explicativa

- “[...] visa identificar os fatores que determinam ou contribuem para a ocorrência dos fenômenos. Aprofunda o conhecimento da realidade porque explica a razão, o “porquê” das coisas. Quando realizada nas ciências naturais, requer o uso do método experimental, e nas ciências sociais requer o uso do método observacional. Assume, em geral, a formas de Pesquisa Experimental e Pesquisa Expost-facto.” (GIL, 1991).
- "[...] também denominada experimental, tem por objetivo não só registrar, analisar e interpretar os fenômenos estudados, mas procura mostrar por que eles ocorrem e os fatores que o determinam. Procura aprofundar o conhecimento da realidade, procurando a razão, o porquê das coisas." (CIRIBELLI, 2003).

Este estudo é enquadrado em pesquisa exploratória pois enfatiza a descoberta de novas informações que contribuam para o desenvolvimento do tema.

(3) Abordagem

Quanto à abordagem a pesquisa pode ser classificada como pesquisa qualitativa ou pesquisa quantitativa. A pesquisa qualitativa enfatiza mais a análise qualitativa (ou subjetiva) dos dados coletados e/ou informações obtidas. A pesquisa quantitativa enfatiza mais a análise quantitativa (ou matemática e/ou estatística) dos dados coletados e/ou informações obtidas. Os tópicos seguintes descrevem a visão de alguns autores a respeito dos conceitos de pesquisa qualitativa e quantitativa.

(3.1) Pesquisa qualitativa

- “[...] considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas

estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento-chave. É descritiva. Os pesquisadores tendem a analisar seus dados indutivamente. O processo e seu significado são os focos principais de abordagem.” (SILVA, 2001, p.20).

- "[...] se dá quando os dados só fazem sentido através de um tratamento lógico secundário, feito pelo próprio pesquisador." (CIRIBELLI, 2003).

(3.2) Pesquisa quantitativa

- “[...] considera que tudo pode ser quantificável, o que significa traduzir em números opiniões e informações para classificá-las e analisá-las. Requer o uso de recursos e de técnicas estatísticas (percentagem, média, moda, mediana, desvio-padrão, coeficiente de correlação, análise de regressão, etc.).” (SILVA, 2001, p.20).
- “[...] o importante é a coleta e a análise quantitativa dos dados, pois, por sua quantificação, aparecem os resultados automaticamente.” (CIRIBELLI, 2003).

Este estudo é enquadrado em pesquisa qualitativa pois enfatiza a análise qualitativa (ou subjetiva) das informações obtidas.

(4) Procedimento

Quanto ao procedimento a pesquisa pode ser classificada como pesquisa experimental, pesquisa de campo ou pesquisa bibliográfica. Esta classificação tem relação com a forma e com o grau de controle com que os dados são coletados e/ou as informações são obtidas. A pesquisa experimental enfatiza a coleta de dados em um ambiente experimental (ou laboratorial) no sentido que se pode exercer certo controle sobre os fatores que influenciam o fenômeno em estudo. A pesquisa de campo enfatiza a obtenção das informações diretamente pelo pesquisador no local de ocorrência do fenômeno. A pesquisa bibliográfica enfatiza a obtenção das informações através de material publicado. Os tópicos seguintes descrevem a visão de alguns autores a respeito dos conceitos de pesquisa experimental, de campo e bibliográfica.

(4.1) Pesquisa experimental

- “[...] quando se determina um objeto de estudo, selecionam-se as variáveis que seriam capazes de influenciá-lo, definem-se as formas de controle e de observação dos efeitos que a variável produz no objeto.” (GIL, 1991).
- “[...] dedicada ao tratamento da face empírica e fatual da realidade; produz e analisa dados, procedendo sempre pela via do controle empírico e fatual." (DEMO, 2000, p. 21).
- “[...] possibilidade que oferece de maior concretude às argumentações, por mais tênue que possa ser a base fatual. O significado dos dados empíricos depende do referencial teórico, mas estes dados agregam impacto pertinente, sobretudo no sentido de facilitarem a aproximação prática" (DEMO, 1994, p. 37).

(4.2) Pesquisa de campo

- "[...] baseia-se na observação dos fatos como eles ocorrem na realidade e os dados que coleta, que podem ser obtidos de diferentes formas, através de entrevistas, questionários, consultas, depoimentos e registros de ocorrências de determinados fenômenos. Neste tipo de pesquisa o pesquisador efetua a coleta dos dados "em campo", isto é, diretamente no local em que ocorrem os fatos ou fenômenos através da observação direta, do levantamento, etc." (CIRIBELLI, 2003).

(4.3) Pesquisa bibliográfica

- “[...] quando elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e atualmente com material disponibilizado na Internet.” (GIL, 1991).

Este estudo é enquadrado em pesquisa bibliográfica pois enfatiza a obtenção das informações através de material publicado.

A ilustração 10 apresenta um diagrama que apresenta a classificação metodológica do estudo sob cada uma das quatro perspectivas consideradas.

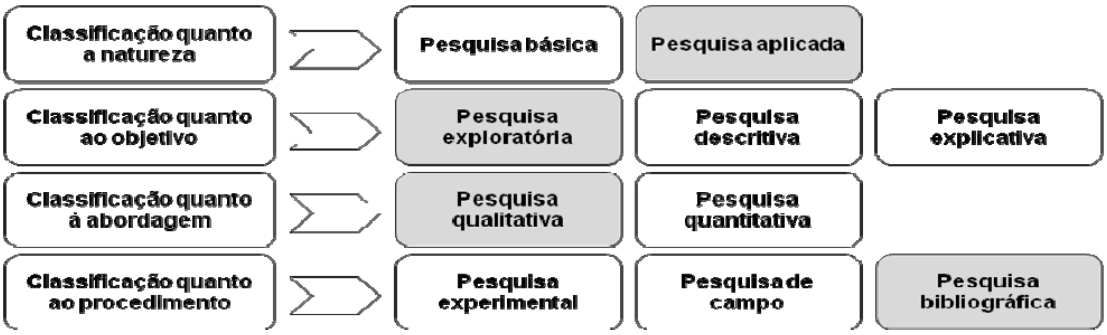

Ilustração 10 - Classificação metodológica do estudo.

Esta segunda parte do capítulo 2 apresentou a classificação metodológica do estudo. O estudo foi classificado como uma pesquisa aplicada, exploratória, qualitativa e bibliográfica conforme os conceitos apresentados pelos autores utilizados.

3 QUALIDADE DE SOFTWARE

Este capítulo apresenta o desenvolvimento das etapas 1, 2 e 3 que concluem o primeiro objetivo específico de "Conceituar qualidade de software e pesquisar formas de mensuração". A ilustração 11 apresenta uma síntese do capítulo.

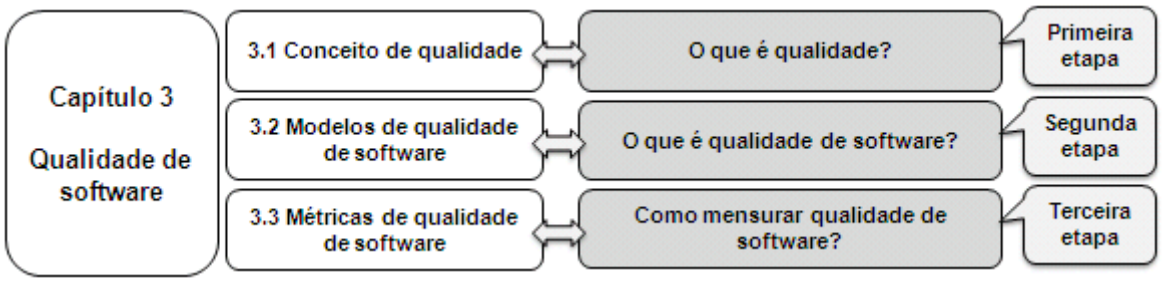

Ilustração 11 - Síntese do capítulo 3.

Para avaliarmos a qualidade de software é necessário considerarmos o modelo de qualidade, as dimensões da qualidade e as métricas de software utilizadas. Khosravi (2008, p. 1) reforça esta idéia colocando-a nos seguintes termos:

> "Métricas de software e modelos de qualidade tem uma função essencial na mensuração da qualidade de software. Diversos modelos de qualidade e métricas de software são utilizados para se construir software de qualidade tanto na indústria quanto na academia."[5]

3.1 Conceito de qualidade

Para conceituar qualidade, as idéias de cinco autores a respeito deste conceito são apresentadas. Os cinco autores selecionados são: Crosby, Deming, Feigenbaum, Ishikawa e Shewart.

---

[5] Software metrics and quality models play a pivotal role in measurement of software quality. A number of well-known quality models and software metrics are used to build quality software both in industry and in academia.

O termo qualidade é geralmente associado a um valor relativo em frases que utilizam expressões como “boa qualidade”, “má qualidade” e “qualidade de vida”. Essas expressões significam coisas diferentes para cada pessoa. Para ter algum controle sobre a qualidade é necessário definir qualidade como “conformidade a especificações”. Desta forma, a não-conformidade a especificações é a ausência de qualidade e qualidade se torna definível. (CROSBY, 1979).

A dificuldade em definir o que é qualidade de um produto está em traduzir as necessidades futuras de um usuário em características mensuráveis de forma que o produto possa ser projetado e satisfazer o usuário a um preço que ele possa pagar. Mas as mudanças dos consumidores, as entradas de competidores, a existência de novos materiais, etc, tornam este objetivo ainda mais difícil. (DEMING, 2000, p. 168 e 169).

A qualidade é determinada pelo consumidor e não pelo engenheiro ou pelo marketing ou pela administração. Ela é baseada na experiência atual do cliente com o produto ou serviço, e medida com base em seus requerimentos explicitados ou não, conscientes ou não, tecnicamente operacional ou totalmente subjetivo. A qualidade de um produto ou serviço pode ser definida como as características do produto que atendem a expectativa do cliente quando utilizado. (FEIGENBAUM, 1991).

O controle de qualidade procura produzir produtos com qualidade que possam satisfazer os requerimentos do consumidor. Mas não é suficiente que os produtos atendam a padrões e especificações de qualidade como o padrão ISO ou o padrão IEC. Os padrões não são perfeitos e não podem acompanhar as mudanças dos requerimentos do consumidor de ano para ano. Portanto, qualidade interpretada de uma forma restrita significa qualidade do produto e interpretada de uma forma ampla significa qualidade do produto, serviço, informação, pessoas, processos, sistemas, etc. (ISHIKAWA, 1985, p. 44 e 45).

Existem dois aspectos comuns da qualidade. Um dos aspectos tem relação com a qualidade de uma coisa como uma realidade objetiva independente da existência do homem. O outro aspecto tem relação com o que nós pensamos, sentimos ou sentimos como resultado desta

realidade objetiva. Em outras palavras, há um lado subjetivo da qualidade.[6] (SHEWART, 1980, p. 53).

É possível observar baseado nas considerações dos cinco autores a respeito do conceito de qualidade que:

- o conceito de qualidade usado coloquialmente é pouco preciso.
- qualidade tem dois aspectos: a qualidade do produto em si e a qualidade do produto percebida pelo usuário.
- para gerenciar ou controlar a qualidade é necessário conseguir mensurá-la.
- a qualidade exigida de produtos e serviços muda rapidamente devido a mudanças dos consumidores e do ambiente de negócios.

Esta primeira parte do capítulo 3 apresentou o conceito de qualidade sob o ponto de vista de cinco reconhecidos autores sobre este tema e destacou os pontos mais relevantes deste conceito para o trabalho.

---

[6] Enough has been said to indicate that there are two common aspects of quality. One of these has to do with the consideration of the quality of a thing as an objective reality independent of the existence of man. The other has to do with what we think, feel or sense as a result of the objective reality. In other word, there is a subjective side of quality.

3.2 Modelos de qualidade de software

Com uma compreensão melhor do conceito de qualidade, o próximo passo se torna entender melhor o conceito de qualidade aplicada a software. Esforços têm sido feitos para se representar qualidade de software através dos chamados modelos de qualidade de software.

Três modelos de qualidade de software são apresentados para auxiliar a refinar este conceito. Os três modelos considerados são: (1) o modelo de qualidade de McCall, (2) o modelo de qualidade de Boehm e (3) o modelo de qualidade ISO 9126.

(1) o modelo de qualidade de McCall

A estrutura do modelo de qualidade de McCall consiste de três perspectivas maiores (tipos de características de qualidade) para definir e identificar a qualidade de software. Cada uma destas três perspectivas maiores consiste de um conjunto de fatores de qualidade. Cada um destes fatores de qualidade possui um conjunto de critérios de qualidade e finalmente cada critério de qualidade pode ser mensurada por uma ou mais métricas. (MILICIC, 2005).

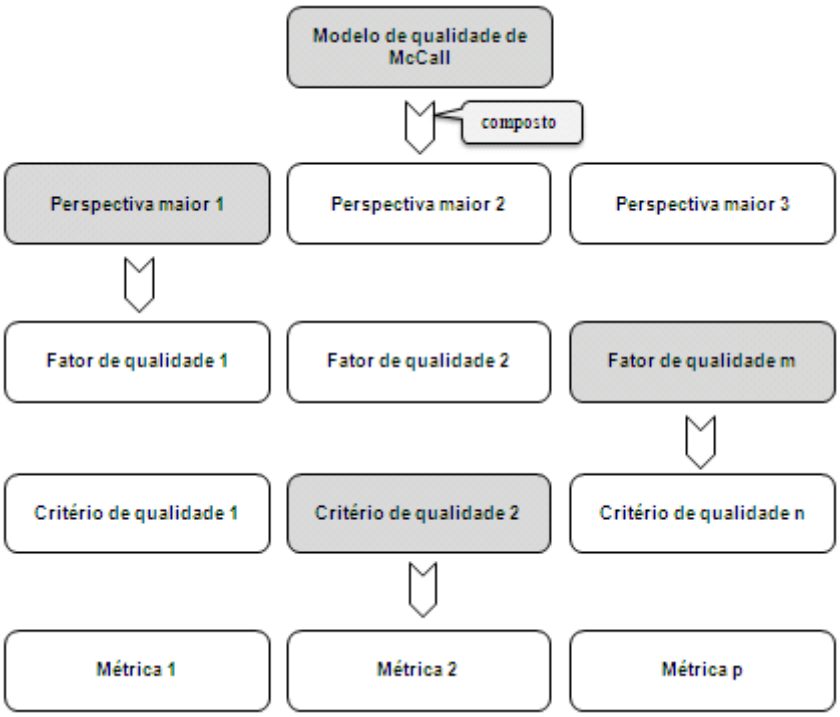

Ilustração 12 - Modelo de qualidade de McCall.
Fonte: Adaptado de Milicic (2005, p. 6).

Os conjuntos de fatores de qualidade que compõem as três perspectivas maiores são:

(1.1) Perspectiva maior 1 – Revisão do produto

Esta perspectiva está relacionada a habilidade do produto de sofrer alterações. Pode ser decomposta em três fatores de qualidade:

a) manutenibilidade: o esforço requerido para localizar e corrigir uma falha em um programa em seu ambiente de operação.

b) flexibilidade: a facilidade de fazer mudanças requeridas devido a alterações no ambiente de operação.

c) testabilidade: a facilidade de testar um programa para garantir que não contenha erros e que atenda as suas especificações.

(1.2) Perspectiva maior 2 – Operação do produto

Esta perspectiva está relacionada a características da operação do produto. Pode ser decomposta em cinco fatores de qualidade:

a) correção: a extensão com que o programa atende a sua especificação.

b) confiabilidade: a habilidade do sistema em não falhar.

c) eficiência: pode ser classificado em eficiência de execução e eficiência de armazenamento.

d) integridade: a proteção do programa de acessos não autorizados.

e) usabilidade: a facilidade de usar o software.

(1.3) Perspectiva maior 3 – Transição do produto

Esta perspectiva está relacionada à capacidade de adaptação do produto a novos ambientes. Pode ser decomposta em três fatores de qualidade:

a) portabilidade: o esforço requerido para transferir um programa de um ambiente para outro.

b) reusabilidade: a facilidade para reutilizar o software em um contexto diferente.

c) interoperabilidade: o esforço requerido para integrar um sistema com outro sistema.

Ao todo, o modelo de qualidade de McCall consiste de 11 fatores de qualidade para descrever a visão externa do software (ponto de vista do usuário), 23 critérios de qualidade para descrever a visão interna do software (ponto de vista do desenvolvedor) e um conjunto de métricas que são definidas para permitir a mensuração.

A ilustração 13 apresenta a decomposição da perspectiva maior Revisão do produto em fatores de qualidade que por sua vez são decompostos em critérios de qualidade.

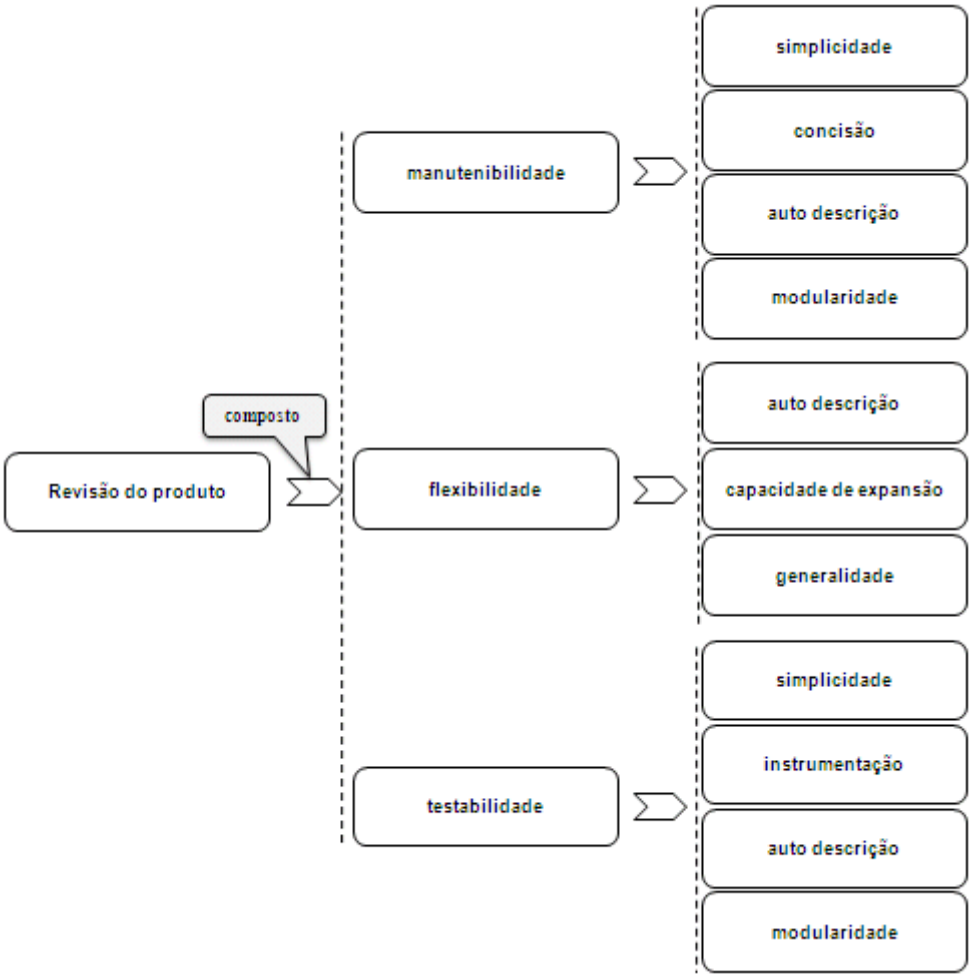

Ilustração 13 - Modelo de qualidade de McCall - decomposição.
Fonte: Adaptado de Al-Qutaish (2010, p. 168).

(2) o modelo de qualidade de Boehm

O modelo de qualidade de Boehm consiste de características de nível alto, características de nível intermediário e características de nível baixo (primitivas).
As três características de nível alto são:

(2.1) utilidade bruta

Esta característica de nível alto está relacionada à facilidade com que o software pode ser usado de forma fácil, confiável e eficiente sem que tenha que ser feito nenhuma modificação.

(2.2) manutenibilidade

Esta característica de nível alto está relacionada à facilidade com que o software pode ser compreendido, modificado e retestado.

(2.3) portabilidade

Esta característica de nível alto está relacionada com a possibilidade de utilizar o produto quando o ambiente se altera.

A ilustração 14 apresenta a estrutura do modelo de qualidade de Boehm e destaca as três características de nível alto.

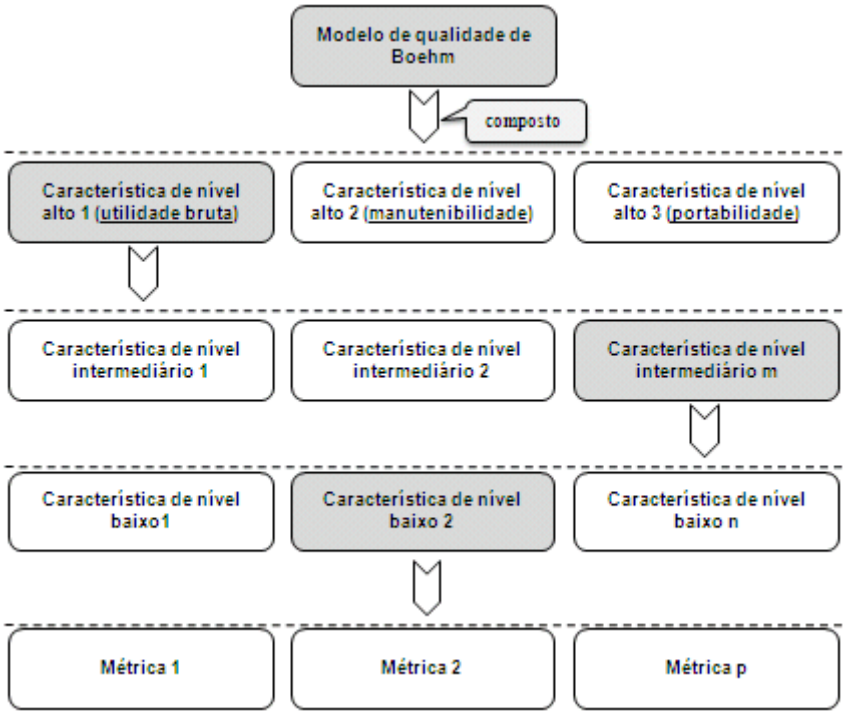

Ilustração 14 - Modelo de qualidade de Boehm.
Fonte: Adaptado de Al-Qutaish (2010, p. 169).

A ilustração 15 apresenta a decomposição das três características de nível alto em sete características de nível intermediário que por sua vez são decompostos em quinze características de baixo nível ou características primitivas distintas.

característica de alto nível	característica de nível intermediário	característica de nível baixo
utilidade bruta	confiabilidade	auto contido acurácia completude robustez/integridade consistência
	eficiência	responsabilidade eficiência de dispositivo acessibilidade robustez/integridade
	engenharia humana	acessibilidade comunicatividade
portabilidade		independência de dispositivo auto contido
manutenibilidade	testabilidade	responsabilidade comunicatividade auto descrição estruturação
	compreensibilidade	consistência estruturação concisão legibilidade
	modificabilidade	estruturação capacidade de aumento

Ilustração 15 - Modelo de qualidade de Boehm - decomposição.
Fonte: Adaptado de Al-Qutaish (2010, p. 169).

As sete características de nível intermediário são conceituadas da seguinte forma:

- portabilidade: característica associada à facilidade de utilizar o software em computadores com configurações diferentes da atual.
- confiabilidade: característica associada à capacidade de realizar suas funções continuamente de forma satisfatória.
- eficiência: característica associada à capacidade de não disperdiçar recursos.
- usabilidade (engenharia humana): característica associada à facilidade de se interagir com o software.
- testabilidade: característica associada à facilidade de se realizar uma verificação e avaliação.
- compreensibilidade: característica associada à facilidade de se compreender o software.
- flexibilidade (modificabilidade): característica associada à facilidade de se incorporar alterações.

