Ministério da Educação
Centro Federal de Educação Tecnológica do Paraná
Programa de Pós-Graduação em Eng. Elétrica e Informática Industrial

DESENVOLVIMENTO DE *SOFTWARE* ORIENTADO A OBJETOS BASEADO NO MODELO AXIOMÁTICO

por
Andrey Ricardo Pimentel
Orientador: Prof. Dr. Paulo Cézar Stadzisz

Curitiba, dezembro de 2004

ANDREY RICARDO PIMENTEL

DESENVOLVIMENTO DE *SOFTWARE* BASEADO NO MODELO AXIOMÁTICO

Monografia de qualificação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) do Centro Federal de Educação Tecnológica do Paraná (CEFET-PR), como parte dos requisitos para obtenção do título de Doutor em Ciências. Área de Concentração: Informática Industrial.

Orientador: Prof. Dr. Paulo Cézar Stadzisz

CURITIBA PR

Dezembro 2004

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

UML	*Unified Modeling Language*
RUP	*Rational Unified Process*
OMT	*Object Modeling Technique*
GRASP	*General Responsibility Assignment Software Patterns*
CPGEI	Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial
CEFET-PR	Centro Federal de Educação Tecnológica do Paraná
WDK	*Workshop Design Konstruktion*
FR	*Functional Requirement*
DP	*Design Parameter*
PV	*Process Variable*
CN	*Customer Needs*
ADo-oSS	*Axiomatic Design of object-oriented Software Systems*
MVC	Modelo-Visão-Controlador
ucpc	*use case points* contados
eucp	estimativa baseada nos *use case points*
pfc	pontos por função contados
epf	estimativa dos pontos por função
WMC	*Weighted Methods per Class*
DIT	*Depth of the Inheritance Tree*
NOC	*Number Of Children*
CBO	*Coupling Between Objects*
RFC	*Response For a Class*
LCOM	*Lack of COhesion of Methods*
CASE	*Computer Aided Software Engineering*
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico

RESUMO

Este trabalho propõe a criação de uma metodologia de projeto de *software* baseada na adaptação da teoria de projeto axiomático ao projeto de *software* orientado a objetos. Esta metodologia tem por objetivo fornecer parâmetros objetivos para a tomada de decisão sobre qual das possíveis soluções para um projeto é a melhor, visando garantir a qualidade e a robustez do projeto.

A motivação para a realização deste trabalho é tornar o processo de projeto de *software* mais preciso e confiável de forma a garantir sua qualidade. A teoria de projeto axiomático foi estabelecida para garantir esta qualidade desde a elaboração do projeto do produto.

A metodologia proposta neste trabalho é baseada na teoria de projeto axiomático. Neste trabalho, os conceitos do projeto axiomático são adaptados para o uso em projetos de *software*, de modo a auxiliar e facilitar a garantia de sua qualidade. Esta metodologia tem como conceito fundamental, os mapeamentos entre os domínios do projeto. Para o uso em projeto de *software*, são propostos novos domínios complementares, típicos de projetos de *software*, como: o domínio de casos de uso, o domínio de classes, o domínio de interações e o domínio de estados. Outro conceito fundamental é o processo de decomposição funcional, que foi adaptado para satisfazer as necessidades do projeto de *software*.

Outro conceito fundamental do projeto axiomático é o conteúdo de informação do projeto. Neste trabalho, é proposta uma adaptação das grandezas usadas para o cálculo deste conteúdo, para facilitar o uso em projetos de *software* orientado a objetos e para facilitar uma possível automação deste cálculo.

1 INTRODUÇÃO

O desenvolvimento de *software* tem sido objeto de estudos desde a criação do computador. Com os avanços tecnológicos e da eletrônica, o custo do *hardware* caiu drasticamente. Isto ocasionou um grande crescimento na utilização do computador, tanto pelas empresas quanto pela população em geral. O computador passou a ser usado como ferramenta corrente de trabalho, equipamento de comunicação e até mesmo como forma de entretenimento. Este fato ocasionou um crescimento muito grande na demanda por *software*, cada vez mais complexos e sofisticados.

Os programas de computador, antes construídos para serem executados poucas vezes, começaram a ter sua vida útil prolongada aumentando, portanto, a necessidade de alterações nos sistemas já existentes. Isto elevou os custos para desenvolver e, principalmente, para realizar correções e evoluções nos sistemas computacionais. Além disso, tem-se o fato de que se consegue desenvolver computadores cada vez melhores e mais baratos e, em contrapartida, não se consegue desenvolver *software* que satisfaça toda a demanda, tanto por tipos e quantidade de sistemas quanto por complexidade. Também não se consegue desenvolver *software* com custos compatíveis com os baixos custos de *hardware*. O processo de desenvolvimento de *software* teve que evoluir para fazer frente às pressões por redução dos custos e aumento da produtividade.

Neste contexto surgiram linguagens e abordagens de desenvolvimento como: metodologia Estruturada (GANE; SARSON, 1995), Análise Essencial (MCMENAMIM; PALMER, 1991), técnicas orientadas a objetos como a proposta por Booch (BOOCH, 1994), *Object Modeling Technique* (OMT) (RUMBAUGH et al., 1991), *Object-Oriented Software Engineering* (OOSE) (Jacobson, 1994), *Unified Modeling Language* (UML) (BOOCH, RUMBAUGH, e JACOBSON, 1999), *Rational Unified Process* (RUP) (JACOBSON, BOOCH e RUMBAUGH, 1999), LARMAN (1997) e, mais recentemente, *Extreme Programming*

(BECK, 1999), Catalisys (D'SOUZA; WILLS, 1998) e desenvolvimento orientado a aspectos (KICZALES et al., 1997). A partir destas técnicas, o desenvolvimento de *software* se tornou cada vez menos um processo baseado nas capacidades individuais ou talento dos desenvolvedores e mais um processo de engenharia, do ponto de vista do rigor dos métodos aplicados.

A teoria de projeto axiomático (SUH, 1990) surgiu como um processo para projetar produtos, peças e componentes, usado em diversas áreas da engenharia. Este processo é baseado em conceitos de independência entre os componentes usados no projeto e na minimização do conteúdo de informação do projeto. Estes conceitos se aplicam também no desenvolvimento de *software* e podem melhorar a qualidade do processo de desenvolvimento, garantindo a qualidade do produto desenvolvido. Com a teoria de projeto axiomático é possível conseguir parâmetros concretos para decidir qual a melhor solução para um problema entre diversas soluções possíveis.

1.1 OBJETIVOS

Os objetivos do trabalho de doutorado apresentado neste documento são:

- Aplicar a teoria de projeto axiomático ao projeto de *software* orientado a objetos.
- Utilizar a UML como linguagem de modelagem para o projeto axiomático de *software*.
- Identificar mecanismos de extensão para a UML e para a teoria de projeto axiomático aplicada ao desenvolvimento de *software.*
- Estabelecer um processo de desenvolvimento de *software* baseado na teoria de projeto axiomático usando a UML.
- Investigar os efeitos da utilização da metodologia estabelecida com relação a aspectos de qualidade de *software* e também com relação às novas

tendências na área de projeto de *software*.

A aplicação da teoria de projeto axiomático ao projeto de *software* orientado a objetos consiste basicamente em fazer uma associação entre os principais elementos do projeto de *software* orientado a objetos como os casos de uso, classes, responsabilidades, atributos e operações, e os elementos da teoria de projeto axiomático. Esta associação é necessária para que possam ser aplicados os princípios do projeto axiomático como os axiomas e teoremas, as matrizes de projeto e o processo de "ziguezagueamento" entre os domínios de projeto.

A utilização da UML como linguagem de modelagem para o projeto axiomático de *software* é a adaptação do processo de projeto axiomático aos mecanismos e à notação dos projetos orientados a objetos. A UML é a linguagem padrão para modelagem de sistemas orientados a objetos. Como a UML e a teoria de projeto axiomático foram criados a partir de diferentes abordagens, esta adaptação envolverá a identificação de meios de representação para a UML e para a teoria de projeto axiomático adequados ao projeto de *software*.