As características de nível baixo podem ser usadas para se definir métricas de qualidade. Boehm define métrica como uma medida da extensão ou grau em que um produto possua e exiba uma certa característica de qualidade. Um ou mais métricas podem ser usadas para medir uma característica de qualidade de nível baixo.

(3) o modelo de qualidade ISO 9126

A ISO publicou o resultado do primeiro consenso internacional sobre a terminologia para as características de qualidade de software e sua avaliação em 1991. Este padrão ISO 9126 foi chamado de Avaliação de produtos de software – Características de qualidade e orientações para o seu uso.[7]
A versão atual do padrão ISO 9126 consiste de um padrão internacional (IS) e três relatórios técnicos (TR):

1. ISO IS 9126-1: Quality Model. (2001)
2. ISO TR 9126-2: External Metrics. (2003)
3. ISO TR 9126-3: Internal Metrics. (2003)
4. ISO TR 9126-4: Quality in Use Metrics. (2004)

O padrão internacional ISO IS 9126-1 é dividido em duas partes:

a) modelo de qualidade interno e externo.
b) modelo de qualidade "em uso".

O modelo de qualidade interno e externo identifica seis características que são decompostas em vinte e sete sub-características para as qualidades internas e externas.
A ilustração 16 apresenta a estrutura deste modelo de qualidade interno e externo.

---

[7] Software Product Evaluation - Quality Characteristics and Guidelines for Their Use

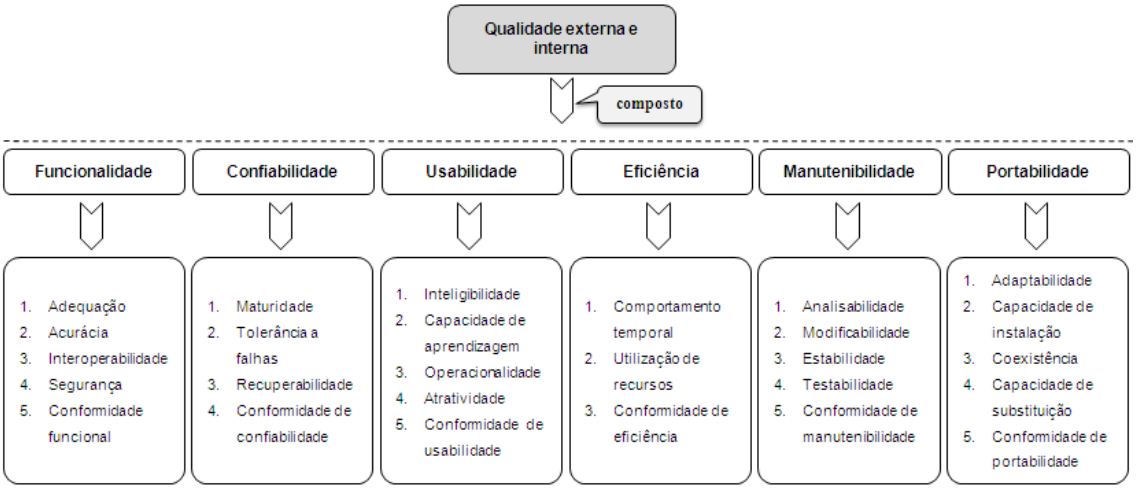

Ilustração 16 - Modelo de qualidade ISO 9126 - qualidade externa e interna.
Fonte: Adaptado de Al-Qutaish (2010, p. 171).

Estas sub-características são resultado de atributos internos do software e podem ser percebidas externamente durante o uso do software. A segunda parte do padrão internacional ISO IS 9126-1, modelo de qualidade "em uso", procura modelar a qualidade de software durante o seu uso. A ilustração 17 apresenta a estrutura deste modelo de qualidade "em uso".

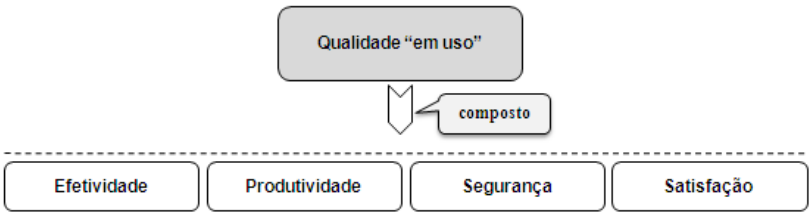

Ilustração 17 - Modelo de qualidade ISO 9126 - decomposição da qualidade "em uso".
Fonte: Adaptado de Al-Qutaish (2010, p. 171).

A ISO (2001) define as características e sub-características da qualidade interna e externa da seguinte forma:

1. Funcionalidade: a capacidade do produto de software de fornecer funções que atendam as necessidades explícitas e implícitas quando o software é utilizado sob condições especificadas.

1a. Adequação: a capacidade do produto de software de fornecer um conjunto apropriado de funções para tarefas especificadas e objetivos do usuário.
1b. Acurácia: a capacidade do produto de software de fornecer os resultados ou efeitos adequados ou acordados com o grau de precisão necessário.

1c. Interoperabilidade: a capacidade do produto de software de interagir com um ou mais sistemas especificados.
1d. Segurança: a capacidade do produto de software de proteger informações e dados de forma que pessoas ou sistemas não autorizados não possam ler ou modificá-las e que pessoas ou sistemas autorizados possam acessá-las.
1e. Conformidade funcional: a capacidade do produto de software de aderir a padrões, convenções ou regulamentos legais e outros requerimentos similares que abordem funcionalidade.

2. Confiabilidade: a capacidade do produto de software de manter um nível especificado de performance quando utilizado em condições especificadas.

2a. Maturidade: a capacidade do produto de software de evitar falhas em decorrência de faltas no software.
2b. Tolerância a falhas: a capacidade do produto de software de manter um nível especificado de performance em casos de faltas no software ou de violações em sua interface especificada.
2c. Recuperabilidade: a capacidade do produto de software de reestabelecer um nível especificado de performance em casos de faltas no software ou de violações em sua interface especificada.
2d. Conformidade de confiabilidade: a capacidade do produto de software de aderir a padrões, convenções ou regulações sobre confiabilidade.

3. Usabilidade: a capacidade do produto de software de ser compreendido, aprendido, usado e atrativo ao usuário quando utilizado em condições especificadas.

3a. Inteligibilidade: a capacidade do produto de software de permitir que o usuário entenda se o software é adequado e como ele pode ser usado para tarefas e condições de uso específicas.
3b. Capacidade de aprendizagem: a capacidade do produto de software de permitir que o usuário o compreenda.

3c. Operacionalidade: a capacidade do produto de software de permitir que o usuário o opere e o controle.
3d. Atratividade: a capacidade do produto de software de ser atrativo ao usuário.

3e. Conformidade de usabilidade: a capacidade do produto de software de aderir a padrões, convenções ou regulações sobre usabilidade.

4. Eficiência: a capacidade do produto de software de fornecer uma performance apropriada relativas a quantidade de recursos utilizados e sob condições determinadas.

4a. Comportamento temporal: a capacidade do produto de software de fornecer uma resposta, um tempo de processamento e uma taxa de transferência apropriada quando executando suas funções sob condições determinadas.
4b. Utilização de recursos: a capacidade do produto de software de usar uma quantidade e tipos apropriados de recursos quando executando suas funções sob condições determinadas.
4c. Conformidade de eficiência: a capacidade do produto de software de aderir a padrões, convenções ou regulações sobre eficiência.

5. Manutenibilidade: a capacidade do produto de software de ser modificado. Modificações podem incluir correções, melhorias ou adaptações do software a mudanças no ambiente e nas especificações funcionais e de requerimentos.

5a. Analisabilidade: a capacidade do produto de software de ser diagnosticado para se descobrir deficiências ou causas de falhas ou para se identificar as partes que devem ser modificadas.
5b. Modificabilidade: a capacidade do produto de software de permitir que uma modificação especificada seja implementada.
5c. Estabilidade: a capacidade do produto de software de evitar efeitos não esperados em decorrência de modificações.
5d. Testabilidade: a capacidade do produto de software de permitir que o software modificado possa ser validado.
5e. Conformidade de manutenibilidade: a capacidade do produto de software de aderir a padrões, convenções ou regulações sobre manutenibilidade.

6. Portabilidade: a capacidade do produto de software de ser transferido de um ambiente para outro.

6a. Adaptabilidade: a capacidade do produto de software de se adaptar a diferentes ambientes especificados sem a necessidade de se aplicar ações que não as que foram fornecidas para este propósito.
6b. Capacidade de instalação: a capacidade do produto de software de ser instalado em um ambiente especificado.
6c. Coexistência: a capacidade do produto de software de coexistir com outros softwares independentes em um ambiente comum compartilhando recursos comuns.
6d. Capacidade de substituição: a capacidade do produto de software de ser usado no lugar de outro software especificado com o mesmo propósito e no mesmo ambiente.
6e. Conformidade de portabilidade: a capacidade do produto de software de aderir a padrões, convenções ou regulações sobre portabilidade.

Os atributos de qualidade interna exercem influência sobre os atributos de qualidade externa que exercem influência sobre os atributos de qualidade "em uso". O relacionamento entre estes três tipos de atributos de qualidade é apresentado na ilustração 18.

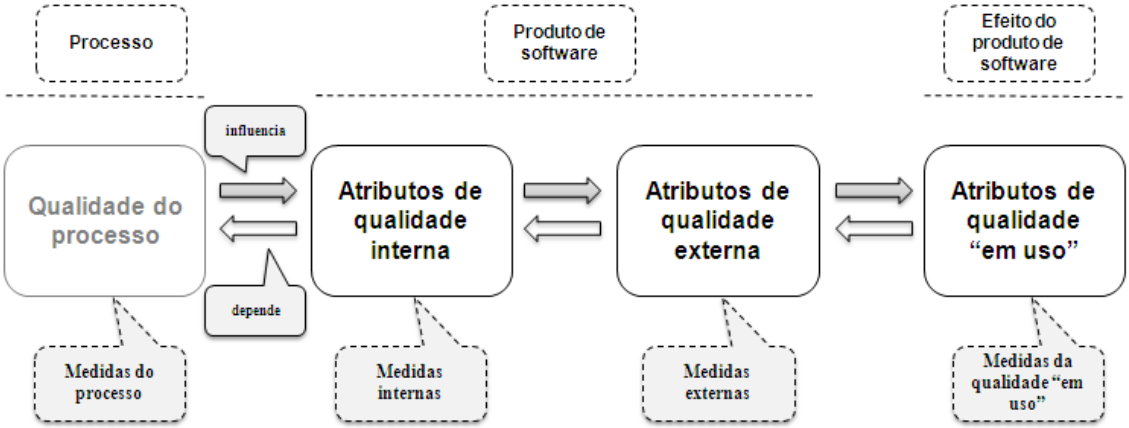

Ilustração 18 - Modelo de qualidade ISO 9126 - relação entre os atributos de qualidade.
Fonte: Adaptado de ISO (2001).

Apesar do modelo de qualidade ISO 9126 ser um padrão internacional, não é raro encontrar autores que citem alguns atributos de qualidade com conceitos que diferem ligeiramente dos conceitos dos modelos apresentados.

Sommerville (2006, p. 653) lista os seguintes atributos de qualidade de software: segurança[8], confiabilidade, resiliência, robustez, compreensibilidade[9], testabilidade, adaptabilidade, modularidade, complexidade, portabilidade, usabilidade, reusabilidade e eficiência.

Kandt (2006, p. 4) considera que os atributos de qualidade de software mais valorizados são: disponibilidade, eficiência, manutenibilidade, portabilidade, confiabilidade, reusabilidade e usabilidade.

Os três modelos de qualidade possuem duas catacterísticas em comum:

(1) Uma estrutura hierárquica que parte do conceito de qualidade de software no primeiro nível e decompõe recursivamente este conceito em um conjunto de características nos demais níveis.
(2) Um processo de decomposição e refinamento sucessivo ao longo dos níveis de forma que o último nível se torne preciso o suficiente para que métricas possam ser definidas.

Esta segunda parte do capítulo 3 apresentou três modelos de qualidade e destacou a importância dos modelos para possibilitar a definição de métricas de qualidade. A sub-característica de modificabilidade pertencente à característica de manutenibilidade do modelo de qualidade ISO 9126, ou em outras palavras, a capacidade de modificação é o atributo utilizado na aplicação apresentada no capítulo 6.

---

[8] O termo segurança está considerando dois atributos de qualidade: *safety* e *security*.
[9] O termo compreensibilidade está considerando dois atributos de qualidade: *understandability* e *learnability*.

3.3 Métricas de qualidade de software

As métricas de qualidade de software ou simplesmente métricas de software são formas de mensurar uma característica da qualidade. O termo característica citado refere-se aos conceitos apresentados no nível mais detalhado dos modelos de qualidade.

Para apresentar alguns exemplos de métricas de software associadas às características ou atributos de qualidade, apenas um atributo de qualidade é escolhido de forma que o exemplo possa ser apresentado detalhadamente.

Um ponto comum levantado por Deming, Feigenbaum e Ishikawa é a rapidez com que os requerimentos do produto de software mudam para continuar a atender as necessidades do mercado. Esse ponto levantado destaca a importância do atributo de qualidade manutenibilidade ou capacidade de modificação de um produto de software.

Os sistemas bem sucedidos tendem a evoluir e crescer. Quanto maior o tamanho do sistema, maior a importância do atributo de qualidade capacidade de modificação. Se o sistema não apresentar este atributo de qualidade, o tempo e o custo para se efetuar a manutenção e modificação do sistema poderão comprometer sua continuidade.

A atividade de manutenção é um ponto crucial do ciclo de vida de qualquer software e permite o seu uso efetivo por um longo período de tempo. Um fator que influi sobre esta atividade é a forma com que as fronteiras dos subsistemas são definidas. (KHAN, SOHAIL e JAVED, 2009, p. 1).

Existem inúmeras métricas de software para cada uma das características de qualidade apresentadas na parte 2 do capítulo 3. Segundo Land (2002, p. 4), as métricas podem ser classificadas em métricas de código e métricas de arquitetura.[10].

---

[10] Tradução livre para os termos *code-level measures* e *architecture-level measures* respectivamente.

Para o atributo de qualidade capacidade de modificação, algumas métricas de código são: *Lines Of Code* (LOC), número de linhas comentadas, comprimento de Halstead, volume de Halstead, esforço de Halstead, número de linhas em branco, número médio de LOC por módulo, complexidade ciclomática e razão entre LOC e número de linhas comentadas.

Para o mesmo atributo de qualidade não existem muitas métricas de arquitetura reportadas na literatura. A idéia básica explorada por estas métricas centra-se na investigação das interdependências entre os componentes. Geralmente a idéia é uma variação do número de chamadas realizadas para o componente ("*fan-in*") e do número de chamadas feitas pelo componente ("*fan-out*"). (LAND, 2002, p. 4).

Uma métrica de qualidade do tipo métrica de arquitetura para o atributo de qualidade capacidade de modificação é a métrica denominada Turbo MQ. Esta métrica é baseada na métrica básica de qualidade de modularização ("*Modularization Quality*", ou MQ) e permite que pesos sejam atribuídos às dependências entre os módulos. (MITCHELL, 2002, p. 65).

A medida ou métrica TurboMQ considera um gráfico representando os módulos ou clusters e suas dependências. Este gráfico é denominado de gráfico de dependência de módulos ("*Module Dependency Graph*" ou MDG). Assumindo um gráfico com k clusters, a métrica é o somatório de k fatores de cluster ("*Cluster Factor*" ou CF). Cada fator de cluster CF_i, onde i varia de 1 a k, é definido como a razão normalizada entre o valor total dos relacionamentos internos (relacionamentos internos ao cluster) e a metade do valor total dos relacionamentos externos (relacionamentos que saem ou entram no cluster). A razão de se considerar a metade do valor total dos relacionamentos externos é que o relacionamento ocorre entre dois clusters externos.

Representando o relacionamento interno de um cluster i por μ_i e o relacionamento externo entre um cluster i e um cluster j por ε_{ij} e ε_{ij} , e assumindo que os pesos de todas os relacionamentos é 1, a métrica TurboMQ pode ser calculada da seguinte forma:

$TurboMQ = \Sigma\ (i=1 \text{ to } k)\ CF_i$, onde

$CF_i = 0$ se $\mu_i = 0$ ou

$CF_i = \mu_i / (\mu_i + 0{,}5 * \Sigma\ (j=1 \text{ to } k,\ j \neq i)\ (\varepsilon_{ij} + \varepsilon_{ij}))$ caso contrário

A ilustração 19 apresenta um exemplo do cálculo do TurboMQ para um sistemas com 8 módulos divididos em 3 clusters.

TurboMQ = $CF_1 + CF_2 + CF_3 = 2/3 + 2/5 + 6/7 = 1{,}92$

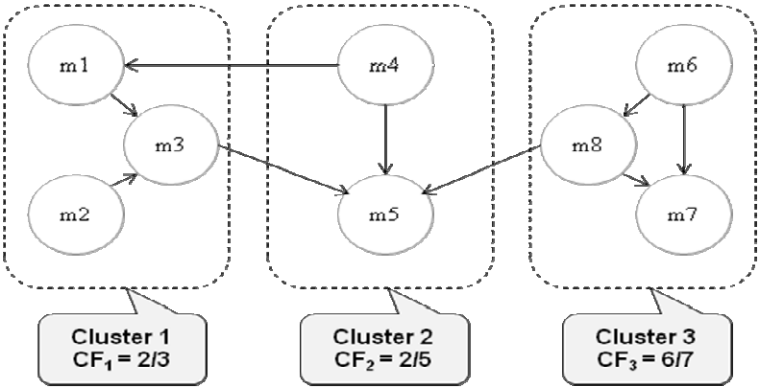

Ilustração 19 - Exemplo de cálculo da métrica TurboMQ.
Fonte: Adaptado de Mitchell (2002, p. 67).

As métricas de qualidade também podem ser classificadas em métricas qualitativas e métricas quantitativas. Algumas métricas como a métrica de pontos de função e a métrica de pontos de objetos dependem de uma intervenção humana. O conhecimento de especialistas é utilizado para julgar ítens como "simples", "médio" ou "complexo". (LAND, 2002, p. 4).

O quadro 1 apresenta dois atributos pelos quais as métricas de software podem ser classificadas.

Quadro 1 - Classificação de métricas de software.

Métricas de software	De código	De arquitetura
Qualitativa	Ex: pontos de função	Ex: algumas métricas obtidas por meio da aplicação do método ATAM
Quantitativa	Ex: linhas de código (LOC)	Ex: qualidade de modularização

As métricas, em última instância, procuram representar a qualidade de software. Por este motivo, quando uma arquitetura é desenhada para otimizar a qualidade de software, na verdade, um ou mais atributos de qualidade e mais especificamente um conjunto de métricas referentes a estes atributos são otimizados.

A este respeito DeMarco (2009, p. 96) comenta:

> "Hoje todos entendemos que as métricas de software custam tempo e dinheiro e precisam ser usadas com cuidadosa moderação. Além disso, o desenvolvimento de software é inerentemente diferente das ciências naturais como a física, e suas métricas são muito menos precisas em capturar as coisas que elas procuram descrever."[11]

Esta terceira parte do capítulo 3 apresentou diversas métricas de software que procuram quantificar uma característica da qualidade de software. O cálculo da métrica TurboMQ foi apresentado através de sua fórmula e de um exemplo. Esta métrica será utilizada no capítulo 6 onde assumirá o papel da função objetivo a ser maximizada.

---

[11] Today we all understand that software metrics cost money and time and must be used with careful moderation. In addition, software development is inherently different from a natural science such as physics, and its metrics are accordingly much less precise in capturing the things they set out to describe.

4 ARQUITETURA DE SOFTWARE

Este capítulo apresenta o desenvolvimento das etapas 4, 5 e 6 que concluem o segundo objetivo específico de "Conceituar arquitetura de software e pesquisar formas de representação". A ilustração 20 apresenta uma síntese do capítulo.

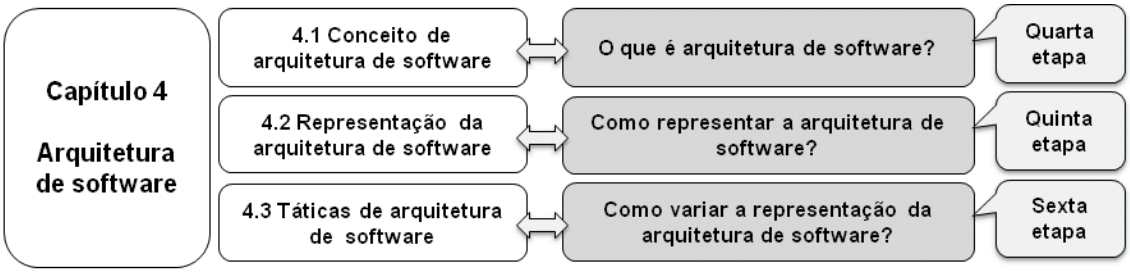

Ilustração 20 - Síntese do capítulo 4.

4.1 Conceito de arquitetura de software

Ainda não há uma definição universal para o termo arquitetura de software. O SEI apresenta em seu website uma coleção com mais de 150 definições obtidas da literatura e de profissionais da área. As cinco definições a seguir são coerentes e conceituam o termo arquitetura de software usado neste trabalho.

"A organização fundamental de um sistema representada por seus componentes, os relacionamentos entre os componentes e destes com o ambiente, e os princípios que guiam seu design e evolução."[12] (IEEE 1471 2000, p. 9)

"A arquitetura de um sistema de software concerne a decomposição de alto nível do sistema em seus principais componentes."[13] (BOSCH, 2000)

---

[12] The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
[13] The architecture of a software system is concerned with the top-level decomposition of the system into its main components.

"A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que compreende elementos de software, as propriedades externas visíveis destes elementos, e os relacionamentos entre eles."[14] (BASS, 2003, p.27).

"A arquitetura de software de um sistema computacional é o conjunto de estruturas necessárias para se analisar o sistema que compreende os elementos de software, os relacionamentos entre eles, e as propriedades de ambos."[15] (GARLAN, 2010, p.1).

A arquitetura de sistemas é um conjunto de entidades, suas propriedades, e seus relacionamentos que definem a estrutura do sistema e a arquitetura de software como um conjunto de componentes de software, subsistemas, relacionamentos, interações, as propriedades de cada um desses elementos, e o conjunto de princípios que juntos constituem as propriedades fundamentais e as restrições de um sistema de software ou de um conjunto de sistemas. (GARLAND, 2003).

Uma questão que decorre da definição de arquitetura de software é qual a diferença entre arquitetura e o design de alto nível de um sistema. Jalote (2008, p. 101) considera que esta fronteira não é clara e que devido a forma como esta área do conhecimento se desenvolveu, a fronteira é posicionada pelo arquiteto ou designer.

Garlan (2010, p. 9) estabelece que um critério para diferenciar arquitetura de design de alto nível é verificar se as decisões que os definem contribuem ou não para a obtenção de um atributo de qualidade. As decisões de arquitetura são as decisões que impactam um atributo de qualidade desejado e seu produto é a definição de parte da arquitetura de software. De forma análoga, as decisões de design de alto nível são as decisões que não influenciam um atributo de qualidade almejado e seu resultado é a definição de parte do design de alto nível.

---

[14] The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[15] The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

Conforme descrito por Garlan (1994, p. 2), à medida que o tamanho e a complexidade do software aumentam, o problema de design da arquitetura ultrapassa os algoritmos e estruturas de dados da computação e atinge o problema do design e da especificação da estrutura global do sistema. Essa questão inclui a organização geral e o controle global da estrutura; os protocolos de comunicação, sincronização e acesso aos dados; a designação de funcionalidades aos elementos de design; a distribuição física; a composição dos elementos de design e a seleção entre as alternativas de design.

Segundo Clements (2006, p. 1), a engenharia de software pode ser vista como uma série de ciclos nos quais as linguagens de programação se tornam cada vez mais sofisticadas e mais capazes de suportar abstrações do espaço de problemas. As linguagens binária e assembly deram lugar às linguagens C++ e Java que também cederam espaço para plataformas de desenvolvimento com uma interface web. Nas palavras de Clements:

> "Quando programas se tornam muito grandes e complexos para se compreender (ou projetar) trabalhando-se apenas com o código fonte, a arquitetura se torna um conceito indispensável. Ela ajuda as pessoas a pensarem em termos de abstrações no domínio do problema enquanto as linguagens de programação correntes não suportam."16

Assim como diversos autores, Lattanze (2009) também considera a arquitetura de software função dos requisitos não funcionais, ou em outras palavras, dos atributos de qualidade. Portanto, um software cuja arquitetura não tenha levado em consideração a capacidade de modificação e adaptação, será severamente prejudicado neste atributo.

Qin (2008, p. 25) argumenta que a arquitetura de software reflete as primeiras decisões de design que envolve todo o sistema, as quais causam os maiores impactos na implementação do sistema e são muito difíceis de serem alteradas posteriormente. Em outras palavras, a arquitetura de software permite representar as características gerais do sistema e analisar e avaliar a qualidade do sistema em uma fase inicial de seu desenvolvimento.

---

[16] When programs became to large and too complex to understand (or to engineer) by working only with source code, architecture became an indispensable concept. It helped people think in terms of problem-domain abstractions that the programming languages of the day did not support.

Existe um consenso sobre o fato de que os atributos de qualidade de um sistema de software são em grande parte restringidos por sua arquitetura. Como consequência, os requerimentos de qualidade mais importantes para o sucesso do sistema de software deveriam orientar o design da arquitetura de software. (BOSCH, 2001, p. 168).

Bass (2003) apresenta alguns exemplos que permitem relacionar qualidade e arquitetura de software. Primeiro, a qualidade de performance de um sistema depende do comportamento temporal de seus elementos e da freqüência e do volume de comunicação entre os elementos.

Segundo, a qualidade capacidade de modificação depende do mapeamento entre as responsabilidades e os elementos do sistema de forma que quando a modificação de uma responsabilidade for necessária, o menor número possível de elementos seja envolvido.

Terceiro, a qualidade de segurança de um sistema depende da proteção da comunicação entre os elementos e da definição de quais elementos podem acessar quais informações.

Quarto, a qualidade de escalabilidade de um recurso do sistema depende do grau de localização (ou concentração) do recurso de forma que quando for necessário um recurso de maior capacidade, ele possa ser introduzido facilmente.

Quinto, a qualidade ou capacidade de reutilização de elementos de um sistema em outros depende do grau de acoplamento entre os elementos de forma que quando for necessário utilizar um elemento em outros sistemas não seja necessário considerar uma quantidade grande de outros elementos que se relacionam a este.

Os exemplos mostram que os atributos de qualidade dependem da arquitetura. No entanto é importante destacar que apenas a arquitetura não define os atributos de qualidade, grosso modo, uma arquitetura adequada é necessária mas não suficiente para se definir a qualidade do sistema. Os produtos de todas as etapas do ciclo de vida de desenvolvimento de software, desde o design de alto nível até a codificação e implementação, influenciam a qualidade do software.

Esta primeira parte do capítulo 4 apresentou o conceito de cinco reconhecidos autores a respeito da arquitetura de software. Cinco exemplos foram apresentados para mostrar a existência de uma relação entre arquitetura de software e qualidade de software.

4.2 Representação da arquitetura de software

Uma das dificuldades de se representar a arquitetura de software é a quantidade de informações necessária e a definição de uma linguagem ou notação adequada. Conforme observado por Kruchten (1995):

> "Nós todos temos vistos livros e artigos nos quais um único diagrama tenta capturar o todo de uma arquitetura de sistema. Mas quando você olha cuidadosamente para as caixas e flechas do diagrama, se torna claro que o autor está tentando representar mais em um único diagrama do que é praticável."[17]

De forma análoga ao que ocorre com os modelos de qualidade, as formas de representação da arquitetura de software são apresentadas por diversos autores com conceitos que variam um pouco de autor para autor. O entendimento de dois autores, (1) Kim (2008) e (2) Bass (2003), sobre a forma de se representar a arquitetura de software é apresentado a seguir:

(1) Kin (2008) descreve um conjunto de visões que permite caracterizar a arquitetura de software de um sistema:

Visão de componentes e conectores: componentes são unidades funcionais que interagem através de uma série de interfaces predefinidas. Essa forma de acesso restrita permite que os componentes sejam encapsulados o que por sua vez permite sua independência e intercâmbio. Os conectores abstraem o protocolo de comunicação geralmente mais complexo que os mecanismos de comunicações simples como uma comunicação assíncrona. Essa visão permite representar algumas táticas de arquitetura utilizadas para lidar com alguns requisitos não funcionais. Um exemplo é a utilização de um canal de comunicação encriptado caso seja necessária alta segurança sobre os dados. Um segundo exemplo é a utilização de redundância se o requisito de alta disponibilidade for solicitado.

---

[17] We all have seen many books and articles in which a single diagram attempts to capture the gist of a system architecture. But when you look carefully at the diagram's boxes and arrows, it becomes clear that the author are struggling to represent more in one diagram than is practical.

Visão decomposicional: permite visualizar a decomposição do sistema em diversos conceitos lógicos através de um estilo *top-down*. Os conceitos lógicos são conectados de acordo com seus relacionamentos no mundo real. Exemplos de relacionamentos são *uso* e *associação*. Essa visão facilita a identificação de blocos básicos utilizados em diversas partes do sistema incentivando a prática do reuso. Um conjunto de conceitos lógicos e seus relacionamentos podem ser agrupados em módulos de forma a fornecer novos comportamentos e facilitar o processo de design, implementação e teste. O conceito de módulo difere do conteito de componente apresentado na visão anterior. O componente é uma abstração lógica de uma unidade funcional em runtime e pode ser implementada por um ou mais módulos.

Visão de alocação: a visão de alocação pode ser subdividida em visão de implementação e visão de disponibilização.[18] A visão de implementação focaliza a implementação das unidades lógicas pelos arquivos fontes e os relacionamentos entre eles. Os arquivos fontes incluem os arquivos de código fonte, arquivos de declaração (como por exemplo, os arquivos de cabeçalho em C++), arquivos de configuração (como por exemplo, o arquivo de script Ant para Java), e etc. Essa visão permite que desenvolvedores e engenheiros de teste compreendam o que eles estão construindo ou testando e qual a influência disso no sistema. A ilustração 21 mostra um exemplo de visão de implementação.

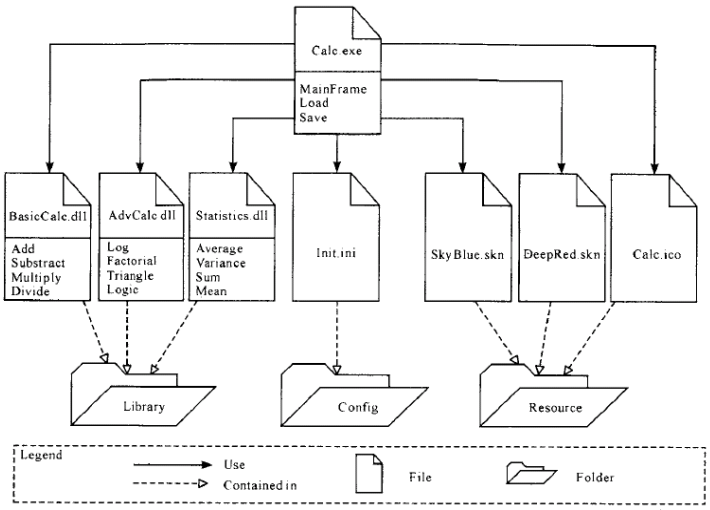

Ilustração 21 - Exemplo de visão de implementação.
FONTE: QIN, 2008, p. 20.

[18] A visão de alocação é uma tradução livre de *allocation view*; a visão de implementação, de *implementation view* e a visão de disponibilização, de *deployment view*.

A visão de disponibilização focaliza o relacionamento entre os elementos de software e os elementos de hardware. Módulos, componentes e processos são exemplos de elementos de software enquanto que servidores, roteadores e dispositivos móveis exemplificam os elementos de hardware. A representação desta visão pode ser feita de várias formas, desde ícones gráficos até notações definidas de forma estrita em uma linguagem formal. A ilustração 22 apresenta um exemplo de visão de disponibilização.

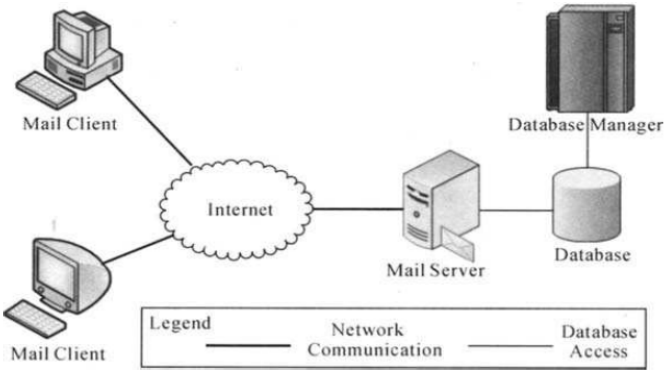

Ilustração 22 - Exemplo de visão de disponibilização.
FONTE: QIN, 2008, p. 20.

Visão de comportamento: focaliza os detalhes da interação entre os elementos do sistema e seu comportamento interno. Existem vários tipos de visão de comportamento que podem ser generalizados em três categorias:

. Comportamento centrado em mensagem: este estilo enfatiza a colaboração dos elementos na troca de mensagens que podem ser chamadas, transmissão de sinal ou uma comunicação assíncrona. Os diagramas de sequência e colaboração da UML são exemplos típicos. A ilustração 23 apresenta um exemplo de diagrama de sequência da UML.

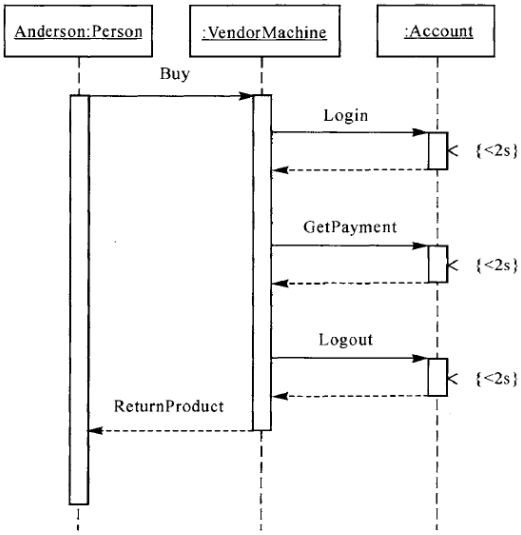

Ilustração 23 - Exemplo de diagrama de sequência da UML.
FONTE: QIN, 2008, p. 22.

. Comportamento de atividade: este estilo focaliza como uma tarefa pode ser processada tornando claro quais os passos necessários a serem implementados. O *flowchart* (mecanismo de visualização muito utilizado no paradigma da programação estruturada) e o diagrama de atividades da UML são dois dos representantes deste estilo. A ilustração 24 apresenta um exemplo de *flowchart* de controle.

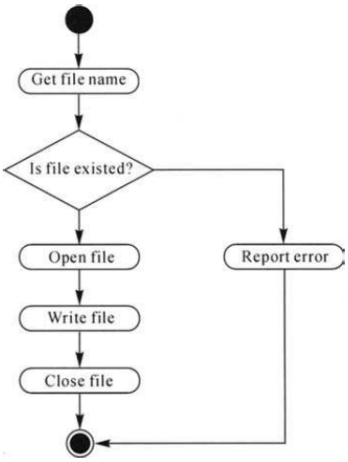

Ilustração 24 - Exemplo de *flowchart* de controle.
FONTE: QIN, 2008, p. 23.

(2) Bass (2003) considera que a complexidade de um sistema pode tornar difícil sua compreensão e visualização. Uma forma de lidar com a complexidade é utilizar o princípio Dividir e Conquistar. As visões da arquitetura podem ser consideradas como o resultado da aplicação deste princípio. Para uma dada visão da arquitetura considera-se apenas um

conjunto particular de elementos do sistema. Um conjunto coerente de elementos define uma estrutura. Portanto, pode-se considerar uma dada visão da arquitetura como a representação de uma dada estrutura da arquitetura e sua respectiva organização.

Bass (2003) descreve um exemplo considerando o corpo humano como o sistema complexo cuja arquitetura deseja-se compreender.
"O neurologista, o ortopedista, o hematologista, e o dermatologista todos tem uma visão diferente da estrutura do corpo humano. Oftalmologistas, cardiologistas, e pediatras concentram-se nos subsistemas. [...] Embora essas visões sejam representadas de forma diferente e tenham propriedades muito diferentes, todas elas são inerentemente relacionadas: juntas elas descrevem a arquitetura do corpo humano."

Bass (2003) considera a natureza dos elementos como critério para uma primeira classificação das visões da arquitetura. A partir deste critério, três tipos de visões da arquitetura foram estabelecidas.

Visão de módulo: enfatiza a organização do código-fonte. Os módulos são os elementos da arquitetura e estão associados à implementação de uma ou mais responsabilidades. Exemplos de visões baseadas em módulos:

. *Visão de decomposição*: apresenta a decomposição de módulos maiores em módulos menores recursivamente até que os módulos tenham um tamanho que possibilite uma fácil compreensão. O tipo de relacionamento desta visão é denominado de "é um sub-módulo de".

. *Visão de uso*: apresenta o relacionamento do tipo uso entre os módulos. Um módulo usa ou depende de outro módulo se o funcionamento do primeiro depende do funcionamento do segundo módulo.

. *Visão de camadas*: apresenta o relacionamento do tipo uso entre grupos de módulos. Um grupo de módulos que implementa um conjunto coerente de responsabilidades é denominado uma camada. Em uma arquitetura de camadas em sua forma restrita, uma camada n pode usar apenas as funcionalidades da camada n-1. Essa arquitetura favorece naturalmente o atributo de qualidade portabilidade pois uma camada funciona como uma máquina virtual para a camada superior desacoplando esta camada da camada inferior.

. *Visão de classes*: apresenta o relacionamento do tipo "herdado de" entre as classes. As classes representam os elementos da arquitetura e são equivalentes aos módulos da visão de decomposição. Essa visão facilita a compreensão das possibilidades de reutilização e adição de novas responsabilidades (ou funcionalidades).

Visão de componente e conector: enfatiza a organização e comportamento em tempo de execução. Os componentes em tempo de execução são os elementos da arquitetura e os conectores representam a comunicação ou relacionamento entre os componentes. Exemplos de visões baseadas em componente e conector:

. *Visão de processo*: os elementos da arquitetura são processos e threads e o tipo de conector pode ser de comunicação, sincronização ou exclusão. Essa visão enfatiza os aspectos dinâmicos do sistema em tempo de execução. Essa visão favorece a análise dos atributos de qualidade performance e disponibilidade do sistema.

. *Visão de concorrência*: os elementos da arquitetura são componentes e os conectores são denominados de threads lógicas. Uma thread lógica é considerada uma sequência de computação que pode ser alocada a uma thread física posteriormente. Essa visão facilita a identificação de pontos onde pode ocorrer contenção de recursos e de pontos onde a aplicação de paralelismo pode ser benéfica.

. *Visão de dados compartilhados ou de repositório*: enfatiza o relacionamento entre os componentes que podem criar, acessar, modificar e remover dados persistentes e os componentes que armazenam estes dados denominados repositórios.

. *Visão cliente-servidor*: os componentes são os clientes e servidores e os conectores são os protocolos utilizados para a comunicação. Essa visão destaca a separação de responsabilidades entre clientes e servidores o que favorece o atributo de qualidade capacidade de modificação.

Visão de alocação: enfatiza a associação entre elementos de software e elementos do ambiente externo. Um exemplo de visão de alocação é considerar módulos como elementos de software

e arquivos como elementos do ambiente externo. Outro exemplo pode considerar componentes como elementos de software e processadores como elementos do ambiente externo. Um terceiro exemplo pode considerar módulos como elementos de software e grupos de desenvolvimento como elementos do ambiente externo. Exemplos de visões baseadas em alocação:

. *Visão de implantação*: os elementos de software são processos e os elementos do ambiente externo são processadores (elementos de hardware). O relacionamento pode ser do tipo "alocado para" que associa os elementos de software aos elementos de hardware ou do tipo "migrar para" se a alocação for dinâmica. Esta visão apresenta informações relevantes para se pensar sobre os atributos de qualidade performance, integridade de dados, disponibilidade e segurança.

. *Visão de implementação*: os elementos de software são usualmente módulos e os elementos do ambiente externo são arquivos. O relacionamento identifica a associação entre os módulos e uma estrutura de arquivos. Esta visão apresenta informações importantes para o gerenciamento das atividades de desenvolvimento e dos processos de construção ("*build process*").

. *Visão de distribuição de trabalho* ("*work assignment*"): os elementos de software são módulos e os elementos do ambiente externo são grupos de desenvolvimento. O relacionamento identifica quais grupos de desenvolvimento são responsáveis pela implementação e integração de quais módulos do sistema. Esta visão permite estimar o conhecimento necessário para cada grupo de desenvolvimento.

Os elementos do conjunto de visões de arquitetura e o aspecto do sistema que é capturado por cada uma das visões apresentam alguma variação de autor para autor. Apesar de algumas diferenças todas as formas de representação da arquitetura de software podem ser enquadradas em um esquema geral conforme apresentado na ilustração 25.

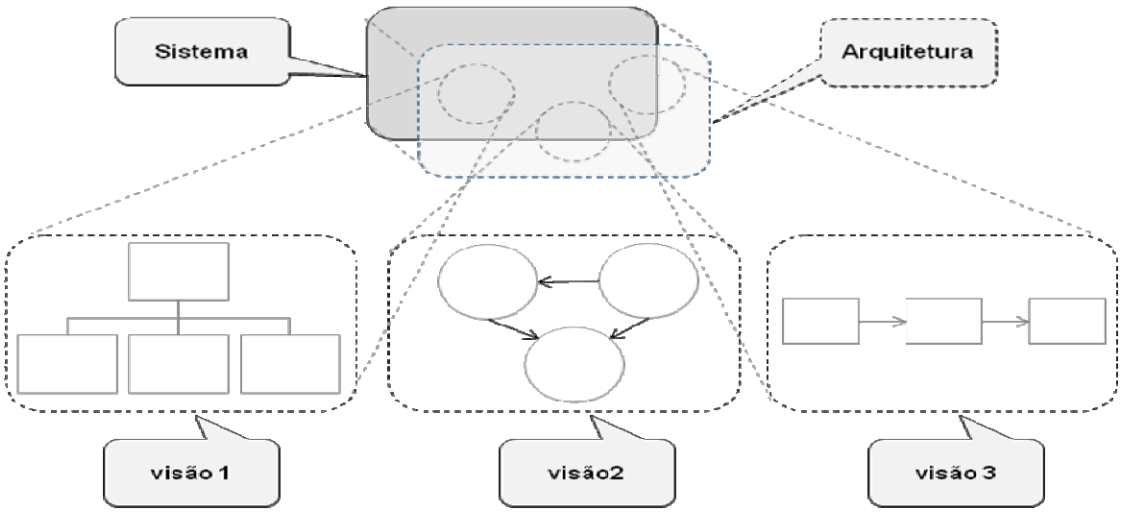

Ilustração 25 - Esquema geral das formas de representação da arquitetura de software.
Fonte: Adaptado de QIN (2008, p. 15).

Esta segunda parte do capítulo 4 apresentou formas de representação da arquitetura de software e destacou que apesar de existirem diferenças entre os autores a respeito de que conjunto de visões adotar e o que cada visão deve focalizar, a arquitetura de software é representada por visões que capturam uma perspectiva do sistema conforme apresentado na ilustração 25.

A próxima parte, terceira parte do capítulo 4, procura evidenciar a relação entre a arquitetura de software (discutida neste capítulo) e a qualidade de software (discutida no capítulo 3). Em particular, a terceira parte considera o atributo de qualidade capacidade de modificação e sua relação com representações diferentes da arquitetura de software. Diferentes arquiteturas de software podem ser vistas como resultado de diferentes decisões de design e boas decisões de design são registradas (capturadas) na forma de táticas de arquitetura.

4.3 Táticas de arquitetura de software

As táticas de arquitetura são decisões de design fundamentais. O processo de design de um sistema pode ser visto como um conjunto de decisões. Existem decisões que permitem a implementação das funcionalidades e decisões que exercem influência sobre um atributo de qualidade do sistema. Esse segundo tipo de decisões é referido como decisões de design fundamentais ou táticas de arquitetura.

O sentido em descrever as táticas como decisões é que cada tática é vista como uma opção de decisão que o arquiteto pode adotar. Ao se decidir por uma opção o arquiteto aplica uma tática de arquitetura e define um aspecto fundamental do design do sistema.

Uma tática comum é a introdução de redundância para aumentar a disponibilidade de um sistema. De posse do conhecimento desta tática, o arquiteto pode optar por sua aplicação. Essa opção pode ser vista como uma decisão de design fundamental uma vez que exerce influência sobre um atributo de qualidade.

A origem das táticas de arquitetura é a experiência obtida pelos arquitetos ao longo do tempo e o conjunto de táticas conhecidas é a documentação da síntese dessa experiência adquirida ao enfrentar os problemas que se apresentaram. Nas palavras de Bass (2003): "As táticas são aquelas que os arquitetos têm usado por anos, e nós as isolamos e descrevemos. Nós não estamos inventando táticas aqui, apenas capturando o que os arquitetos fazem na prática."[19]

Para o caso do atributo de qualidade capacidade de modificação, as táticas de arquitetura podem ser organizadas em três categorias:

A primeira categoria reúne as táticas que tem por objetivo reduzir o número de módulos que são afetados diretamente pela necessidade de se realizar uma mudança. Essa categoria é denominada de "Localizar as modificações"[20].

---

[19] The tactics are those that architects have been using for years, and we isolate and describe them. We are not inventing tactics here, just capturing what architects do in practice.
[20] Localize modifications

A segunda categoria engloba as táticas que procuram restringir as modificações apenas aos módulos diretamente afetados. Os módulos diretamente afetados são os módulos cujas responsabilidades são modificadas para atender a mudança requerida e os módulos indiretamente afetados são aqueles que não têm suas responsabilidades alteradas mas precisam ser modificados para acomodar as modificações realizadas nos módulos diretamente afetados. Essa categoria é chamada de "Evitar efeitos colaterais"[21].

A terceira categoria de táticas considera as táticas que almejam reduzir o tempo e custo para se efetuar uma mudança e possibilitar que determinadas alterações possam ser realizadas diretamente pelos usuários finais. Essa terceira categoria é conhecida por "Adiar o tempo de ligação"[22].

Categoria "Localizar as modificações"

As táticas desta categoria assumem que existe uma relação entre o custo e tempo para se implementar uma mudança e o número de módulos afetados por esta mudança. Assume-se que geralmente restringir as modificações ao menor número de módulos possíveis resulta em um menor tempo e custo para realizar a mudança. A idéia é alocar as responsabilidades aos módulos de forma que as mudanças mais prováveis afetem o menor número possível de módulos. A descrição de três táticas desta categoria é apresentada a seguir:

.1 "Manter coerência semântica"[23]

O termo coerência semântica se refere ao relacionamento entre as responsabilidades de um módulo. A informação a respeito de quais mudanças tem a maior probabilidade de ocorrer definem o grau de relacionamento entre as responsabilidades. Dado um cenário concreto para uma determinada mudança, procura-se alocar as responsabilidades afetadas em um ou em um pequeno número de módulos pois entende-se que estas responsabilidades têm um relacionamento forte.

---

[21] Prevent ripple effect
[22] Defer binding time
[23] Maintain semantic coherence

.2 “Generalizar o módulo”[24]

O conceito de generalizar o módulo é generalizar o tipo de responsabilidade que o módulo implementa. Esta generalização é obtida tornando a interface mais flexível. A interface pode ser vista como uma linguagem para se comunicar com o módulo. Esta linguagem pode ser simples e rígida como um conjunto de parâmetros de entrada ou complexa e flexível como uma linguagem de programação específica deste módulo. No caso da interface funcionar como uma linguagem de programação, o módulo assume a responsabilidade de interpretar a linguagem. A aplicação deste conceito aumenta a probabilidade de que uma requisição de mudança possa ser atendida usando-se esta linguagem ao invés de se modificar o módulo.