A identificação de mecanismos de extensão para a UML e para a teoria de projeto axiomático consiste na identificação da necessidade de adaptação dos seus conceitos. A UML e o projeto axiomático serão avaliados do ponto de vista das diferenças de aplicabilidade entre seus processos e ferramentas. Então, serão criados mecanismos que irão estender os já existentes, para que a UML e o projeto axiomático possam ser aplicados em conjunto.

É necessária a criação de um modelo de ciclo de vida adequado, muito próximo do ciclo de vida de um projeto de *software* orientado a objetos. Também é necessária a identificação de quais diagramas e modelos serão usados em cada fase, quais são os pré-requisitos de cada fase e quais são seus produtos. Tanto a teoria de projeto axiomático como a UML são aplicáveis em vários modelos de processos de desenvolvimento.

A idéia é aplicar a teoria axiomática ao desenvolvimento de sistemas computacionais, de tal forma que se tenha elementos de projeto mais independentes e mais

coesos. Para isso será necessário fazer uma associação entre os modelos e diagramas da UML e os da teoria axiomática. Isto envolve a possível criação de novos mecanismos para a UML e para a Teoria Axiomática.

Um dos objetivos deste trabalho é investigar os efeitos da utilização da metodologia estabelecida com relação a aspectos de qualidade de *software* e, também, com relação às novas tendências na área de projeto de *software*. Um dos aspectos importantes é a investigação sobre a influência da metodologia em aspectos de qualidade de *software* como manutenibilidade, adaptabilidade, portabilidade, reutilização, disponibilidade, usabilidade, eficiência e correção (PRESSMAN, 2004). Outro aspecto envolve a investigação sobre a possibilidade de desenvolvimento de ferramentas de *software* para auxiliar no processo de desenvolvimento, em termos de quais são e como seriam os itens automatizados. Também envolve a investigação da aplicabilidade da metodologia em conjunto com novas tendências na área de desenvolvimento de *software* como *Extreme Programming* (BECK, 1999), desenvolvimento orientado a aspectos (KICZALES et. al., 1997), padrões (GAMMA et al., 2000), entre outros.

1.2 MOTIVAÇÃO E JUSTIFICATIVA PARA O TEMA E OBJETIVOS PROPOSTOS

A motivação para este trabalho é tornar o processo de projeto de *software* mais preciso e confiável de forma a garantir sua qualidade. O projeto de *software* evoluiu muito desde o surgimento dos primeiros computadores, tendo surgido diversas técnicas, como as citadas no capítulo anterior. Mas, mesmo empregando corretamente uma metodologia, muitas decisões importantes de projeto não são baseadas estritamente na metodologia, mas sim na experiência pessoal e profissional do desenvolvedor. Esta experiência não garante que as decisões estejam corretas. Ao contrário, pode levar a modelos com erros, às vezes graves. Os modelos criados dessa forma podem comprometer a qualidade do *software* criado em termos de adaptabilidade, reutilização e portabilidade. Ao se realizar um projeto

de *software*, a estrita aplicação de uma das metodologias existentes, não garante a geração de um produto com qualidade. As metodologias atuais não possuem mecanismos para garantir a qualidade do modelo. Por isso, começou-se a adotar os padrões de projeto, que nada mais são do que a aplicação da experiência de outros desenvolvedores (LARMAN, 1997). Uma metodologia de desenvolvimento deve prover mecanismos para garantir que os produtos gerados através dela tenham a qualidade desejada.

No desenvolvimento de *software* orientado a objetos usando a UML ocorrem, freqüentemente, decisões de projeto tomadas sem parâmetros mais precisos que não a experiência do desenvolvedor. Entre elas estão a identificação e criação de classes e a atribuição de operações a classes.

A identificação e a criação de classes é feita a partir da descrição dos casos de uso, segundo processos orientados a objetos como o *Unified Process* (JACOBSON, BOOCH e RUMBAUGH, 1999) e o apresentado por LARMAN (1997). As classes são identificadas a partir de palavras ou expressões importantes na especificação dos casos de uso. O que ocorre muitas vezes é o fato de uma classe possuir operações que não seriam próprias para ela, existindo então a necessidade da criação de uma nova classe. Entretanto, a decisão sobre a criação desta classe normalmente é tomada de acordo com a experiência do projetista ou seguindo padrões de projeto de *software* como *General Responsibility Assignment Software Patterns* (GRASP) (LARMAN, 1997). Do mesmo modo, ocorre a atribuição de uma responsabilidade a uma classe. Às vezes, identifica-se uma responsabilidade a ser resolvida pelo sistema e esta precisa ser atribuída a uma classe. Novamente, usa-se a experiência do projetista ou padrões de projeto.

Na teoria de projeto axiomático temos ferramentas como os axiomas, teoremas e seus corolários, a matriz de projeto, a hierarquia funcional, o “ziguezagueamento” e os domínios de projeto (SUH, 1990), que ajudam na tomada de decisões sobre a qualidade do projeto de maneira mais precisa. A teoria de projeto axiomático, desenvolvida para projetos em engenharia industrial pode fornecer ferramentas para ajudar na solução dos problemas

relacionados com qualidade de projetos.

É importante prover um mecanismo para saber a ordem na qual os módulos de *software* devem ser projetados e quais módulos podem ser projetados em paralelo por equipes diferentes. Muitas vezes é necessária a divisão do trabalho em diferentes equipes de desenvolvimento. A teoria de projeto axiomático fornece ferramentas para tornar esta divisão mais eficiente e, também, formas para fazer com que as equipes trabalhem da maneira mais independente possível. Estas ferramentas são: os axiomas já citados, que vão permitir tornar os módulos mais independentes, e a matriz de projeto (SUH, 1990). Esta matriz fornece informações sobre qual parâmetro de projeto satisfaz determinado requisito funcional. Isto permite visualizar quais parâmetros de projeto estão relacionados com o mesmo requisito funcional e, portanto, saber quais são independentes. Esta independência funcional permite que as diferentes equipes possam realizar o desenvolvimento do *software*, independentemente. Caso contrário, haverá a necessidade de comunicação e sincronização entre as equipes durante o desenvolvimento. Isto elevará a duração e o custo do desenvolvimento.

Um aspecto importante para garantir a qualidade de *software* é a capacidade de rastrear os requisitos a partir de algum artefato que satisfaz este requisito (LEFFINGWELL, WIDRIG, 2003). Esta capacidade de rastreamento é garantida pela teoria de projeto axiomático através de ferramentas como a decomposição funcional, a hierarquia funcional e a matriz de projeto.

A introdução de mais níveis de abstração no projeto evita a tomada de decisão diretamente sobre itens de um nível de abstração muito mais baixo. Isto ocorre freqüentemente na criação dos diagramas de seqüência. Neste caso, o desenvolvedor passa de um nível de abstração alto (casos de uso) para um nível de abstração muito baixo, que é a troca de mensagens entre objetos. Assim, uma decomposição funcional ajudaria a evitar esta discrepância.

Durante a modelagem das classes acontece, freqüentemente, um problema que é

o de se obter classes que possuem responsabilidades que elas não deveriam ter. Este problema pode ser minimizado aplicando conceitos como herança e agregação. Mas mesmo assim, é necessário que o desenvolvedor possua experiência para evitar estes problemas. A metodologia proposta pode auxiliar a identificação dos métodos de cada classe através da identificação dos serviços técnicos (resultado da decomposição funcional), da matriz de projeto e do cálculo do conteúdo de informação, possibilitando uma tomada de decisão a respeito das classes com parâmetros mais concretos, obtidos a partir dos axiomas.