.3 “Abstrair serviços comuns”[25]

O conceito de se abstrair serviços comuns é implementar em um único módulo uma funcionalidade comum a diversos módulos. Esta implementação da funcionalidade comum em um único módulo evita que cada um dos módulos tenha a responsabilidade de implementá-la. O benefício em se usar esta tática ocorre para o cenário em que uma mudança da funcionalidade comum for solicitada. Dada a ocorrência deste cenário, apenas uma modificação no módulo que implementa a funcionalidade comum é necessária deixando os demais módulos intactos.

Categoria “Evitar efeitos colaterais”

O conceito de efeito colateral é a necessidade de se realizar uma modificação a um módulo que não é diretamente afetado pela necessidade de se realizar uma determinada mudança. Considerando dois módulos A e B, se a mudança afetar diretamente um módulo A, ou seja, se uma ou mais funcionalidades do módulo A tiver que ser alterada, então uma modificação no módulo B é necessária apenas devido a alteração do módulo A. Essa necessidade existe porque, de alguma forma, o módulo B depende do módulo A. Bass (2003) descreve oito tipos de dependência:

---

[24] Generalize the module
[25] Abstract common services

.1 Sintaxe de

.1.1 Dados: para que o módulo B possa ser compilado ou executado corretamente, o tipo (ou formato) dos dados produzidos por A e consumidos por B devem ser consistentes com o tipo (ou formato) de dados assumidos pelo módulo B.
1.2 Serviços: para que o módulo B possa ser compilado ou executado corretamente, a assinatura dos serviços fornecidos por A e utilizados por B devem ser consistentes com a assinatura dos serviços assumidos pelo módulo B.

.2 Semântica de

.2.1 Dados: para que o módulo B possa ser executado corretamente, a semântica dos dados produzidos por A e consumidos por B devem ser consistentes com a semântica dos dados assumidos pelo módulo B.
.2.1 Serviços: para que o módulo B possa ser executado corretamente, a semântica dos serviços fornecidos por A e utilizados por B devem ser consistentes com a semântica dos serviços assumidos pelo módulo B.

.3 Seqüência de

.3.1 Dados: para que o módulo B possa ser executado corretamente, ele precisa receber os dados produzidos por A em uma determinada seqüência.
.3.2 Controle: para que o módulo B possa ser executado corretamente, o módulo A precisa ser previamente executado respeitando alguma restrição temporal.

.4 Identidade da interface de um módulo: Considerando que o módulo A possua diversas interfaces, para que o módulo B possa ser compilado ou executado corretamente, a identidade ou nome de uma interface de A deve ser consistente com a identidade ou nome da interface assumida por B.

.5 Localização de um módulo em tempo de execução: para que o módulo B possa ser executado corretamente, a localização do módulo A em tempo de execução deve ser consistente com a localização assumida pelo módulo B.

.6 Qualidade de

.6.1 Dados: para que o módulo B possa ser executado corretamente, alguma propriedade da qualidade dos dados produzidos por A e consumidos por B deve ser consistente com a assumida pelo módulo B. Por exemplo, considerando que o módulo A é responsável por enviar os dados capturados por um sensor para o módulo B, a precisão dos dados enviados por A deve satisfazer a precisão assumida pelo módulo B.
.6.2 Serviços: para que o módulo B possa ser executado corretamente, alguma propriedade da qualidade dos serviços fornecidos por A e utilizados por B deve ser consistente com a assumida pelo módulo B.

.7 Existência de um módulo: para que o módulo B possa ser executado corretamente, o módulo A deve existir. Por exemplo, se o módulo B solicitar um serviço do módulo A, e o módulo A não existir e não puder ser criado dinamicamente, então o módulo B não pode executar corretamente.

.8 Característica de um recurso utilizado por um módulo: para que o módulo B possa ser executado corretamente, uma característica de um recurso utilizado pelo módulo A deve ser consistente com a assumida pelo módulo B. Por exemplo, considerando o recurso memória e a característica localização, o módulo B pode não executar corretamente se o módulo A não usar a mesma localização (espaço) de memória utilizado pelo módulo B.

O conhecimento dos tipos de dependência auxilia a compreender como as táticas dessa categoria permitem evitar os efeitos colaterais resultantes de uma mudança que afete um dos tipos de dependência.

.1 “Ocultamento de informações”[26]

---

[26] Information hiding

Essa tática almeja diminuir o impacto de efeitos colaterais ao se realizar uma mudança em um módulo expondo o mínimo conjunto de responsabilidades para os demais módulos. Em outras palavras, a tática procura ocultar informações (conhecimentos) contidas em um módulo dos demais módulos de forma que alterações na parte oculta deste módulo não causem a necessidade de se alterar outros módulos. O conhecimento a respeito de quais mudanças tem maior probabilidade de ocorrência indicam que responsabilidades podem ser ocultadas de forma a maximizar os benefícios desta tática.

.2 "Manter interfaces existentes"[27]

Esta tática procura minimizar a propagação de efeitos colaterais mantendo os módulos com interfaces estáveis. A estabilidade da interface mantém a sintaxe dos dados e dos serviços da interface constante. Um módulo B que tenha uma dependência somente de sintaxe sobre a interface de um módulo A não será afetado por uma mudança no módulo A desde que a interface seja mantida. Outros tipos de dependência como a dependência de semântica, de seqüência, de qualidade, e de uma característica de um recurso utilizado pelo módulo A não são mascaradas (anuladas) por esta tática.

Bass (2003) observa que os padrões de design podem ser vistos como uma forma de se implementar táticas de arquitetura. Deste ponto de vista os padrões de design podem ser analisados quanto as táticas de arquitetura que implementam e conseqüentemente a decisão de aplicar um determinado padrão de design se torna uma decisão de design.

Uma forma de implementar uma tática desta categoria é adicionar um adaptador entre o módulo A e o módulo B. Uma mudança na interface do módulo A não afeta mais o módulo B pois o adaptador assume a responsabilidade de manter a interface original do módulo A para o módulo B.

---

[27] Maintain existing interfaces

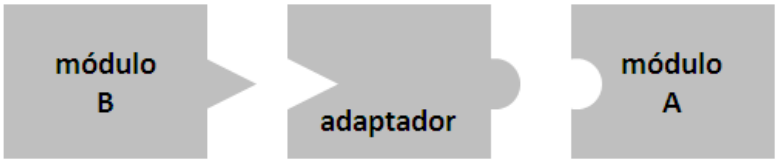

Ilustração 26 - Exemplo da tática "Manter interfaces existentes".

Com a adição do adaptador uma mudança na interface do módulo A pode ser acomodada através de uma alteração no adaptador deixando o módulo B intacto. Considera-se que o custo de se alterar o módulo B é maior que o custo de se alterar o adaptador. Se o módulo B for composto por diversos sub-módulos e cada sub-módulo precisar ser alterado para acomodar a mudança do módulo A, a consideração assumida é provavelmente válida.

Gamma (1995) considera que o padrão de design Adaptador[28] tem a intenção de converter a interface de uma classe A em outra interface esperada por uma classe B. Esse padrão de design permite que duas classes com interfaces incompatíveis se comuniquem através do adaptador.

.3 “Restringir os caminhos de comunicação”[29]

Essa tática de modificação procura minimizar o número de caminhos de comunicação de dados entre os módulos. Se cada um dos módulos A, B e C produzem dados que são consumidos por cada um dos módulos D, E e F, um total de nove caminhos de comunicação existem. Um cenário de modificação que requeira a mudança do módulo A para o módulo A’ pode afetar os três módulos D, E e F que consomem os dados produzidos pelo módulo A. Cada um dos três módulos podem ter que referenciar o novo módulo A’. Se um novo módulo G for introduzido no sistema e assumir a responsabilidade de receber os dados fornecidos pelos módulos A, B e C e entregar os dados utilizados pelos módulos D, E e F; para o mesmo cenário de modificação apenas o módulo G pode ser modificado de forma a referenciar o novo módulo A’.

.4 “Usar um intermediário”

---

[28] Adapter design pattern
[29] Restrict communication paths

Essa tática de modificação procura anular a dependência de um módulo B sobre um módulo A através da introdução de um módulo intermediário que assume a responsabilidade de gerenciar a dependência. Bass (2003) considera que a dependência do tipo semântica não pode ser tratada aplicando-se essa tática. Para os demais tipos de dependência algumas soluções são apontadas:

. Sintaxe de dados: se um módulo B tem uma dependência de sintaxe de dados em relação a um módulo A, a tática "Usar um intermediário" pode ser utilizada para anular essa dependência. Os padrões de arquitetura Repositório, Publicar/Subscrever e MVC podem ser utilizados para implementar essa tática. [30]

. Sintaxe de serviços: se um módulo B tem uma dependência de sitaxe de serviços em relação a um módulo A, os padrões de design Façade, Bridge, Mediator, Strategy, Proxy e Factory podem ser utilizados para introduzir um intermediário responsável por converter a sintaxe de serviço fornecido pelo módulo A para a sintaxe de serviço requerida.

Gamma (1995) considera que o objetivo do padrão de design Ponte ("*Bridge*") é desacoplar uma abstração de sua implementação com a finalidade de que uma mudança na abstração não afete a implementação e vice-versa.

A ilustração 27 apresenta o diagrama de classes do padrão de design Ponte. Esse design introduz um intermediário *Abstraction* que converte a sintaxe de serviço fornecido pelo módulo A (*Implementor*) para a sintaxe de serviço requerida pelo módulo B (*Client*).

Uma mudança na sintaxe de serviço do módulo A (*Implementor*) de OperationImp() para OperationImp2() não tem o efeito colateral de afetar o módulo B (*Client*) uma vez que essa mudança pode ser tratada pelo intermediário (*Abstraction*) alterando a implementação do serviço ou método Operation() de "imp->OperationImp()" para "imp->OperationImp2()".

---

[30] Uma visão geral dos padrões de arquitetura Repositório, Publicar/Subscrever e MVC são apresentados no capítulo Padrões de Arquitetura.

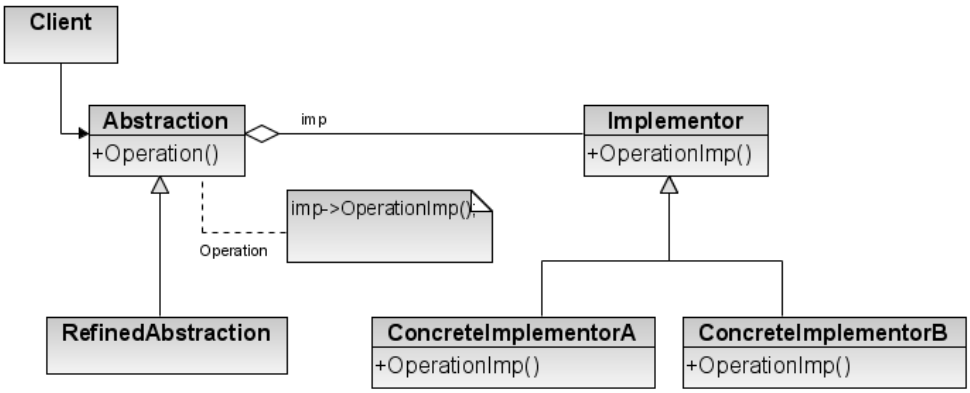

Ilustração 27 - Diagrama de classes do padrão de design Ponte.
Fonte:http://upload.wikimedia.org/wikipedia/commons/1/14/Bridge_design_pattern.png

. Identidade da interface de um módulo: se um módulo B depende da identidade da interface de um módulo A, o padrão de arquitetura *Broker* pode ser aplicado para anular esse tipo de dependência. Nessa arquitetura um componente denominado broker assume a responsabilidade de manter a identidade da interface vista pelo módulo B fixa apesar de mudanças na identidade da interface do módulo A.

Avgeriou (2005, p. 34) contextualiza o padrão de arquitetura *Broker* dentro de sistemas de software distribuído. Nesse contexto, a arquitetura *Broker* procura separar a funcionalidade relacionada a comunicação entre os objetos da funcionalidade relacionada a aplicação. Uma das características da arquitetura é a transparência de localização dos objetos do sistema. Essa transparência permite que a identidade da interface de um módulo A possa ser alterada sem introduzir efeitos colaterais em um módulo B. A alteração pode ser acomodada pelos componentes da arquitetura *Broker*.

. Localização de um módulo em tempo de execução: se um módulo B depende da localização de um módulo A em tempo de execução, um servidor de nomes pode ser usado para anular esse tipo de dependência. A alteração da localização do módulo A pode ser mascarada pelo servidor de nomes que deve ser atualizado com a informação da nova localização pelo módulo A. Após essa atualização, uma nova requisição do módulo B é direcionada para o servidor de nomes (o componente intermediário) que assume a responsabilidade de direcionar a requisição para a nova localização do módulo A. O padrão de arquitetura *Broker* pode ser utilizado para anular esse tipo de dependência usando a mesma idéia aplicada para anular a dependência do tipo identidade da interface de um módulo descrita no item anterior.

. Característica de um recurso utilizado por um módulo: se um módulo B depende da característica de um recurso utilizado por um módulo A, um gerenciador de recurso pode fazer o papel do componente intermediário e anular esse tipo de dependência. No contexto de sistemas de tempo real um módulo B pode depender do tempo de resposta de um módulo A. O tempo de resposta do módulo A pode ser visto como uma função da forma de alocação (característica) de tarefas ao processador (recurso). Essa característica desse recurso, ou seja, a forma de alocação de tarefas ao processador pode ser delegada a um gerenciador de recursos (o componente intermediário) de forma que a dependência seja transferida do módulo A para o gerenciador de recurso.

Quadro 2 - Táticas de capacidade de modificação.

Táticas de capacidade de modificação		
Localizar as modificações	**Evitar efeitos colaterais**	**Adiar tempo de ligação**
Manter coerência semântica	Ocultamento de informações	Registro em tempo de execução
Generalizar o módulo	Manter interfaces existentes	Arquivos de configurações
Abstrair serviços comuns	Restringir os caminhos de comunicação	Troca de componentes
	Usar um intermediário	Aderir à protocolos definidos

Fonte: Adaptado de BASS (2003).

Bass (2003) define um padrão de arquitetura como uma descrição dos tipos de elementos, dos tipos de relacionamentos e de um conjunto de restrições que limitam a forma como esses elementos e relacionamentos podem ser usados. Considerando esta definição, a arquitetura cliente-servidor pode ser entendida como um padrão de arquitetura. Os dois tipos de elementos são o cliente e o servidor e o tipo de relacionamento é dado pelo protocolo de comunicação. Uma restrição deste padrão de arquitetura é que um elemento servidor pode se comunicar com diversos elementos clientes mas os elementos clientes não se comunicam entre si.

Esta terceira parte do capítulo 4 apresentou formas de se variar a representação da arquitetura de software através de decisões de design. As decisões de design recorrentes que impactam sobre um ou mais atributos de qualidade podem ser registradas na forma de táticas de arquitetura. As táticas de arquitetura mostram que arquiteturas de software diferentes podem implementar o mesmo conjunto de funcionalidades e diferirem por apresentarem atributos de qualidade diferentes.

Os dois capítulos 5 e 6 seguintes apresentam formas (abordagens) de se desenhar (projetar) a arquitetura de software. Nestas abordagens, o foco será a maneira pela qual o processo de design da arquitetura de software é conduzido ou guiado e como este processo auxilia ou restringe as possibilidades de se obter a qualidade de software desejada através da especificação da arquitetura de software.

Quatro abordagens foram consideradas. A primeira abordagem apresenta o paradigma clássico (ou estruturado) de desenvolvimento de software e é referida como abordagem clássica. A segunda considera o paradigma orientado a objetos e é referida como abordagem orientada a objetos. A terceira abordagem apresenta a técnica de design orientado a atributos ("Attribute Driven Design" ou ADD) e é referida como abordagem orientada a atributos e a quarta abordagem considera a aplicação de técnicas meta-heurísticas para se encontrar uma boa solução candidata e é referida como abordagem orientada a busca.

5 PROCESSO DE DESIGN DA ARQUITETURA DE SOFTWARE

Este capítulo apresenta o desenvolvimento da etapa 7 que conclui o terceiro objetivo específico de "Analisar as formas pelas quais a arquitetura de software pode ser especificada dada a qualidade de software desejada ". A ilustração 28 apresenta uma síntese do capítulo.

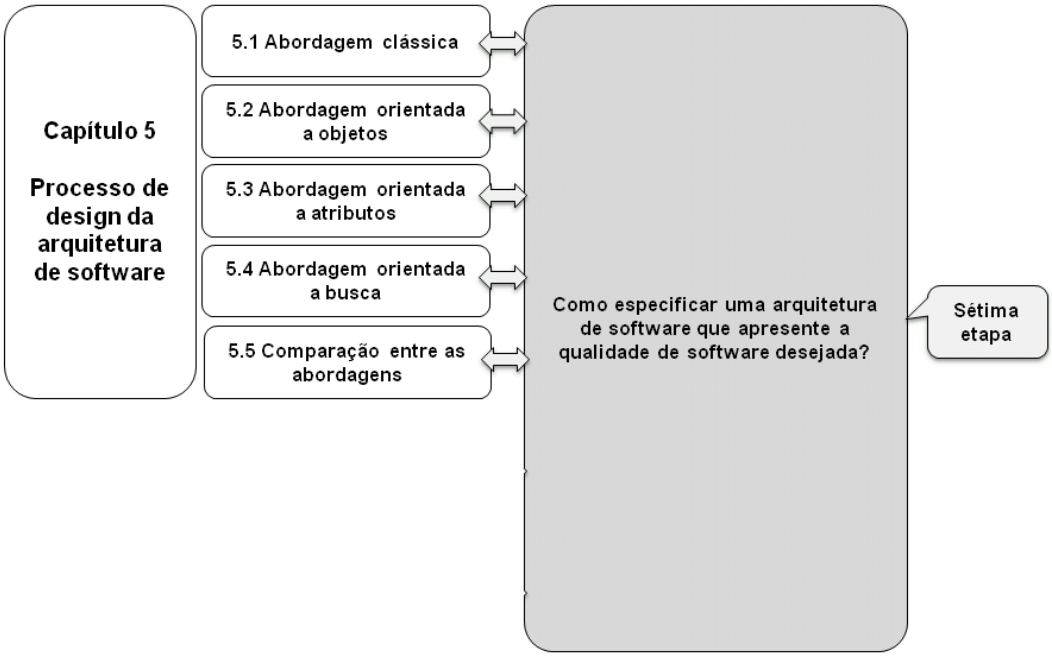

Ilustração 28 - Síntese do capítulo 5.

Segundo Jalote (2008, p. 121), a atividade de design começa quando os documentos de requerimentos para o software a ser desenvolvido estiverem disponíveis e a arquitetura estiver desenhada. O processo de design pode ser considerado como uma continuação ou refinamento do processo de design da arquitetura.

O processo de design de sistemas de software pode ser dividido em dois níveis. No primeiro nível o foco está em decidir quais os módulos necessários para o sistema, as especificações destes módulos e como os módulos devem ser interligados. Esse nível pode ser chamado de design do módulo ou design de alto nível.

No segundo nível, o foco está em como as especificações do módulo podem ser satisfeitas, ou em outras palavras, o foco passa a ser os detalhes internos dos módulos. Esse nível é chamado de design lógico ou design detalhado.

5.1 Abordagem clássica

Uma metodologia de design é uma abordagem sistemática para criar o design aplicando-se um conjunto de princípios e técnicas. A maioria das metodologias de design concentra-se no design de alto nível e não prescrevem uma sequência de passos bem definidas.[31] (JALOTE, 2008, p. 121 e 122).

Para avaliar o design é necessário especificar algum critério de avaliação. Um critério comum é o critério de modularidade de um sistema. Um sistema é considerado modular se os módulos discretos que o compõem puderem ser implementados separadamente e se uma mudança em um dos módulos causar o mínimo de impacto nos demais módulos.

Jalote (2008, p.123) considera que o projeto de um design modular que suporte uma abstração bem definida pode utilizar dois critérios e um princípio para alcançar uma modularidade alta. Os dois critérios são o acoplamento e a coesão e o princípio é o princípio do aberto-fechado[32].

Dois módulos são considerados independentes se um deles pode funcionar completamente sem a presença do outro. Portanto, quanto mais conexões entre os módulos mais dependentes eles são no sentido de que é necessário maior conhecimento de um módulo para se entender e projetar o outro módulo.

O acoplamento entre módulos é a força das interconexões entre os módulos. Em geral, quanto mais precisamos conhecer sobre o módulo A para compreender o módulo B, mais

---

[31] A design methodology is a systematic approach to creating a design by applying of a set of techniques and guidelines. Most design methodologies focus on module design, but do not reduce the design activity to a sequence of steps that can be blindly followed by the designer.
[32] Tradução livre de open-closed principle.

intimamente conectados estão A e B. Dessa forma, módulos com alto acoplamento são unidos por uma forte conexão e módulos com baixo acoplamento são unidos por uma fraca conexão.

A metodologia de design clássico ou estruturado para a construção do design de sistema orientado a função utiliza o gráfico de estruturas[33]. No gráfico de estruturas, um módulo é representado por uma caixa com o nome do módulo em seu interior. Uma flecha do módulo A para o módulo B significa que o módulo A faz uma chamada ao módulo B.

O gráfico de estruturas é uma representação de um design que usa uma abstração funcional. O gráfico apresenta os módulos e suas hierarquias de chamadas, as interfaces entre os módulos e quais as informações que transitam entre os módulos.

O objetivo da metodologia de design estruturado é determinar a estrutura do sistema durante a etapa de design de forma que um programa que a implemente tenha uma estrutura hierárquica com módulos coesos funcionalmente e com o menor número de conexões possíveis entre os módulos. A ilustração 29 apresenta um exemplo de gráfico de estruturas.

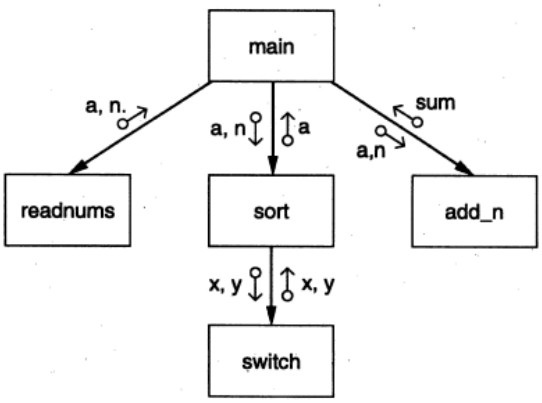

Ilustração 29 - Exemplo do gráfico de estruturas.
FONTE: JALOTE, 2008, p. 133.

O princípio básico da metodologia de design estruturado assim como diversas outras metodologias é o problema do particionamento. A metodologia de design estruturado particiona o sistema em vários subsistemas. Um para gerenciar cada uma das grandes

---

[33] Tradução livre de structure chart.

entradas, um para gerenciar cada uma das grandes saídas e um para gerenciar cada uma das grandes transformações de dados.

A lógica para esta forma de particionamento é que em muitos sistemas, em particular sistemas de processamento de dados, uma grande parte do código do sistema trata do gerenciamento das entradas e saídas. Os módulos que tratam das entradas precisam lidar com questões como o tipo de interface de entrada, o formato dos dados, a estrutura das informações, etc. De forma semelhante, os módulos que tratam das saídas precisam preparar a saída para um determinado formato de apresentação, produzir gráficos e relatórios, etc.

Essa forma de particionamento é o núcleo da metodologia de design estruturado e em termos gerais pode ser realizada seguindo-se quatro grandes passos:

. descreva o problema na forma de um diagrama de fluxo de dados.
. identifique os elementos de dados de entrada e saída.
. fatoração de primeiro nível.
. fatoração das três grandes áreas: entrada, saída e transformação.

Uma idéia central da abordagem clássica é que o diagrama de fluxo de dados é utilizado como base para decidir sobre as alocações de responsabilidades à funções. A notação deste diagrama facilita a associação entre seus elementos visuais e funções e estruturas de controle do código.

5.2 Abordagem orientada a objetos

O design orientado a objetos tem uma representação próxima dos elementos do domínio do problema o que facilita o entendimento e o projeto do design. Jalote (2008, p. 142) descreve que as abordagens orientadas a objetos são mais naturais e fornecem uma estrutura mais rica para se raciocinar e abstrair[34].

O design orientado a objetos consiste de uma especificação de todas as classes e objetos que existirão na implementação do sistema. Uma possível abordagem consiste na seguinte sequência de passos:

. identificar classes e relacionamentos entre elas.
. desenvolver o modelo dinâmico e utilizá-lo para definir operações sobre as classes.
. desenvolver o modelo funcional e utilizá-lo para definir operações sobre as classes.
. identificar classes internas e operações.
. otimizar e empacotar.

Identificar classes e relacionamentos entre elas

Esse passo requer a identificação dos tipos de objetos no domínio do problema, as estruturas entre as classes (herança e agregação), os atributos das diferentes classes, as associações entre as diferentes classes e os serviços que cada classe precisa fornecer para suportar o sistema. Basicamente, este passo procura desenhar um diagrama de classes inicial.

Para a identificação dos objetos, pode-se inicialmente observar o problema e sua descrição. Adicione uma entidade como um objeto se o sistema precisa memorizar algo a seu respeito, se o sistema precisa de algum serviço desta entidade para executar algum outro serviço, ou se a entidade possui múltiplos atributos.

---

[34] Object-oriented approaches are believed to be more natural and provide richer structures for thinking and abstraction.

Desenvolver o modelo dinâmico e utilizá-lo para definir operações sobre as classes.

O diagrama de classes obtido no passo anterior fornece um design inicial no nível do módulo. O design será remodelado pelas definições dos eventos do sistema. O modelo dinâmico procura especificar como o estado dos vários objetos muda de acordo com a ocorrência dos eventos. Cada evento tem uma entidade que o inicia e uma entidade que o atende. Eventos podem ser internos ao sistema no caso em que tanto a entidade que o inicia quanto a entidade que o atende são internos ao sistema ou podem ser externos no caso em que a entidade que o inicia for uma entidade externa como, por exemplo, um usuário ou um sensor.

Um cenário é uma sequência de eventos que ocorre em uma execução particular do sistema. A partir dos cenários, os diversos eventos que atingem os diferentes objetos podem ser identificados o que auxilia na identificação dos serviços dos objetos. Diferentes cenários podem caracterizar o comportamento do sistema. Se o design for elaborado de tal forma que atenda a todos os cenários, o comportamento dinâmico do sistema será atendido pelo design. Os diagramas de interação para os principais cenários são resultados da modelagem dinâmica e são conhecidos como casos de uso[35].

---

[35] Tradução livre para *use cases*.

5.3 Abordagem orientada a atributos

O processo de design da arquitetura de software orientada a atributos desenvolvida por Bass (2003) é denominada de design orientado a atributos (ADD - "Attribute driven design"). Os principais passos deste método são descritos a seguir:

1. Escolha um módulo para a decomposição estrutural. O módulo inicial usualmente considera todo o sistema. Todas as entradas requeridas pelo módulo devem estar disponíveis. As restrições, os requerimentos funcionais e os requerimentos não-funcionais.

2. Refinar o módulo de acordo com os seguintes passos:

2.1 Escolher os requerimentos principais de arquitetura a partir dos cenários de qualidade concretos e dos requerimentos funcionais. Esse passo determina o que é importante para esta decomposição.

2.2 Escolha um padrão de arquitetura que satisfaça os requerimentos principais de arquitetura. Crie ou selecione o padrão baseado nas táticas que podem ser utilizadas para alcançar estes requerimentos. Identifique os sub-módulos requeridos para implementar estas táticas.

2.3 Instancie (defina) o módulo e aloque as funcionalidades dos casos de uso e represente usando múltiplas visões de arquitetura.

2.4 Defina as interfaces dos sub-módulos. A decomposição fornece módulos e restrições sobre os tipos de interações entre os módulos. Documente esta informação no documento de interface de cada módulo.

2.5 Verifique e refine os casos de uso e os cenários de qualidade e torne-os restrições para os sub-módulos. Este passo verifica que nada importante foi esquecido e prepara os sub-módulos para decomposições adicionais ou implementação.

3. Repita os passos acima para cada módulo que necessite de decomposições adicionais.

Bass (2003) considera que "funcionalidade e atributos de qualidade são ortogonais"[36]. O significado desta consideração é que a funcionalidade não determina os atributos de qualidade de forma única. A funcionalidade de ordenar um grande banco de dados pode tornar difícil a obtenção de um valor alto para o atributo de qualidade performance mas decisões de arquitetura diferentes resultam em diferentes valores para este atributo de qualidade.

Em outras palavras, uma dada funcionalidade pode ser obtida através da escolha de uma dentre as várias possibilidades de estruturas. Uma possível estrutura é alocar toda a funcionalidade para um único módulo. Outra possibilidade é decompor a funcionalidade e alocar as partes para módulos distintos. Desta forma, a mesma funcionalidade é atendida nos dois casos mas no segundo caso, a facilidade para distribuir partes da funcionalidade para grupos de desenvolvimento distintos é maior. Se o tempo para o lançamento do produto no mercado[37] for considerado como um atributo de qualidade de software sob a perspectiva de negócios, a segunda estrutura deve possibilitar obter um valor maior para este atributo.

A determinação dos atributos de qualidade depende das decisões tomadas em cada etapa do desenvolvimento de software passando pelo design, implementação e implantação. Tanto a arquitetura esboçada na etapa de design quanto a codificação realizada na etapa de implementação contribuem para a definição final dos atributos de qualidade do sistema.

O atributo de qualidade capacidade de modificação pode ser influenciado por aspectos de arquitetura como a alocação de funcionalidades para os módulos e por aspectos que não se referem à arquitetura como o estilo de codificação utilizado para implementar um dos módulos. Uma requisição de modificação é mais ou menos fácil de ser atendida dependendo do número de módulos envolvidos, e o número de módulos envolvidos depende da forma como as funcionalidades foram distribuídas entre eles. O estilo de codificação também torna o sistema mais ou menos fácil de ser modificado pois influencia a facilidade com que o código pode ser compreendido.

---

[36] "Functionality and quality attributes are orthogonal."
[37] Time-to-market

O atributo de qualidade performance depende de vários aspectos de arquitetura: a forma como os componentes se comunicam, a distribuição das funcionalidades entre os componentes e a forma como recursos compartilhados são acessados. Aspectos que não se referem à arquitetura também influenciam a performance como por exemplo a escolha do algoritmo para implementar uma dada funcionalidade e a forma como esse algoritmo é codificado.

Segundo Bass (2003), existem diversas classificações e definições para os atributos de qualidade e várias comunidades de pesquisadores e de praticantes que as adotam. No entanto, três problemas estão presentes em todas elas:

"As definições fornecidas para um atributo não são operacionais. Não tem sentido dizer que um sistema é modificável. Todo sistema é modificável com relação a um conjunto de mudanças e não modificável com relação a outro conjunto."
"Um foco de discussão freqüente é a respeito de qual qualidade um aspecto particular pertence. Uma falha do sistema é um aspecto da disponibilidade, um aspecto da segurança, ou um aspecto da usabilidade?"

"Cada comunidade desenvolveu um vocabulário próprio. A comunidade que trata da performance fala sobre 'eventos' chegando ao sistema, a comunidade que trata da segurança fala sobre 'ataques' chegando ao sistema, a comunidade que trata da disponibilidade fala sobre 'falhas' do sistema, e a comunidade que trata da usabilidade fala sobre 'entrada de usuário'. Todas as comunidades podem estar se referindo à uma mesma ocorrência mas usando termos diferentes para descrevê-la."

Essas questões podem ser tratadas através da utilização de cenários. Os cenários de atributos de qualidade são compostos por seis partes:

. Fonte do estímulo: refere-se à uma entidade que gera o estímulo.
. Estímulo: é uma condição (ou um evento) que precisa ser considerado (tratado).
. Ambiente: caracteriza as circunstâncias (condições) sob as quais o estímulo ocorre.
. Artefato: refere-se à parte do sistema que é estimulada.
. Resposta: refere-se à atividade executada devido à ocorrência de um estímulo.
. Medida da resposta: refere-se ao valor da mensuração da resposta.

A ilustração 30 apresenta as partes que compõem um cenário.

Ilustração 30 - Cenário de atributo de qualidade - estrutura.

Os cenários podem ser classificados em duas categorias: geral e concreto. Os cenários de atributos de qualidade gerais são aqueles que independem do sistema, ou seja, podem ser aplicados a qualquer subsistema. Os cenários de atributos de qualidade concretos são aqueles que dependem de um sistema em particular, ou seja, consideram apenas o sistema sob consideração.

Os cenários gerais procuram caracterizar os atributos de qualidade enquanto que os cenários concretos procuram especificar os requerimentos de atributos de qualidade de um determinado sistema. A forma pela qual os cenários concretos especificam os requerimentos de atributos de qualidade é através da particularização dos cenários gerais para o caso do sistema considerado.

Para ilustrar essa relação entre o cenário geral e o cenário concreto considere o seguinte cenário geral: uma requisição de mudança de funcionalidade é feita, ela deve ser atendida durante o processo de desenvolvimento e respeitar um prazo determinado. Um exemplo de particularização deste cenário geral para um cenário concreto pode ser: uma requisição para adicionar suporte a um novo navegador para um sistema baseado na web é feita e deve ser atendida em duas semanas.

Bass (2003) apresenta um exemplo de cenário concreto e cenário geral para o atributo de qualidade capacidade de modificação. Para este atributo de qualidade duas questões precisam ser consideradas:

. O que deverá (poderá) ser modificado? Ou em outras palavras o que é (pode ser) o artefato? Entre diversos aspectos do sistema que podem ser modificados podemos listar: a

funcionalidade (código), a plataforma (hardware e sistema operacional), o ambiente (outros sistemas com os quais o sistema precisa interagir e os protocolos utilizados para estas interações) e atributos de qualidade (performance, confiabilidade, etc). O caso em que o artefato é a plataforma é um caso comum e é também referido como portabilidade.

. Quando a mudança deverá ser realizada e quem é responsável por ela? Talvez o caso mais comum seja a mudança feita na etapa de codificação realizada pelo desenvolvedor mas outras alternativas são possíveis. A mudança da interface com o usuário durante o uso do sistema feito pelo próprio usuário é outra opção. Estendendo-se as possibilidades podemos visualizar mudanças ocorrendo em tempo de implementação, compilação, construção ("*build*"), configuração ou execução. Da mesma forma, podemos listar o desenvolvedor, o usuário e o administrador de sistemas como opções para a questão sobre quem é o responsável pela mudança.

As respostas dessas questões auxiliam a compreensão de um processo para a geração de cenários gerais para o atributo de qualidade capacidade de modificação. Para se gerar um cenário geral pode-se escolher uma das opções para cada uma das partes que compõem o cenário. O quadro 3 apresenta as seis partes e suas respectivas opções.

Quadro 3 - Descrição das partes de um cenário.

Partes do cenário	Opções possíveis
Fonte	Desenvolvedor, administrador de sistema e usuário final.
Estímulo	Deseja adicionar, remover ou modificar uma funcionalidade ou atributo de qualidade.
Artefato	Funcionalidade (código), plataforma, ambiente e atributo de qualidade.
Ambiente	Em tempo de implementação, compilação, construção, configuração e execução.
Resposta	Consiste em realizar e testar a modificação sem alterar outras funcionalidades.
Medida da resposta	Consiste em medir o custo em termos de tempo, valor monetário, número de elementos afetados, etc.

Para o caso do atributo de qualidade capacidade de modificação, o conceito das partes que compõem o cenário geral pode assumir os seguintes significados:

. Fonte do estímulo: determina quem realiza a mudança. Exemplos de opções são o desenvolvedor, o administrador de sistema e o usuário final.

. Estímulo: especifica a mudança a ser realizada. A mudança pode incidir sobre a funcionalidade ou sobre um atributo de qualidade. A mudança pode ser a adição, modificação ou remoção de uma funcionalidade; ou uma alteração em um dos atributos de qualidade. Um exemplo deste último caso é a necessidade de aumentar a capacidade de número de usuários simultâneos do sistema.

. Artefato: especifica o que deve ser alterado para que a mudança possa ser realizada. Exemplos de opções são: uma funcionalidade (código), um atributo de qualidade, a plataforma, um elemento do ambiente ou a interface de usuário.

. Ambiente: especifica quando a mudança deve ser realizada. Exemplos de opções são: em tempo de implementação, compilação, construção, configuração ou execução.

. Resposta: é a realização da mudança. Pode incluir considerações adicionais como a necessidade da mudança ser realizada sem a introdução de efeitos colaterais.

. Medida de resposta: é o valor da mensuração de um aspecto da resposta. Os dois aspectos mais comuns são o tempo e o custo. A mensuração pode ser feita de forma indireta, por exemplo: se o tempo for um aspecto difícil de ser mensurado, uma alternativa poderia ser sua substituição pelo número de módulos afetados pela mudança.

Um cenário concreto pode ser construído com o auxílio do cenário geral. A construção é feita selecionando uma das possíveis opções para as partes do cenário geral e particularizando as opções com considerações adicionais do sistema em questão. O exemplo a seguir descreve um cenário concreto e a respectiva ilustração 31.

“Um desenvolvedor deseja mudar a interface de usuário. A mudança será feita no código durante a etapa de implementação (codificação). Esta alteração deverá ocorrer em menos de três horas sem a introdução de efeitos colaterais.”

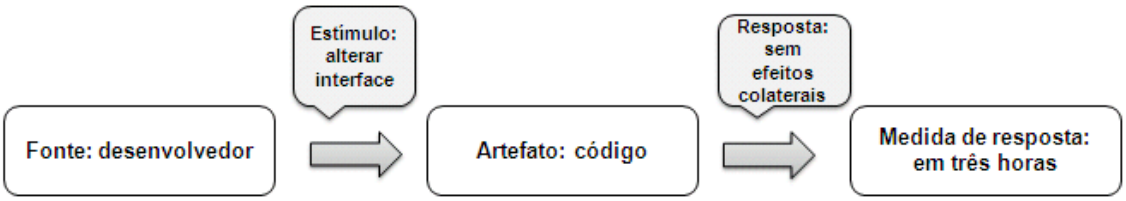

Ilustração 31 - Exemplo de cenário para o atributo de qualidade capacidade de modificação.

Relação dos atributos de qualidade com as necessidades de negócio

As necessidades de negócio são especificadas principalmente baseadas em considerações de mercado. Essas considerações determinam as qualidades de negócio que o sistema deve apresentar. Qualidades freqüentemente desejadas como um rápido tempo de chegada ao mercado ("*time to market*"), uma boa relação custo-benefício, um alto tempo de vida útil e uma boa capacidade de integração com sistemas legados podem ser atendidas através dos atributos de qualidade do sistema. Em outras palavras, a arquitetura de software define em grande parte os atributos de qualidade de software que por sua vez define as qualidades sob o ponto de vista do negócio que o sistema deve apresentar. Bass (2003) apresenta alguns exemplos que ilustram essa relação:

. "*Time to market*": se um sistema ou produto tiver uma pequena janela de oportunidade, o tempo de desenvolvimento se torna um fator importante. Essa situação pode pressionar a favor de uma decisão que compre ou reutilize componentes para a construção do sistema. E a capacidade ou facilidade que um sistema tem de receber ou fornecer componentes depende da decomposição do sistema em subsistemas (módulos).

. Relação custo-benefício: uma arquitetura cuja implementação dependa ou se baseie em uma determinada tecnologia que não é familiar à equipe de desenvolvimento tem um custo maior que uma arquitetura que possibilite a utilização de tecnologias conhecidas. Uma arquitetura flexível geralmente tem um custo maior que uma arquitetura mais rígida mas apresenta a vantagem de ser modificada a um custo e tempo menor.

. Tempo de vida útil: se o tempo de vida útil for uma característica importante do sistema ou produto, os atributos de qualidade capacidade de modificação, escalabilidade e portabilidade

se tornam atributos desejáveis. As necessidades de negócio podem ser conflitantes entre si: a necessidade de se obter um sistema com um tempo de vida útil maior pode requerer uma capacidade de modificação maior o que pode afetar negativamente o "*time to market*".

. Capacidade de integração com sistemas legados: a necessidade de integração com sistemas legados influencia a forma como a arquitetura pode ser decomposta em módulos e os tipos de comunicação que devem ser suportados. Essa influência tem o efeito de restringir o universo de arquiteturas possíveis que por sua vez pode impactar a possibilidade de se atender diversos atributos de qualidade do sistema.

5.4 Abordagem orientada a busca

A abordagem orientada a busca deste estudo refere-se à aplicação de técnicas meta-heurísticas para executar o processo de design da arquitetura de software. Para se aplicar uma das diversas técnicas meta-heurísticas é necessário reformular o problema de design em um problema de busca.

O termo Engenharia de software baseada em busca ("Search Based Software Engineering" ou SBSE) foi criado por Mark Harman em 2001 embora existam trabalhos que apliquem técnicas de busca em problemas de engenharia de software desde 1992. (HARMAN, 2007, p. 342).

Harman (2007) identificou oito áreas de aplicação de técnicas de otimização em problemas de engenharia de software:

- otimizar a busca por uma estimativa de custo precisa.
- otimizar a busca para alocações em um planejamento de projetos.
- otimizar a busca por requerimentos para formar a próxima versão.
- otimizar decisões de design.
- otimizar código-fonte.
- otimizar a geração de dados de teste.
- otimizar a seleção de dados de teste e priorização.
- otimizar manutenção e engenharia reversa.

Em contraposição às abordagens qualitativas descritas anteriormente que procuram guiar o processo de design através das aplicações de princípios e técnicas, a abordagem quantitativa propõe a utilização de um modelo matemático e a aplicação de uma técnica de busca para a identificação do melhor design do sistema.

Um ponto positivo desta abordagem é a possibilidade de se obter uma solução através de um procedimento automático ou semi-automático para problemas com um espaço de soluções muito grande e com objetivos que competem entre si.

A SBSE basicamente procura reformular os problemas da engenharia de software como problemas de otimização baseado em busca. Entre as diversas técnicas de busca existentes as mais utilizadas são a busca local, a maleabilização simulada, o algoritmo genético e a programação genética. (HARMAN, 2009, p.1).

Existem dois pontos fundamentais para a aplicação de uma otimização baseada em busca para os problemas da engenharia de software:

. a escolha da representação do problema.
. a definição da função objetivo.

Os algoritmos utilizados utilizam diferentes abordagens para localizar uma solução ótima ou uma solução próxima da solução ótima mas realizam essencialmente uma busca através das inúmeras soluções candidatas guiadas pela função objetivo de forma que o algoritmo possa comparar as soluções e identificar a melhor.

Os algoritmos genéticos utilizam o conceito de população e recombinação. De todos os algoritmos de otimização, os algoritmos genéticos são os mais aplicados na SBSE embora isto possa ser decorrência de motivos históricos e não por razões científicas.

Para o estudo desta abordagem, uma aplicação é apresentada no capítulo 6. Esta aplicação demonstra como o atributo de qualidade capacidade de modificação de um sistema de software pode ser otimizada através da busca de uma boa solução em um espaço de soluções utilizando a técnica de algoritmos genéticos.

5.5 Comparação entre as abordagens

Considerando as quatro abordagens apresentadas neste capítulo, três características se destacam como relevantes para diferenciar a maneira pela qual cada abordagem conduz o processo de design da arquitetura de software. As três características são:
- o critério base para a decomposição estrutural.
- a visão da arquitetura de software privilegiada.
- o momento de aplicação de conhecimentos de arquitetura privilegiado.

A abordagem clássica orienta que a decomposição estrutural seja feita em torno de três grandes módulos: entrada (E), processamento (P) e saída (S). Estes três grandes módulos podem ser decompostos em módulos menores observando-se o processamento necessário sobre o fluxo de dados. Este critério de decomposição pode ser visto como um critério semântico. A principal visão da arquitetura de software é o diagrama de fluxo de dados (DFD) e o momento favorecido de aplicação de conhecimentos de arquitetura é após as primeiras decomposições.

A abordagem orientada a objetos conduz a uma decomposição estrutural semelhante. Um módulo composto por classes responsáveis pela interação com as entidades externas, ou seja, classes responsáveis por processos de entrada (E) e saída (S). Um módulo composto por classes que gerenciam informações persistentes (I). E um módulo composto por classes responsáveis pela implementação dos procedimentos e algoritmos (P). A visão estrutural na forma do diagrama de classes da UML recebe um destaque maior e o momento favorecido para a aplicação de conhecimentos de arquitetura é após a primeira decomposição.

A abordagem orientada a atributos adota como critério para a decomposição estrutural os atributos de qualidade. Todas as visões da arquitetura de software recebem a mesma prioridade pela abordagem. Uma ou outra visão da arquitetura pode receber um destaque maior devido à necessidade de se compreender melhor como otimizar um ou outro atributo de qualidade de um dado sistema. A aplicação de conhecimentos de arquitetura ocorre desde a primeira decomposição.

A abordagem orientada a busca permite a escolha de um atributo de qualidade para ser otimizado. A visão da arquitetura de software que recebe destaque depende do atributo de qualidade escolhido. E a aplicação de conhecimentos de arquitetura pode ser incorporada no algoritmo de busca utilizado.

O quadro 4 resume a forma pela qual cada uma das quatro abordagens lida com as três características observadas que auxiliam a caracterizar o processo de design da arquitetura de software.

Quadro 4 - O processo de design da arquitetura de software - três características.

Abordagem	**Clássica**	**Orientada a objetos**	**Orientada a atributos**	**Orientada a busca**
critério base para a decomposição estrutural	E /P / S Semântica	E / S / P / I Semântica	Atributos de qualidade	Atributos de qualidade
visão da arquitetura de software privilegiada	Diagrama de fluxo de dados	Diagrama de classes	Depende do atributo de qualidade	Depende do atributo de qualidade
o momento de aplicação de conhecimentos de arquitetura privilegiado	Após as primeiras decomposições	Após a primeira decomposição	Desde a primeira decomposição	Aplicado se incorporado ao algoritmo de busca

6 APLICAÇÃO DA ABORDAGEM ORIENTADA A BUSCA

Este capítulo apresenta o desenvolvimento da etapa 7 que conclui o terceiro objetivo específico de "Analisar as formas pelas quais a arquitetura de software pode ser especificada dada a qualidade de software desejada ". A ilustração 32 apresenta uma síntese do capítulo.