Segundo descrito em (PRESSMAN, 2004), existem várias métricas de projeto de *software*. Métricas baseadas na funcionalidade do sistema como pontos por função (*Function Points*) (VAZQUEZ; SIMÕES; ALBERT, 2003), métricas baseadas em características das classes como as métricas CK (CHIDAMBER e KEMERER, 1994), e métricas para o código fonte como as definidas em (HALSTEAD, 1977) e linhas de código. Estas métricas são baseadas na complexidade do produto a ser desenvolvido. Medem a complexidade das funções do *software*, a complexidade das classes, o tamanho da solução, entre outras. Não se consegue aferir a qualidade do projeto em si, mas apenas do produto. Usando o projeto axiomático pretende-se poder decidir sobre a qualidade do projeto em si com base em uma teoria estabelecida e não, somente, na experiência do projetista.

1.3 ORGANIZAÇÃO DO DOCUMENTO

Este trabalho apresenta a proposta de continuidade do trabalho de doutorado que está sendo realizado no programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) do Centro Federal de Educação Tecnológica do Paraná (CEFET-PR). São apresentados os resultados obtidos e as atividades a serem efetuadas para o término do trabalho.

No capítulo 1 é apresentada a introdução ao tema além dos objetivos deste trabalho e a motivação. No capítulo 2 é apresentado um resumo da teoria de projeto

axiomático além de trabalhos relacionados ao projeto axiomático e desenvolvimento de *software*. Nos capítulo 3 e 4 é apresentada a metodologia de desenvolvimento proposta neste trabalho de doutorado. No capítulo 5 são apresentadas próximas etapas a serem realizadas no trabalho de doutorado, o cronograma de realização e conclusões.

A metodologia de desenvolvimento proposta neste trabalho de doutorado é apresentada com base nos dois axiomas do projeto axiomático. No capítulo 3 são apresentados os aspectos relacionados com o axioma um como: a adaptação das etapas da decomposição funcional para o projeto de *software*, a adaptação dos principais conceitos do projeto axiomático para o projeto de *software* e as extensões à teoria propostas neste trabalho. No capítulo 4 é apresentado o cálculo do conteúdo de informação para a metodologia proposta neste trabalho.

2 TEORIA DE PROJETO AXIOMÁTICO

Este capítulo apresenta um resumo da teoria de projeto axiomático. A primeira seção apresenta uma introdução à teoria e suas definições básicas. A segunda seção apresenta os principais conceitos do projeto axiomático relacionados com o axioma da independência. Na terceira seção são apresentados os principais conceitos relacionados com o axioma da informação. Na quarta seção são apresentados trabalhos relacionados com a teoria de projeto axiomático e desenvolvimento de *software*.

2.1 INTRODUÇÃO E DEFINIÇÕES INICIAIS

Segundo NORLUND (1996), “Projetar é basicamente uma atividade de processamento de informação. O processo começa com a entrada da descrição das necessidades, requisitos e restrições, e termina com uma descrição completa do sistema que irá satisfazer estes requisitos, juntamente com a descrição de como construir este sistema” (NORLUND, 1996).

A pesquisa sobre projeto de engenharia começou, de forma mais sistemática, na Alemanha, na metade do século XIX. Muitas destas pesquisas foram compiladas em (BJÄRNEMO, 1983) e apresentadas em conjunto em (NORLUND, 1996). Entre elas estão teorias como a da Resolução Inventiva de Problemas (TRIZ) (ALTSHULLER, 1988), desenvolvimento com qualidade total de CLAUSING (1994), a ciência de projeto da escola WDK (*Workshop Design Konstruktion*) (HUBKA; EDER, 1992), o Projeto Robusto de TAGUCHI (1987) e a Teoria do Projeto Axiomático (SUH, 1990).

A teoria de projeto axiomático de Suh começou com a seguinte pergunta: “Dado um conjunto de requisitos funcionais para um determinado produto, existem axiomas de

aplicação genérica que levam a decisões corretas em cada passo da fabricação (desde a etapa de projeto até a montagem final e inspeção) para planejar um sistema de produção ótimo?" (SUH, BELL e GOSSARD, 1978). Para responder a esta pergunta foi elaborado um conjunto de axiomas que foram testados e avaliados em estudos de caso. Os 12 axiomas iniciais foram reduzidos a apenas dois axiomas e um conjunto de corolários e teoremas (SUH, 1990).