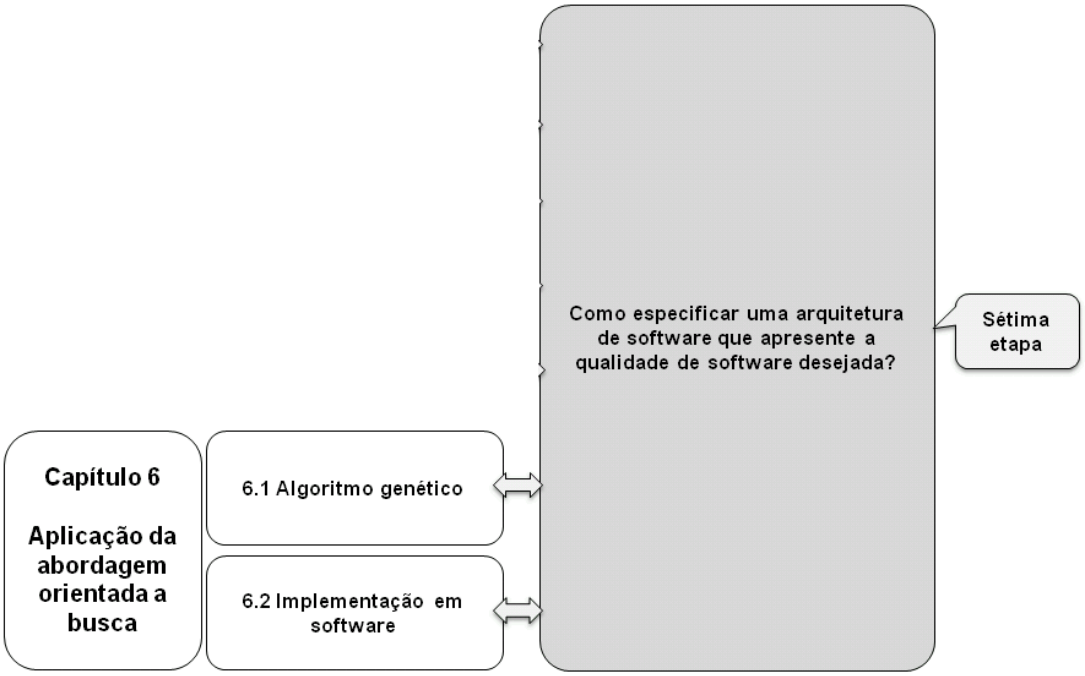

Ilustração 32 - Síntese do capítulo 6.

A abordagem orientada a busca para o processo de design da arquitetura de software é a única abordagem quantitativa entre as quatro abordagens consideradas. Este capítulo procura avançar o entendimento desta abordagem através da exposição de uma aplicação.

Conforme descrito no capítulo 3, o atributo de qualidade que se procura otimizar nesta aplicação é a capacidade de modificação do software. E conforme descrito no capítulo 4, a métrica de software associada ao atributo de qualidade capacidade de modificação é a qualidade de modularização ou TurboMQ.

A fórmula desta métrica (TurboMQ) é, portanto, a função objetivo a ser otimizada pela técnica de busca meta-heurística denominada algoritmo genético.

6.1 Algoritmo genético

O exemplo focaliza a arquitetura de software sob a perspectiva estrutural. A questão abordada sob esta perspectiva é a etapa de atribuição de tarefas (ou responsabilidades) aos subsistemas (ou módulos) do sistema.

A abordagem utilizada para atacar esta questão é uma abordagem quantitativa. O método metaheurístico utilizado é o algorítmo genético e o exemplo é descrito de forma genérica e não particularizado para um tipo de sistema específico.

O software utilizado para implementar o algorítmo genético é o Evolver versão 5.5 da Palisade. O Evolver foi desenvolvido na forma de um Add-in para o Excel da Microsoft. As ilustrações deste capítulo que contém as telas do Evolver foram capturadas da versão 2007 do Excel.

O Evolver usa uma abordagem ou técnica denominada de estado estacionário ("*steady-state*") ao invés da abordagem de troca de gerações ("*generational replacement*"). Desta forma, apenas um organismo é trocado de cada vez.

A operação de seleção escolhe dois pais contidos na população corrente. Os organismos cujo valor da função objetivo é maior têm maior probabilidade de serem escolhidos como pais. Esta escolha é baseada em um mecanismo de ranking ao invés de considerar a probabilidade de escolha proporcional ao valor da função objetivo. Este mecanismo procura evitar que os melhores organismos dominem completamente o processo evolutivo prematuramente.

O Evolver possui um conjunto de métodos de resolução ("*solving methods*") que ajustam as variáveis de diferentes maneiras. Diferentes métodos de resolução podem utilizar uma rotina para a operação de *crossover* otimizada para um determinado tipo de problema.

O método de resolução receita ("*recipe solving method*") executa a operação de *crossover* usando uma rotina de *crossover* uniforme. Desta forma, cada variável da lista de variáveis em um dado cenário pode receber os cromossomos de qualquer um dos pais. No caso do *crossover* do tipo ponto simples ("*single-point*") e ponto duplo ("*double-point*"), um intervalo de variáveis recebe os cromossomos de um dos pais. A rotina de *crossover* uniforme permite

gerar qualquer esquema a partir dos dois pais e evita criar um viés através da escolha de um ou dois pontos de corte.

A operação de mutação para o método de resolução receita gera um número aleatório entre zero e um para cada uma das variáveis do organismo. Caso o número gerado seja maior que a taxa de mutação, a variável sofre mutação. O novo valor da variável é gerado aleatoriamente respeitando a faixa de valores válidos.

O novo organismo criado a partir das operações de seleção, *crossover* e mutação , sempre substitui o organismo com a pior classificação de acordo com o mecanismo de ranking. O valor da função objetivo deste organismo não é levado em consideração para se realizar uma escolha probabilística do organismo a ser substituído.

6.2 Implementação em software

O exemplo contempla um sistema que implementa 12 responsabilidades representadas pelos ícones ovais numerados de R1 a R12. As setas azuis representam os relacionamentos entre as responsabilidades com peso (ou intensidade) 1. As setas verdes representam os relacionamentos com peso 2.

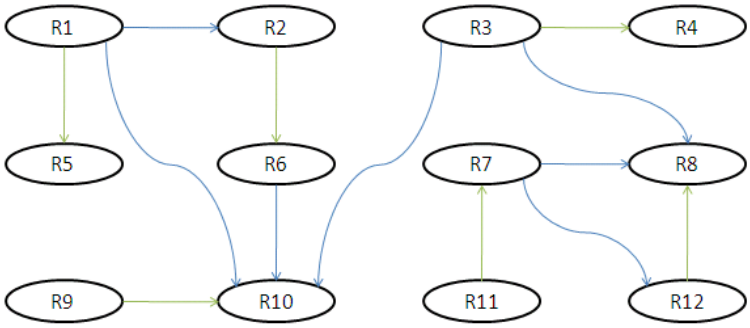

Ilustração 33 - Representação gráfica das responsabilidade e relacionamentos do sistema.

A partir da representação das responsabilidades e dos relacionamentos entre elas, deseja-se propor uma divisão das responsabilidades em subsistemas de forma que a qualidade de modularização seja a máxima possível. Esse particionamento representa uma perspectiva estrutural da arquitetura de software do sistema.

Para utilizar o Evolver é necessário representar o problema no Excel. A planilha apresentada a seguir foi elaborada para registrar as informações necessárias para a criação do modelo e para apresentar a saída do relatório criado através de uma macro do Excel desenvolvida para este exemplo.

	A	B	C	D	E	F	G	H
1	Qualidade de modularização:	1,89		Apresentar relatório			1,89	
2	Fatores de cluster	0,50	0,55	0,40	0,44			
3	relac. intra	2,00	3,00	2,00	2,00			
4	relac. extra	4,00	5,00	6,00	5,00			
5								
6	subsistema	tarefa	matriz de dependência					
7	1	1	2,01	5,02	10,01			
8	2	2	6,02					
9	3	3	4,02	8,01	10,01			
10	4	4						
11	1	5						
12	2	6	10,01					
13	3	7	8,01	12,01				
14	4	8						
15	1	9	10,02					
16	2	10						
17	3	11	7,02					
18	4	12	8,02					
19								
20								

Ilustração 34 - Planilha Excel inicial.

A célula B2 contém o valor da qualidade de modularização dado através da fórmula: =Calcular_cluster_factor(subsistema_indice; matriz_dep)

A fórmula Calcular_cluster_factor foi escrita na linguagem VBA no editor Microsoft Visual Basic. O código fonte da fórmula é apresentada no apêndice 1.

O primeiro parâmetro subsistema_indice (índice do subsistema) se refere às celulas A7 à A18. Essas células devem conter um número natural inteiro positivo que identifique o subsistema associado a cada uma das responsabilidades (ou tarefas) apresentadas nas células de B7 à B18.

Portanto, para a situação apresentada na planilha inicial a responsabilidade 1 é implementada no subsistema 1, a responsabilidade 2 é implementada no subsistema 2, ..., e a responsabilidade 12 é implementada no subsistema 4.

O segundo parâmetro matriz_dep (matriz de dependência) refere-se às celulas C7 à I18. Essas células devem conter um número com duas casas decimais que identifique o relacionamento e o respectivo peso entre duas responsabilidades.

O relacionamento é identificado pela parte inteira do número e o peso pelas duas casas decimais.

Para cada responsabilidade listada nas células de B7 à B18 existe uma linha correspondente na matriz de dependência. Para cada linha, cada célula identifica uma responsabilidade da qual a responsabilidade correspondente na coluna B depende.

A ilustração 34 da planilha inicial mostra que a responsabilidade 1 posicionada na célula B7 depende das responsabilidades 2, 5 e 10 posicionadas nas células C7, D7 e E7 com pesos de 1, 2 e 1 respectivamente. Da mesma forma, a responsabilidade 7 posicionada na célula B13 depende das responsabilidades 8 e 12 posicionadas nas células C13 e D13 respectivamente, ambas com peso 1.

A partir desta planilha inicial, a definição do modelo pode ser realizada na caixa de diálogo "Evolver – Model" que pode ser acessada clicando no botão "Model Definition" conforme ilustrado na ilustração 35.

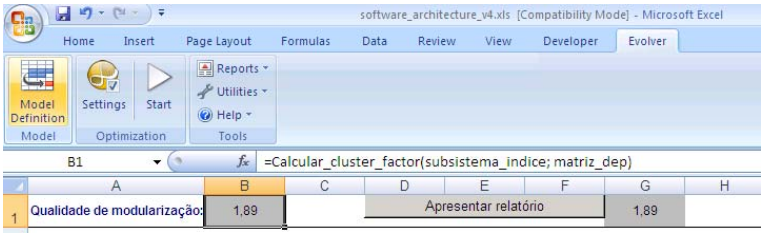

Ilustração 35 - Menu da aba do Evolver.

Na caixa de diálogo "*Evolver – Model*", o seguinte modelo foi representado:

. Campo "*Optimization Goal*" definido como *Maximum*. Deseja-se maximizar a qualidade de modularização da célula B1.

. Campo "*Cell*" definido em B1. Este campo define a célula cujo valor deseja-se maximizar.

. Área "*Adjustable Cell Ranges*" define o intervalo de células que o Evolver ajustará para maximizar o valor da célula indicada no campo "*Cell*". Neste caso, como quatro subsistemas são considerados, o campo "*Minimum*" e "*Maximum*" foram definidos em 1 e 4

respectivamente e o campo "*Values*" foi definido em *Integer* para que os subsistemas possam ser identificados por números inteiros variando de 1 a 4.

. A área "*Adjustable Cell Ranges*" mostra o grupo que contém o intervalo de A7 à A18 associado ao método de resolução receita.

A ilustração 36 apresenta a caixa de diálogo com os valores preenchidos.

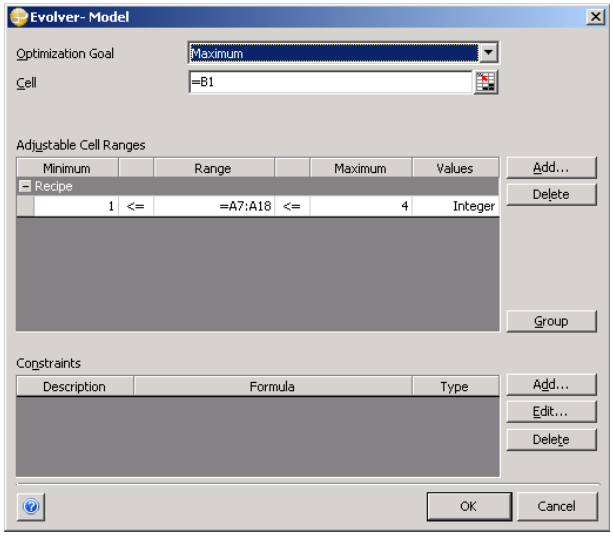

Ilustração 36 - Caixa de diálogo para a definição do modelo do Evolver.

Através do botão *Group* da ilustração 36, pode-se acessar a caixa de diálogo "*Evolver – Adjustable Cell Group Settings*" que permite configurar os parâmetros de otimização *Crossover Rate* e *Mutation Rate*. Os valores padrões de 0,5 e 0,1 foram mantidos. O método de resolução do grupo é definido no campo "*Solving Method*". Neste caso, o método *Recipe* foi selecionado. A ilustração 37 apresenta esta caixa de diálogo preenchida.

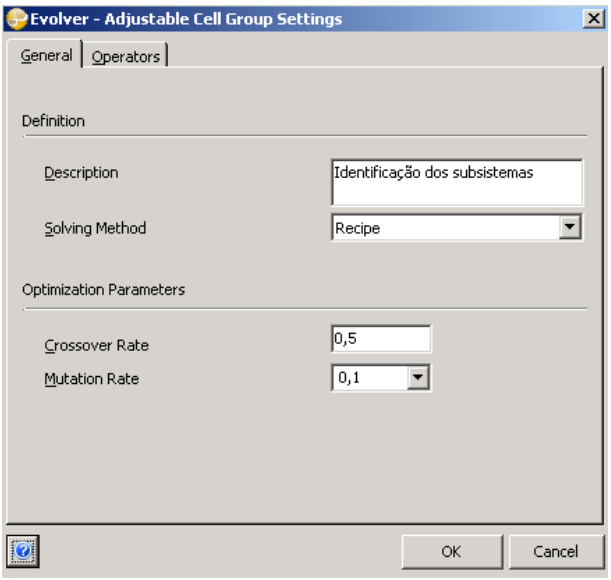

Ilustração 37 - Caixa de diálogo das configurações do grupo de células ajustáveis.

A configuração do modelo pode ser realizada na caixa de diálogo "*Evolver – Optimization Settings*" que pode ser acessada pressionando o botão "*Settings*" conforme ilustrado na ilustração 38 "Menu da aba do Evolver". Nesta caixa de diálogo é definido o tamanho da população no campo "*Population Size*" contido na aba *General*. Para o exemplo, o tamanho da população foi mantido com seu valor padrão de 50. A ilustração 38 apresenta a caixa de diálogo com o valor preenchido.

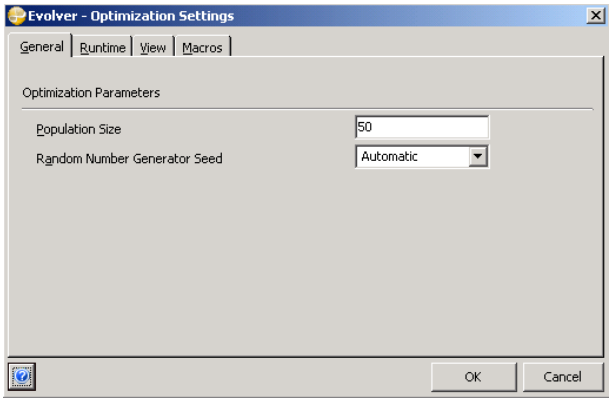

Ilustração 38 - Caixa de diálogo das configurações de otimização.

O botão "Apresentar relatório" executa uma macro do Excel ao ser pressionado. Esta macro foi desenvolvida para apresentar as informações das linhas 2, 3 e 4. As informações contidas nestas linhas são o fator de cluster, o relacionamento interno e o relacionamento externo de cada subsistema. Estas três informações são identificadas pelos textos "Fatores de cluster", "relac. intra" e "relac. extra" respectivamente. Desta forma, cada subsistema tem suas informações apresentadas em uma coluna.

As informações do primeiro subsistema ocupam a coluna B, as informações do segundo subsistema ocupam a coluna C, e assim por diante. Na situação apresentada na ilustração 34 "Planilha Excel inicial", o primeiro subsistema tem um fator de cluster de 0,50, um valor de relacionamento interno de 2,00, e um valor de relacionamento externo de 4,00. Adicionalmente, a macro também calcula o valor da qualidade de modularização e apresenta o resultado na célula G1.

Para iniciar o processo de otimização, o botão "*Start*" deve ser pressionado. Durante a execução do processo de otimização, a janela "*Evolver – Progress*" é apresentada exibindo as seguintes informações: o número de tentativas realizadas, o tempo de execução, o valor original (inicial) da função objetivo e o melhor valor da função objetivo obtido até o momento. A ilustração 39 apresenta a janela "*Evolver – Progress*".

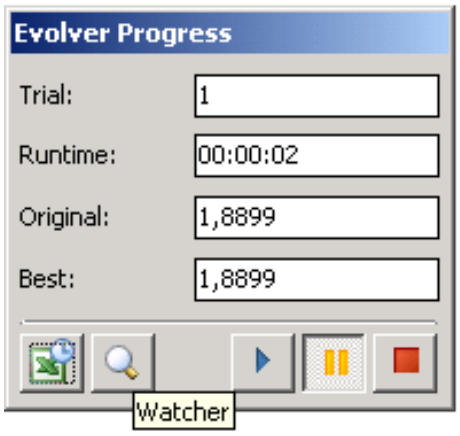

Ilustração 39 - Caixa de diálogo do progresso do Evolver.

Durante a execução do processo de otimização, pode-se pressionar o botão *Watcher* da janela "*Evolver – Progress*". Desta forma, a janela "*Evolver – Watcher*" será apresentada. A aba *Diversity* apresenta a diversidade da população. O eixo vertical apresenta os organismos (possíveis soluções) e o eixo horizontal apresenta os cromossomos destes organismos (o conjunto de células ajustáveis de cada solução).

A ilustração 40 apresenta a diversidade de uma população composta por 50 organismos. Cada organismo é representado por 12 células ajustáveis cujo valor pode assumir um dos quatro valores possíveis (quatro subsistemas possíveis). Esses valores são representados através de cores, neste caso, as quatro cores utilizadas foram: preto, roxo, verde e branco.

Ilustração 40 - Caixa de diálogo da diversidade da população - 50 organismos.

Finalmente, efetuando-se a otimização, o resultado obtido após 1000 iterações (tentativas) é o valor de 2,87 para a qualidade de modularização e a atribuição das responsabilidades 1 e 5 para o subsistema 1; das responsabilidades 2, 6, 9 e 10 para o subsistema 2; das responsabilidades 3, 4, 8 e 12 para o subsistema 3; e das responsabilidades 7 e 11 para o subsistema 4.

A ilustração 41 apresenta a planilha Excel com os valores atualizados. O botão "Apresentar relatório" foi pressionado após a otimização para atualizar as informações do relatório.

	A	B	C	D	E	F	G
1	Qualidade de modularização:	2,87		Apresentar relatório			2,87
2	Fatores de cluster	0,67	0,77	0,67	0,77		
3	relac. intra	2,00	5,00	2,00	5,00		
4	relac. extra	2,00	3,00	2,00	3,00		
5							
6	subsistema	tarefa	matriz de dependência				
7	1	1	2,01	5,02	10,01		
8	2	2	6,02				
9	4	3	4,02	8,01	10,01		
10	4	4					
11	1	5					
12	2	6	10,01				
13	3	7	8,01	12,01			
14	4	8					
15	2	9	10,02				
16	2	10					
17	3	11	7,02				
18	4	12	8,02				

Ilustração 41 - Resultado para 50 organismos e 4 subsistemas.

Para facilitar a visualização das responsabilidades atribuídas a cada subsistema, as ilustrações a seguir utilizam o esquema de cores abaixo:

Quadro 5 - Esquema de cores dos subsistemas.

Cor	Subsistema
Laranja	1
Verde	2
Azul	3
Roxo	4

A ilustração 42 apresenta a distribuição das responsabilidades entre os subsistemas antes da otimização. O valor da qualidade de modularização é 1,89.

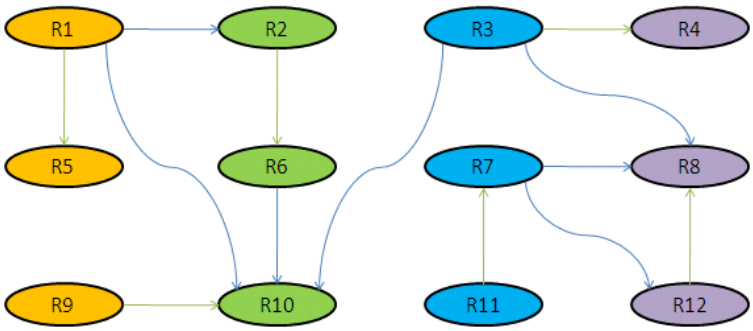

Ilustração 42 - MDG inicial das 12 responsabilidades.

Ilustração 43 apresentando a distribuição das responsabilidades entre os subsistemas depois da otimização. O valor da qualidade de modularização é 2,87.

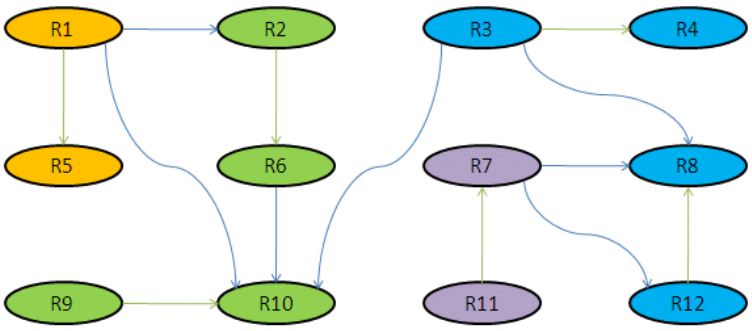

Ilustração 43 - MDG final para 4 subsistemas.

A janela "*Evolver – Watcher*", aba *Progress*, apresenta a evolução do valor da função objetivo (qualidade de modularização) ao longo das 1000 iterações.

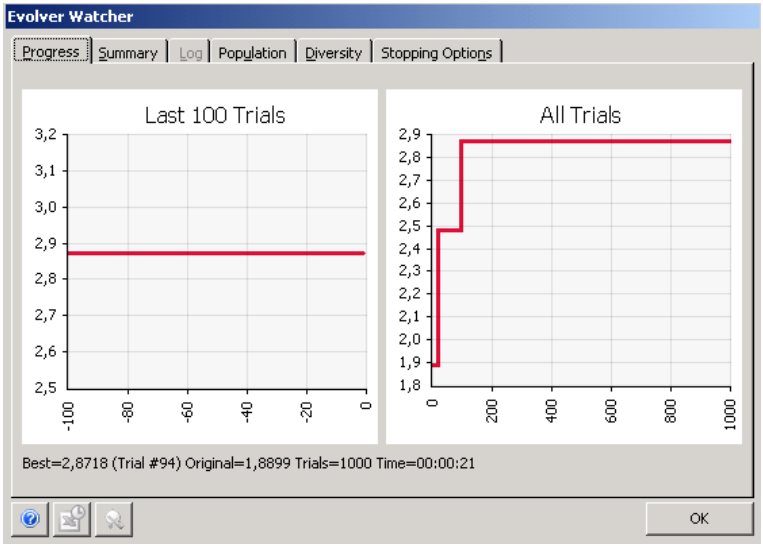

Ilustração 44 - Caixa de diálogo de monitoramento - 50 organismos.

A janela "*Evolver – Watcher*", aba *Diversity*, mostra que ao final das 1000 iterações, o conjunto dos 50 organismos, convergiu para um único tipo de organismo.

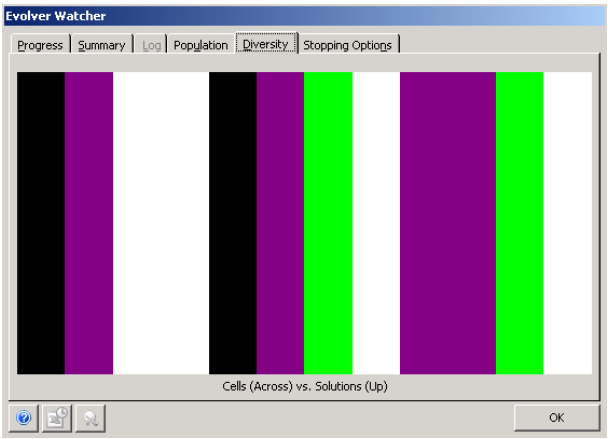

Ilustração 45 - Diversidade da população - convergência para 50 organismos.

Experimentando o processo de otimização algumas vezes variando-se o método de resolução, o tamanho da população inicial e a taxa de mutação, o melhor resultado obtido para o valor da qualidade de modularização foi de 3,03.

Neste caso, foi utilizado o método de resolução por agrupamento ("*Grouping*"), uma população inicial de 300 e uma taxa de mutação de 0,2. Os demais parâmetros foram os mesmos do experimento descrito anteriormente.

A ilustração 46 apresenta a planilha Excel com os valores atualizados.

	A	B	C	D	E	F	G
1	Qualidade de modularização:	3,03		Apresentar relatório			3,03
2	Fatores de cluster	0,67	0,67	0,77	0,92		
3	relac. intra	2,00	2,00	5,00	6,00		
4	relac. extra	2,00	2,00	3,00	1,00		
5							
6	subsistema	tarefa	matriz de dependência				
7	1	1	2,01	5,02	10,01		
8	2	2	6,02				
9	3	3	4,02	8,01	10,01		
10	3	4					
11	1	5					
12	2	6	10,01				
13	4	7	8,01	12,01			
14	4	8					
15	3	9	10,02				
16	3	10					
17	4	11	7,02				
18	4	12	8,02				

Ilustração 46 - Resultado da otimização para 300 organismos.

A ilustração 47 apresenta a distribuição das responsabilidades entre os subsistemas depois da otimização. O valor da qualidade de modularização é 3,03

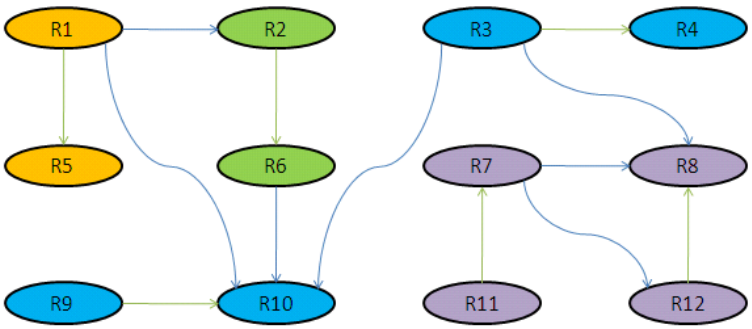

Ilustração 47 - MDG final para uma população de 300 organismos.

.

A ilustração 48 mostra que o melhor valor de qualidade de modularização de 3,03 foi obtido na tentativa número 2.604. O algorítmo foi executado até a iteração 5.753 e o tempo consumido foi de 2 minutos e 10 segundos.

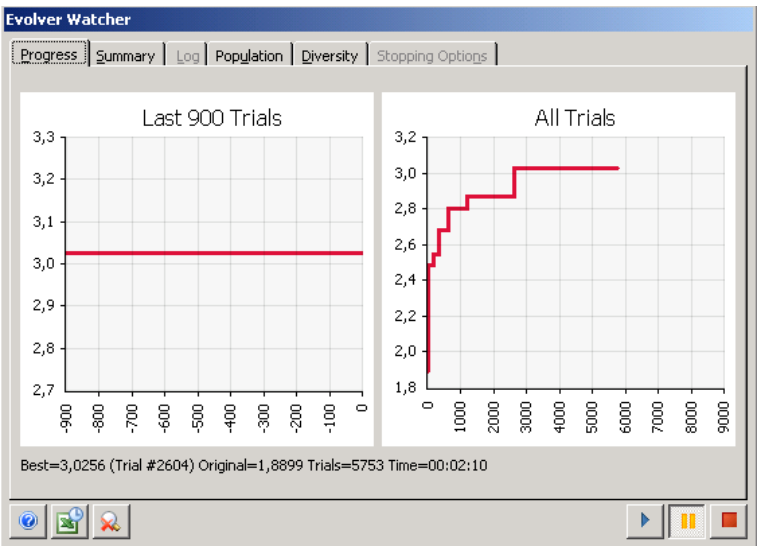

Ilustração 48 - Caixa de diálogo de monitoramento - 300 organismos.

No experimento com 50 organismos, vimos na ilustração 44 (Evolver – Watcher para 50 organismos) que após 1000 iterações, a população já havia convergido e não havia mais diversidade.

Neste experimento com 300 organismos, a ilustração 49 mostra que mesmo na iteração número 5.753, a população ainda apresenta diversidade.

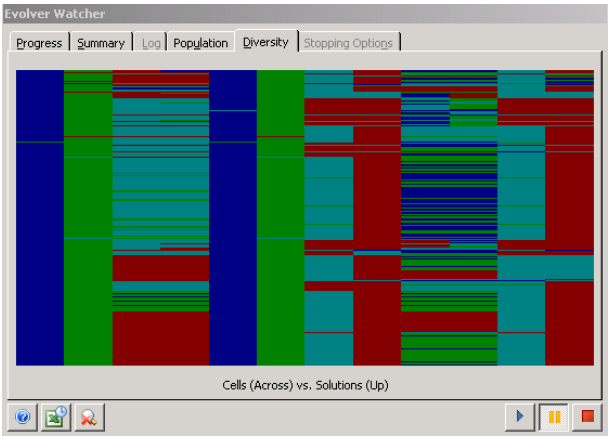

Ilustração 49 - Caixa de diálogo de diversidade da população para 300 organismos.

Para o particionamento do sistema em três subsistemas que em conjunto implementem as 12 responsabilidades, a seguinte solução inicial foi considerada: o subsistema 1 implementa as responsabilidades 1, 2, 3 e 4; o subsistema 2 implementa as responsabilidades 5, 6, 7 e 8; e o subsistema 3 implementa as responsabilidades 9, 10, 11 e 12.

Nesta situação o valor da qualidade de modularização é de 0,95. As ilustrações abaixo apresentam respectivamente a planilha inicial e a representação visual da solução inicial.

	A	B	C	D	E	F	G
1	Qualidade de modularização:	0,95		Apresentar relatório			0,95
2	Fatores de cluster	0,46	0,15	0,33			
3	relac. intra	3,00	1,00	2,00			
4	relac. extra	7,00	11,00	8,00			
5							
6	subsistema	tarefa	matriz de dependência				
7	1	1	2,01	5,02	10,01		
8	1	2	6,02				
9	1	3	4,02	8,01	10,01		
10	1	4					
11	2	5					
12	2	6	10,01				
13	2	7	8,01	12,01			
14	2	8					
15	3	9	10,02				
16	3	10					
17	3	11	7,02				
18	3	12	8,02				

Ilustração 50 - Planilha Excel inicial para o particionamento de 3 subsistemas.

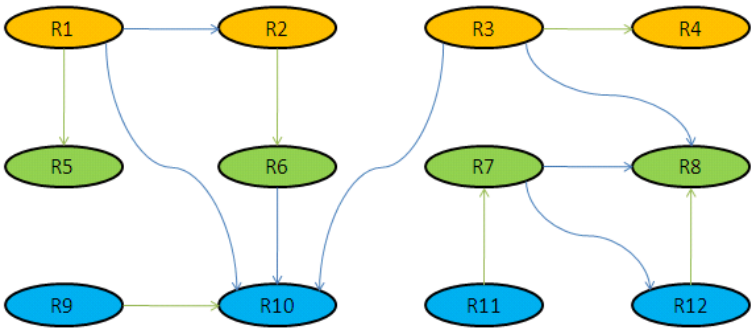

Ilustração 51 - MDG inicial para o particionamento em 3 subsistemas.

Como o número de subsistemas desejados se alterou de 4 para 3, é necessário atualizar o valor da variável *number_of_subsystems* de 4 para 3 na seção 'Atribuição de valores iniciais' da função Calcular_cluster_factor e da macro Apresentar_cluster_factor. Os anexos 1 e 2 apresentam o código-fonte da função e macro mencionadas.

Os valores da definição do modelo e das configurações de otimização foram mantidos com os mesmos valores do experimento que resultou em um melhor valor para a qualidade de modularização para o caso do particionamento em quatro subsistemas. O quadro 6 apresenta o valor dos principais parâmetros da definição do modelo e das configurações de otimização:

Quadro 6 - Principais parâmetros de otimização

Parâmetro	Valor
Método de resolução	Grouping
Taxa de crossover	0,5
Taxa de mutação	0,2
População	300

O melhor resultado obtido foi um valor de 2,54 para a qualidade de modularização na iteração de número 4.364. O algoritmo foi executado durante 2 minutos e 40 segundos e foi possível realizar 7.026 tentativas neste intervalo de tempo.

As ilustrações as seguir apresentam, respectivamente, a planilha Excel com a melhor solução obtida, a representação gráfica desta solução e a evolução do valor da qualidade de modularização ao longo das iterações.

	A	B	C	D	E	F	G
1	Qualidade de modularização:	2,54		Apresentar relatório			0,95
2	Fatores de cluster	0,46	0,15	0,33			
3	relac. intra	3,00	1,00	2,00			
4	relac. extra	7,00	11,00	8,00			
5							
6	subsistema	tarefa	matriz de dependência				
7	3	1	2,01	5,02	10,01		
8	3	2	6,02				
9	1	3	4,02	8,01	10,01		
10	1	4					
11	3	5					
12	3	6	10,01				
13	2	7	8,01	12,01			
14	2	8					
15	3	9	10,02				
16	3	10					
17	2	11	7,02				
18	2	12	8,02				

Ilustração 52 - Resultado da otimização para particionamento em 3 subsistemas.

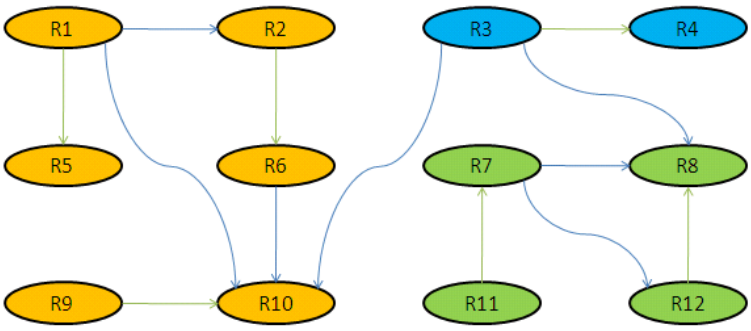

Ilustração 53 - MDG final para particionamento em 3 subsistemas.

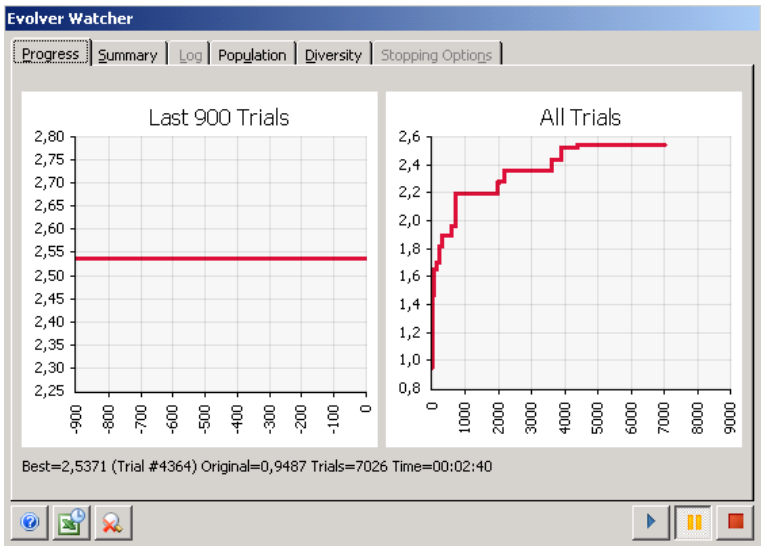

Ilustração 54 - Caixa de diálogo de monitoramento - 300 organismos, 3 subsistemas.

Executando-se os passos utilizados nos casos com três e quatro subsistemas para os demais casos (n subsistemas, com n = {2, 5, 6, ..., 11}), obtém-se os seguintes valores para a qualidade de modularização.

Quadro 7 - Número de subsistemas vs Qualidade de modularização.

Número de subsistemas	Qualidade de modularização
1	1,00
2	1,89
3	2,54
4	3,03
5	3,49
6	3,81
7	3,24
8	2,67
9	2,00
10	1,33
11	0,67
12	0,00

O quadro 7 pode ser representado graficamente pela ilustração 55.

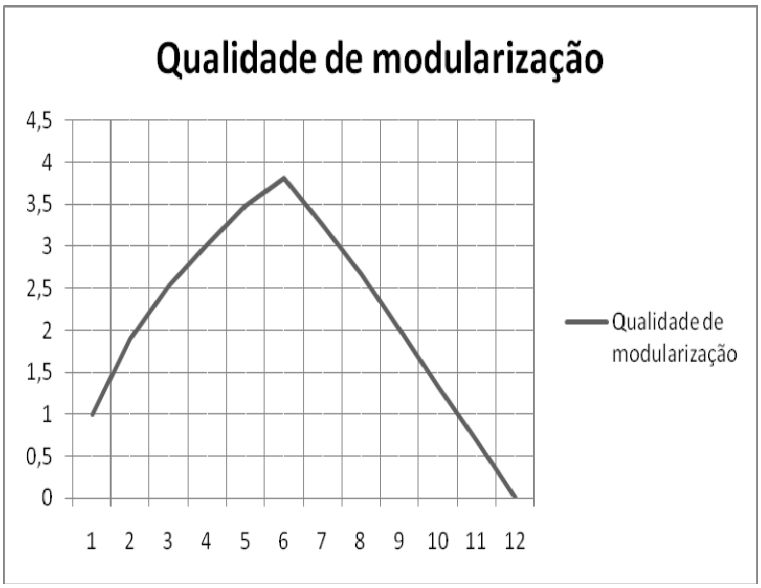

Ilustração 55 - Qualidade de modularização em função do número de subsistemas.

O melhor valor para a qualidade de modularização considerando todos os números possíveis de subsistemas é 3,81. Esse valor ótimo ocorre para o particionamento em 6 subsistemas. As duas ilustrações a seguir apresentam o particionamento inicial (MQ = 1,77) e final (MQ = 3,81) para este caso.

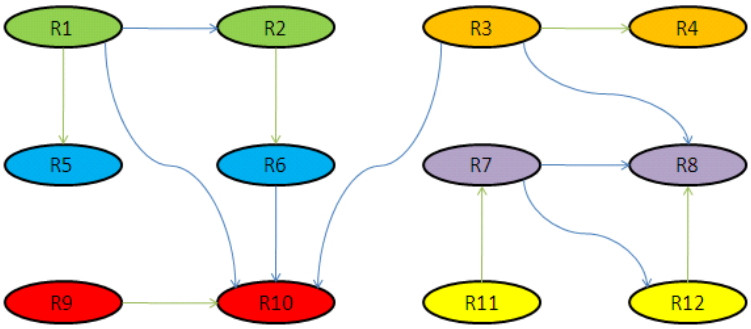

Ilustração 56 - MDG inicial para 6 subsistemas.

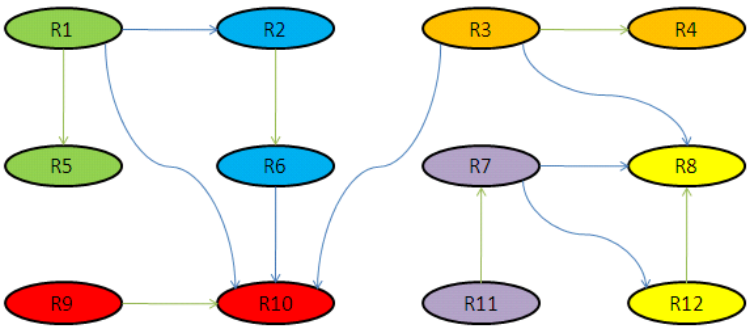

Ilustração 57 - MDG final para 6 subsistemas.

Considerando o particionamento final para 6 subsistemas proposto, é interessante observar que todos os relacionamentos com peso 2 (representados através das setas de cor verde), estão contidos dentro dos subsistemas.

As ilustrações seguintes apresentam, respectivamente, o sistema sem nenhum particionamento proposto e o sistema particionado em 6 subsistemas evidenciando que os relacionamentos de maior peso foram alocados internamente aos subsistemas.

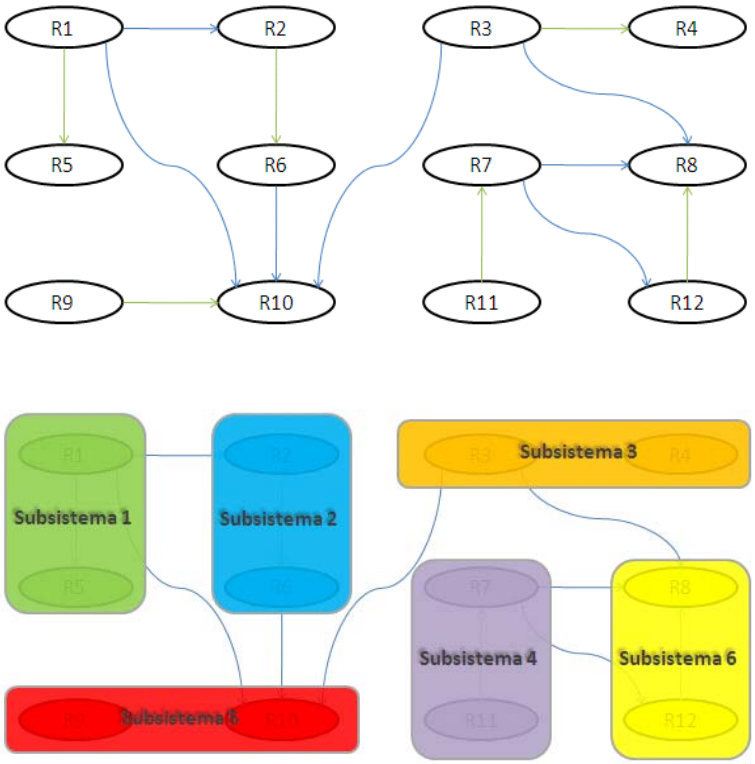

Ilustração 58 - MDG apresentando a melhor solução encontrada.

Quando o método de resolução por agrupamento é utilizado, o Evolver permite escolher se é necessário que todos os grupos possuam pelo menos um elemento. Essa escolha pode ser feita marcando a opção '*All Groups Must Be Used*' da janela '*Evolver – Adjustable Cell Group Settings*'.

Para solicitar ao Evolver que otimize o valor da qualidade de modularização considerando qualquer número de subsistemas entre 1 e 12, pode-se atribuir o valor 12 para a variável *number_of_subsystems* na seção 'Atribuição de valores iniciais' da função Calcular_cluster_factor e da macro Apresentar_cluster_factor e não marcar a opção '*All Groups Must Be Used*' da janela '*Evolver – Adjustable Cell Group Settings*'.

Mantendo-se as demais configurações inalteradas e partindo-se de uma situação em que cada uma das 12 responsabilidades define um subsistema, este experimento apresentou a mesma solução do experimento que se iniciou com 6 subsistemas fixos. O mesmo particionamento em 6 subsistemas e o mesmo valor da qualidade de modularização foi alcançado conforme apresentado na ilustração 59.

	A	B	C	D	E	F	G
1	Qualidade de modularização:	3,81		Apresentar relatório			3,81
2	Fatores de cluster	0,67	0,67	0,00	0,67	0,57	0,00
3	relac. intra	0,00	0,00	0,57	0,67	0,00	0,00
4	relac. extra	0,00	0,00	2,00	2,00	0,00	0,00
5		0	0,00	3,00	2,00	0,00	
6	subsistema	tarefa	matriz de dependência				
7	1	1	2,01	5,02	10,01		
8	2	2	6,02				
9	4	3	4,02	8,01	10,01		
10	4	4					
11	1	5					
12	2	6	10,01				
13	11	7	8,01	12,01			
14	5	8					
15	10	9	10,02				
16	10	10					
17	11	11	7,02				
18	5	12	8,02				

Ilustração 59 - Resultado para particionamento podendo variar de 1 a 12.

A diferença entre o experimento em que se fixou o número de subsistemas em 6 e o experimento em que se deixou o número de subsistemas livre é que no primeiro caso o melhor resultado foi obtido na iteração 2.100 enquanto que no segundo caso este resultado só foi alcançado na iteração 23.758.

As ilustrações seguintes apresentam a evolução do valor da qualidade de modularização ao longo das iterações para os dois experimentos respectivamente.

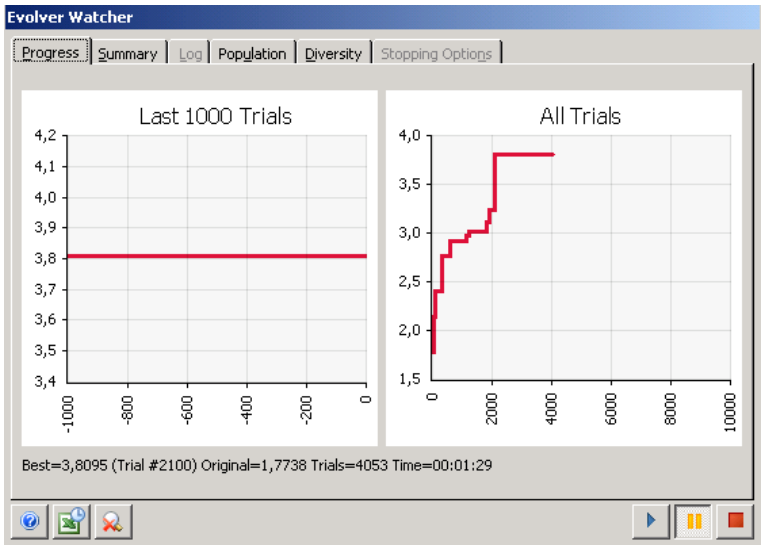

Ilustração 60 - Evolução da otimização com 6 subsistemas fixos.

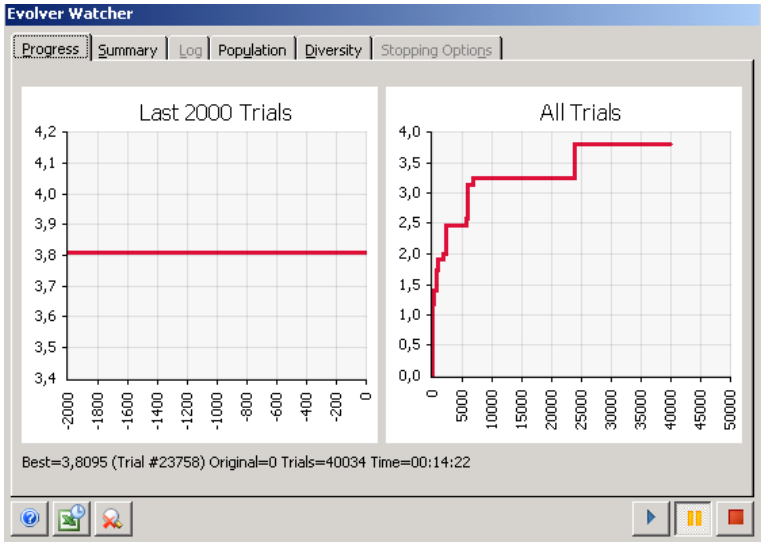

Ilustração 61 - Evolução da otimização com número de subsistemas livre.

Este capítulo exemplificou uma abordagem quantitativa para determinar uma possível arquitetura de software que almeja maximizar um atributo de qualidade. A abordagem quantitativa se baseou no método de algorítmo genético, o aspecto da arquitetura de software enfocada foi o aspecto estrutural, e o atributo de qualidade considerado foi a capacidade de modificação do sistema mensurado pela métrica qualidade de modularização.

7 CONSIDERAÇÕES FINAIS

Este capítulo apresenta: (1) uma síntese das sete questões (etapas) que compõem o trabalho, (2) as contribuições do estudo, (3) as limitações do estudo e (4) questões para estudos futuros.

(1) uma síntese das sete questões que compõem o trabalho.

1 O que é qualidade?

Qualidade é um termo cujo conceito é amplo e pode variar de pessoa para pessoa. Com o objetivo de restringir o conceito e compreender melhor o que causa a variação no entendimento deste conceito de uma pessoa para outra, as visões sobre qualidade de cinco autores foram apresentadas. Um aspecto que afeta o entendimento deste conceito é o ponto de vista adotado. Um ponto de vista é centrado no produto e neste caso qualidade está associada à conformidade a um conjunto de requerimentos do produto. Outro ponto de vista é centrado no usuário e neste caso qualidade está associada a satisfação do usuário com relação ao produto.

2 O que é qualidade de software?

Com uma compreensão melhor sobre o conceito de qualidade, três modelos de qualidade de software foram apresentados com o objetivo de refinar o conceito de qualidade para o produto software. Os modelos apresentam em comum uma organização na forma hierárquica. O número de níveis da hierarquia e os elementos em cada nível da hierarquia variam de um modelo para outro mas o conceito subjacente a esta forma de organização é o mesmo. O primeiro nível divide o conceito de qualidade de software em diversas características (perspectivas). Os demais níveis dividem cada uma das características (perspectivas) do nível acima em novas características (ou sub-características) de forma que cada nova característica (ou sub-característica) tenha um conceito mais preciso.

3 Como mensurar qualidade de software?

A mensuração da qualidade de software não é definida por uma única métrica. Cada característica (ou sub-característica) da qualidade cujo conceito é mais preciso (restrito) pode ser mensurada por uma ou mais métricas de software. Com o objetivo de ilustrar (exemplificar) como uma característica (ou sub-característica) pode ser mensurada, uma métrica de software foi apresentada. A métrica de software apresentada procura mensurar a característica capacidade de modificação do software e é utilizada no capítulo 6 – Aplicação do processo de design da arquitetura de software.

4 O que é arquitetura de software?

Arquitetura de software é um termo cujo conceito apresenta alguma variação de autor para autor. Dois aspectos do conceito são comuns à maioria das interpretações. O primeiro aspecto é a associação do termo à idéia de uma estrutura geral (estrutura com alto grau de abstração). O segundo aspecto é a associação do termo à idéia de uma estrutura composta. Os elementos e os relacionamentos entre os elementos que compõem a estrutura definem a arquitetura de software. Garlan (2010, p.1) acrescenta que a estrutura ou conjunto de estruturas que definem a arquitetura de software são determinadas pela necessidade de se analisar algum aspecto do sistema. Essa idéia conduz à questão 5 devido a necessidade de se representar o conjunto de estruturas para se efetuar a análise.

5 Como representar a arquitetura de software?

As representações ou visões da arquitetura de software são diversas e não existe um consenso sobre qual conjunto de visões melhor representa a arquitetura de software. Analisando diversas visões, dois fatores parecem explicar ou direcionar qual parte da arquitetura de software é apresentada em uma única visão. O primeiro fator está associado ao objetivo da visão, e o objetivo da visão parece ser auxiliar a responder uma ou um pequeno número de questões. Freqüentemente, as questões procuram responder se um sistema apresenta ou não um determinado atributo de qualidade. O segundo fator está associado à coerência a um critério. Por exemplo, se o critério for composição, procura-se representar em uma visão apenas elementos que compostos totalizem o sistema. Se em outro momento o critério for

tempo de execução, procuram-se representar nesta visão apenas os elementos que tenham existência em tempo de execução.

6 Como variar a representação da arquitetura de software?

O processo de design da arquitetura de software pode ser visto como um processo de decisão. No decorrer do processo diversas decisões são tomadas de forma que ao final do processo a arquitetura de software esteja definida. Uma forma de facilitar ou direcionar a tomada de decisões é através da reutilização de decisões tomadas e que foram bem sucedidas. Um conjunto de decisões tomadas pode ser registrado e posteriormente reutilizado na forma de táticas de arquitetura, padrões de arquitetura, padrões de design e frameworks. A utilização de uma ou mais destas quatro formas de se representar um conjunto de decisões tomadas cria uma variedade de representações da arquitetura de software. Estas várias arquiteturas de software possíveis simplificam e orientam o processo de design da arquitetura de software ao restringir o total de arquiteturas de software possíveis à um número menor.

7 Como especificar uma arquitetura de software que apresente uma qualidade de software desejada?

A especificação da arquitetura de software é elaborada durante o processo de design de alto nível. Dois grandes paradigmas do processo de software (o paradigma clássico e o paradigma orientado a objetos) foram estudados para compreender como eles determinam ou direcionam a especificação de uma arquitetura de software. A terceira abordagem para o processo de design da arquitetura de software considerada é a abordagem orientada à atributos de Bass (2003). A quarta e última abordagem foi denominada de abordagem orientada a busca e pode ser considerada uma aplicação da Engenharia de software orientada a busca ("*Search based software engineering*"). O estudo das quatro abordagens permite compreender formas diferentes pelas quais a arquitetura de software é especificada para se satisfazer uma dada qualidade de software.

(2) as contribuições do estudo.

Muitos estudos têm sido feitos sobre o tema arquitetura de software e sua relação com qualidade de software. Em muitos destes estudos a relação entre arquitetura de software e qualidade de software é dada como certa apesar do termo arquitetura de software não ter uma definição consensual até hoje. A contribuição deste estudo é apresentar este tema enfatizando a natureza desta relação. Um entendimento mais claro sobre a natureza desta relação pode estimular a criação de novos processos de design de arquitetura de software e/ou o aperfeiçoamento dos processos existentes.

O capítulo 4 enfatiza a importância de se refinar o conceito de arquitetura de software. Em última instância a relação entre arquitetura de software e qualidade de software se dá por meio da relação entre uma representação (perspectiva) da arquitetura de software e a qualidade de software.

Outro ponto destacado, no capítulo 3, é a importância de se refinar o conceito de qualidade de software. Em última instância a relação entre arquitetura de software e qualidade de software ocorre entre a arquitetura de software e uma métrica de software que mensura um aspecto da qualidade de software.

O capítulo 5 apresenta quatro abordagens para direcionar o processo de design da arquitetura de software e encerra apresentando um quadro comparativo destas abordagens com base em três características observadas durante o estudo.

O capítulo 6 aprofunda o entendimento da abordagem orientada a busca através de uma aplicação que usa a técnica de algoritmos genéticos para otimizar o atributo de qualidade capacidade de modificação do software. Esta técnica apresenta como ponto forte a obtenção automática da representação da arquitetura de software dado que o modelo matemático foi especificado. O ponto fraco é justamente a especificação de um modelo matemático que não limite ou simplifique muito o problema real. Uma das simplificações realizadas para apresentar a aplicação foi a consideração de apenas um atributo de qualidade. Na prática, a maioria das situações solicitam a otimização de diversos atributos de qualidade.

(3) as limitações do estudo.

O termo limitações em “as limitações do estudo” tem o sentido de pressupostos, ou seja, é necessário assumir que os pressupostos são verdadeiros para que se possa estabelecer uma relação entre qualidade e arquitetura de software. Nesse sentido os pressupostos condicionam, ou seja, limitam a existência desta relação. Os três pressupostos assumidos são:

- A qualidade de software ou uma característica da qualidade de software pode ser mensurada por uma métrica de software.

- A arquitetura de software pode ser representada adequadamente por um conjunto de visões de arquitetura de software.

- As descrições das quatro abordagens para o processo de design da arquitetura de software apresentadas nos capítulos 5 e 6 representam adequadamente a essência destas abordagens.

(4) questões para estudos futuros.

A ilustração 62 apresenta o processo de design da arquitetura de software e suas relações com: o repositório de conhecimentos de arquitetura, a arquitetura de software e a qualidade de software. Esta ilustração é utilizada para contextualizar as três questões propostas para estudos futuros.

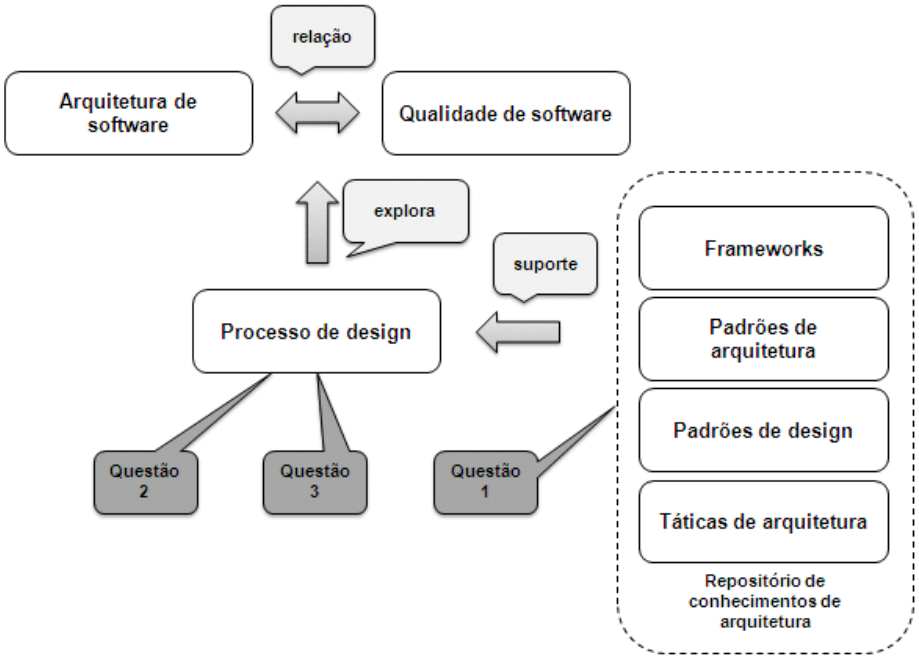

Ilustração 62 - Três questões para estudos futuro contextualizadas no processo de design.

1 Como organizar conceitualmente o repositório de conhecimentos de arquitetura?

Esta questão sugere estudos para esclarecer quais devem ser os elementos do repositório e como estes elementos podem ser definidos conceitualmente de forma coerente.

2 Como aperfeiçoar o processo de design orientado a busca?

Esta questão sugere estudos que: (1) permita levar em consideração vários atributos de qualidade simultaneamente, (2) permita levar em consideração informações futuras a respeito do ambiente de software e (3) permita incorporar conhecimentos de arquitetura no algoritmo de busca escolhido.

3 Como aperfeiçoar o processo de design orientado a atributos?

Um ponto que pode ser investigado é como estruturar o processo de design orientado a atributos de forma que parte do processo possa ser automatizada.

REFERÊNCIAS

AL-QUTAISH, R. E. Quality Models in Software Engineering Literature: An Analytical and Comparative Study. **Journal of American Science**, p. 166-175, 2010.

AVGERIOU, P.; ZDUN, U. **Architectural Patterns Revisited - A Pattern Language**. Tenth European Conference on Pattern Languages of Programs (EuroPlop). [S.l.]: [s.n.]. 2005.

BACHMANN, F.; BASS, L.; KLEIN, M. **Illuminating the fundamental contributors to software architecture quality**. [S.l.]: Software Engineering Institute, Carnegie Mellon University, 2002.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. [S.l.]: Addison-Wesley, 2003.

BROOKS, F. P. J. **The mythical man-month**. [S.l.]: Addison-Wesley, 1995.

CASTELEYN, S. et al. **Engineering Web Applications**. [S.l.]: Springer-Verlag, 2009.

CIRIBELLI, M. C. **Como elaborar uma dissertação de mestrado através da pesquisa científica**. [S.l.]: Editora 7Letras, 2003.

CROSBY, P. B. **Quality is free:** the art of making quality certain. New York: McGraw-Hill, 1979.

DEMARCO, T. Software engineering: An idea whose time has come and gone? **IEEE Software**, 2009.

DEMING, W. E. **Out of the crisis**. [S.l.]: MIT Press, 2000.

DEMO, P. **Pesquisa e construção do conhecimento:** metodologia científica no caminho de Habermas. [S.l.]: Editora Tempo Brasileiro, 1994.

DEMO, P. **Metodologia do conhecimento científico**. [S.l.]: Editora Atlas, 2000.

FEIGENBAUM, A. V. **Total quality control**. [S.l.]: McGraw-Hill, 1991.

FEYERABEND, P. **Contra o método**. [S.l.]: Francisco Alves, 1989.

GAMMA, E. et al. **Design Patterns:** Elements of reusable object-oriented software. [S.l.]: Addison-Wesley, 1995.

GARLAN, D.; SHAW, M. **An introduction to software architecture**. Advances in Software Engineering and Knowledge Engineering. [S.l.]: World Scientific Publishing. 1993.

GARLAND, J.; RICHARD, A. **Large-scale software architecture**. [S.l.]: Wiley, 2003.

GIL, A. C. **Como elaborar projetos de pesquisa**. [S.l.]: Editora Atlas, 1991.

HARMAN, M. **The Current State and Future of Search Based Software Engineering**. Future of Software Engineering. Minneapolis: King's College London. 2007. p. 342-357.

HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. **Search Based Software Engineering: Comprehensive Analysis and Review of Trends Techniques and Applications**. [S.l.]. 2009.

IEEE. **IEEE Recommended Practice for Architectural Description of Software-Intensive Systems**. IEEE Computer Society. [S.l.]. 2000. (IEEE Std 1471-2000).

ISHIKAWA, K. **What is total quality control? The japanese way**. [S.l.]: Prentice-Hall, 1985.

JALOTE, P. **A concise introduction do software engineering**. [S.l.]: Springer, 2008.

JOHNSON, B. et al. **Flexible Software Design**. [S.l.]: Auerbach Publications, Taylor & Francis Group, 2005.

KANDT, R. K. **Software Engineering Quality Practices**. [S.l.]: Auerbach Publications, Taylor & Francis Group, 2006.

KHAN, B.; SOHAIL, S.; JAVED, M. Y. ESBASCA: A novel software clustering approach. **International Journal of Software Engineering and Its Applications**, p. 1-18, 2009.

KHOSRAVI, K.; GUÉHÉNEUC, Y.-G. **On Issues with Software Quality Models**. [S.l.]: ICFAI University Press, 2008.

KRUCHTEN, P. The 4+1 view model of software architecture. **IEEE Sofware**, v. 12, n. 6, 1995.

KUHN, T. **The structure of scientific revolutions**. [S.l.]: University of Chicago Press, 1970.

LAND, R. **Software deterioration and maintainability – A model proposal**. Second Conference on Software Engineering Research and Practise. [S.l.]: [s.n.]. 2002. p. 1-7.

LATTANZE, A. J. **Architecting software intensive systems:** a practitioner's guide. [S.l.]: Auerbach Publications, Taylor & Francis Group, 2009.

MILICIC, D. **Software Quality Attributes and Trade-offs**. [S.l.]: Blekinge Institute of Technology, 2005.

MITCHELL, B. S. **A Heuristic Search Approach to Solving the Software Clustering Problem**. Drexel University. [S.l.], p. 262. 2002.

PARNAS, D. On the criteria to be used in decomposing systems into modules. **Communications of ACM**, p. 1053-1058, 1972.

POPPER, K. **A lógica da pesquisa científica**. [S.l.]: Cultrix, 1993.

QIN, Z.; XING, J.; ZHENG, X. **Software architecture in Advanced topics in science and technology in China**. [S.l.]: Zhejiang University Press and Springer-Verlag, 2008.

RÄIHÄ, O. **Applying Genetic Algorithms in Software Architecture Design**. [S.l.]. 2008.

SHEWART, W. A. **Economic control of quality of manufactured product**. [S.l.]: ASQ Quality Press, 1980.

SILVA, E. L. D.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. [S.l.]: UFSC, 2001.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Addison-Wesley, 2007.

STAIR, R. M.; REYNOLDS, G. W. **Princípios de sistemas de informação**. [S.l.]: Thomson, 2006.

GLOSSÁRIO

Arquitetura de software. "A organização fundamental de um sistema conforme representada por seus componentes, pelos relacionamentos entre os componentes, e com o ambiente, e pelos princípios que orientam seu design e evolução." (QIN, 2008, p. 13).[38]

Interface. Um conjunto de assinaturas que podem ser implementadas por um módulo. A interface de um módulo define o conjunto de serviços suportados por ele.

Manutenibilidade. Facilidade com que um sistema de software ou componente pode ser modificado para corrigir falhas, melhorar a performance ou outros atributos, ou adaptado para uma mudança de ambiente.

Objeto. Uma entidade existente em tempo de execução que contêm dados e procedimentos que operam sobre estes dados.

Sistema de entrada de pedidos. Sistema que captura dados básicos necessários para processar os pedidos dos clientes.

Sistema de planejamento de entregas. Sistema que determina quais pedidos em aberto serão cumpridos e de que local eles serão enviados.

Sistema de controle de estoque. Sistema que atualiza os registros do estoque para refletir a quantidade exata em mãos para cada unidade de estoque.

Sistema de administração de relacionamento com clientes. Sistema que auxilia a empresa na administração das relações com os clientes de modo eficaz.

Sistema de roteamento. Sistema que determina a melhor maneira de se transportar produtos de um local a outro.

---

[38] "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution".

Sistema de contas a receber. Sistema que auxilia a administração do fluxo de caixa da empresa por meio do acompanhamento dos valores devido à empresa por bens vendidos e serviços prestados.

Sistemas de inteligência artificial. Sistemas que demonstrem características de inteligência.

Sistemas de realidade virtual. Sistemas que permitem que um ou mais usuários interajam em um ambiente simulado por computador.

Sistemas de robótica. Sistemas compostos por dispositivos mecânicos ou computacionais que efetuem tarefas que exigem um alto grau de precisão ou que são repetitivas ou perigosas para os seres humanos.

Sistemas de visão. Sistemas que permitem que computadores capturem, armazenem e manipulem imagens visuais.

Sistemas de processamento de linguagem natural. Sistemas que permitem que computadores entendam declarações em linguagem natural como o português ou o inglês.

Sistemas de aprendizado. Sistemas que permitem que o computador altere suas funções ou reaja a situações com base em realimentação.

Sistemas de redes neurais. Sistemas que permitem que o computador simule o funcionamento do cérebro humano.

Sistemas especialistas. Sistemas que permitem que um usuário comum possa ter um desempenho similar ao de um especialista em um campo específico.

APÊNDICES

APÊNDICE 1 – Código-fonte da macro Apresentar_cluster_factor

```
Sub Apresentar_cluster_factor()

    'Variáveis para a responsabilidade
    Dim number_of_resp As Integer
    Dim resp_iter As Integer
    Dim resp_chamada As Integer
    Dim peso_do_relac As Double

    'Variáveis para o subsistema
    Dim number_of_subsystem As Integer
    Dim subsistema_indice As Range
    Dim subsistema_atual As Integer
    Dim subsistema_resp_chamada As Integer
    Dim subsistema_iter As Integer

    'Variáveis para a matriz de dependência
    Dim matriz_dep As Range
    Dim matriz_dep_cel_iter As Range

    'Variáveis dos fatores de cluster
    Dim intra() As Long
    Dim extra() As Long
    Dim cluster_factor() As Double
    Dim cluster_factor_total As Double

    'Variáveis para debug
    Dim debug_cluster As Range
    Dim debug_intra As Range
    Dim debug_extra As Range

    'Atribuição de valores iniciais
    number_of_resp = 12
    number_of_subsystem = 12
    Set subsistema_indice = Range("a7:a18")
    Set matriz_dep = Range("c7:i18")
    Set debug_cluster = Range("b2:h2")
    Set debug_intra = Range("b3:h3")
    Set debug_extra = Range("b4:h4")
    ReDim intra(1 To number_of_subsystem)
    ReDim extra(1 To number_of_subsystem)
    ReDim cluster_factor(1 To number_of_subsystem)

    'Início do cálculo da qualidade de modularização
    'Soma dos pesos dos relacionamentos dentro e entre os subsistemas
    For resp_iter = 1 To number_of_resp
        subsistema_atual = subsistema_indice.Cells(resp_iter).Value
        For Each matriz_dep_cel_iter In matriz_dep.Rows(resp_iter).Cells
            If (matriz_dep_cel_iter.Value > 1) Then
                resp_chamada = Int(matriz_dep_cel_iter)
                subsistema_resp_chamada = subsistema_indice.Cells(resp_chamada)
                peso_do_relac = 100 * (CDbl(matriz_dep_cel_iter) - resp_chamada)
                If subsistema_atual = subsistema_resp_chamada Then
                    intra(subsistema_atual) = intra(subsistema_atual) + peso_do_relac
                Else
```

```
                extra(subsistema_atual) = extra(subsistema_atual) + peso_do_relac
                extra(subsistema_resp_chamada) = extra(subsistema_resp_chamada) + peso_do_relac
            End If
        End If
    Next
Next

'Cálculo dos fatores de cluster para cada subsistema e a qualidade de modularização do sistema
For subsistema_iter = 1 To number_of_subsystem
    If intra(subsistema_iter) <> 0 Then
        cluster_factor(subsistema_iter) = intra(subsistema_iter) / (intra(subsistema_iter) + 0.5 *
extra(subsistema_iter))
    Else
        cluster_factor(subsistema_iter) = 0
    End If
    debug_cluster.Cells(subsistema_iter).Value = cluster_factor(subsistema_iter)
    cluster_factor_total = cluster_factor_total + cluster_factor(subsistema_iter)
    debug_intra.Cells(subsistema_iter).Value = intra(subsistema_iter)
    debug_extra.Cells(subsistema_iter).Value = extra(subsistema_iter)
Next
Range("g1") = cluster_factor_total

End Sub
```

APÊNDICE 2 – Código-fonte da função Calcular_cluster_factor

```
Public Function Calcular_cluster_factor(ByVal p_subsistema_indice As Range, ByVal p_matriz_dep
As Range) As Double

    'Variáveis para a responsabilidade
    Dim number_of_resp As Integer
    Dim resp_iter As Integer
    Dim resp_chamada As Integer
    Dim peso_do_relac As Double

    'Variáveis para o subsistema
    Dim number_of_subsystem As Integer
    Dim subsistema_indice As Range
    Dim subsistema_atual As Integer
    Dim subsistema_resp_chamada As Integer
    Dim subsistema_iter As Integer

    'Variáveis para a matriz de dependência
    Dim matriz_dep As Range
    Dim matriz_dep_cel_iter As Range

    'Variáveis dos fatores de cluster
    Dim intra() As Long
    Dim extra() As Long
    Dim cluster_factor() As Double
    Dim cluster_factor_total As Double

    'Atribuição de valores iniciais
    Set subsistema_indice = p_subsistema_indice
    Set matriz_dep = p_matriz_dep
    number_of_resp = 12
    number_of_subsystem = 12
    ReDim intra(1 To number_of_subsystem)
    ReDim extra(1 To number_of_subsystem)
    ReDim cluster_factor(1 To number_of_subsystem)

    'Início do cálculo da qualidade de modularização
    'Soma dos pesos dos relacionamentos dentro e entre os subsistemas
    For resp_iter = 1 To number_of_resp
        subsistema_atual = subsistema_indice.Cells(resp_iter).Value
        For Each matriz_dep_cel_iter In matriz_dep.Rows(resp_iter).Cells
            If (matriz_dep_cel_iter.Value > 1) Then
                resp_chamada = Int(matriz_dep_cel_iter)
                subsistema_resp_chamada = subsistema_indice.Cells(resp_chamada)
                peso_do_relac = 100 * (CDbl(matriz_dep_cel_iter) - resp_chamada)
                If subsistema_atual = subsistema_resp_chamada Then
                    intra(subsistema_atual) = intra(subsistema_atual) + peso_do_relac
                Else
                    extra(subsistema_atual) = extra(subsistema_atual) + peso_do_relac
                    extra(subsistema_resp_chamada) = extra(subsistema_resp_chamada) + peso_do_relac
                End If
            End If
        Next
    Next
```

```
    'Cálculo dos fatores de cluster para cada subsistema e a qualidade de modularização do sistema
    For subsistema_iter = 1 To number_of_subsystem
        If intra(subsistema_iter) <> 0 Then
            cluster_factor(subsistema_iter) = intra(subsistema_iter) / (intra(subsistema_iter) + 0.5 *
extra(subsistema_iter))
        Else
            cluster_factor(subsistema_iter) = 0
        End If
        cluster_factor_total = cluster_factor_total + cluster_factor(subsistema_iter)
    Next
    Calcular_cluster_factor = cluster_factor_total

End Function
```