Os dois axiomas propostos são: o axioma 1, também chamado de axioma da independência, e o axioma 2, também chamado de axioma do conteúdo de informação. O primeiro axioma define um projeto "bom" como sendo aquele, onde os requisitos funcionais são independentes entre si. O segundo axioma estabelece que o "melhor projeto" é aquele que possui o menor conteúdo de informação (SUH, 1990).

Na teoria de projeto axiomático, o desenvolvimento é realizado através do mapeamento entre os domínios do cliente, funcional, físico e de processo (SUH, 2001). Este mapeamento é ilustrado na Figura 1. O domínio do cliente representa o conjunto das necessidades do cliente. Como exemplo de necessidades do cliente pode-se ter: performance desejada, satisfação dos clientes, atributos desejados, lucro, respostas desejadas do sistema.

O domínio funcional representa o conjunto dos requisitos funcionais do sistema que especificam as exigências impostas ao produto a partir das necessidades dos clientes. Neste domínio estarão representadas as funcionalidades desejadas. Como exemplo, pode-se ter: prover acesso ao interior, movimentar o veículo, suportar a carga, manter a carga imóvel. No caso de sistemas de *software*: registrar um cliente, registrar uma venda, mostrar o relatório de acessos, entre outros.

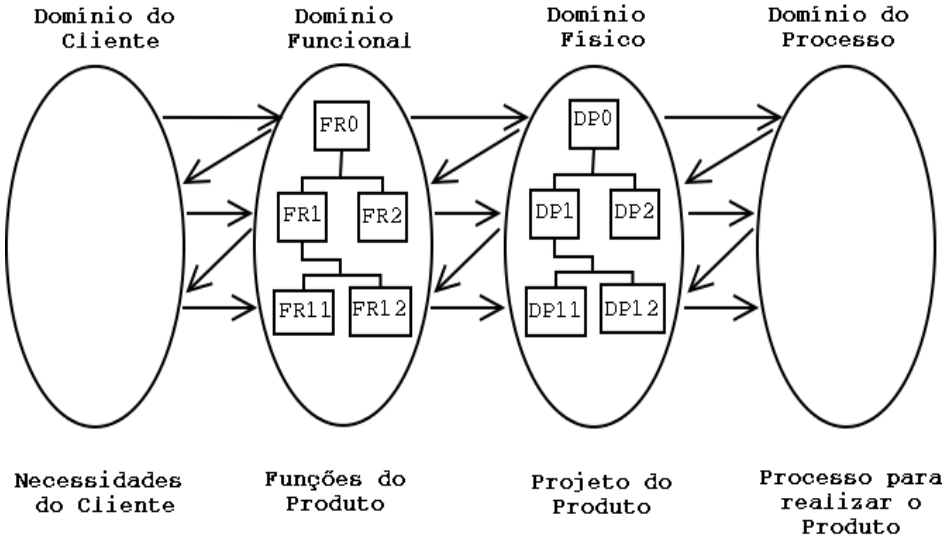
O domínio físico representa o conjunto dos parâmetros de projeto que satisfazem os requisitos funcionais. Os parâmetros de projeto são as variáveis físicas que são criadas de forma a satisfazer os requisitos funcionais. Como exemplo, pode-se ter: porta, motor, material de isolamento, eixos, paredes, suportes. No caso de sistemas de *software*: objetos,

funções, programas, algoritmos, entre outros. O domínio de processo é o conjunto dos processos, representados pelas variáveis de processo, que realizam os parâmetros de projeto (SUH, 2001). Como exemplo, pode-se ter, no caso de sistemas de *software*: linhas de código, variáveis, compiladores, entre outros.

Inicialmente, é feito o mapeamento entre o domínio do cliente e o domínio funcional. A seguir é feito o mapeamento entre o domínio funcional e o físico. Por fim é feito o mapeamento entre o domínio físico e o de processo. Em um processo de engenharia concorrente, entretanto, seria possível realizar um “ziguezagueamento” entre mais de dois domínios (STADZISZ, 1997).

Mapeamento é o processo que relaciona um elemento de um domínio com um, ou mais elementos do outro domínio. Por exemplo, o mapeamento entre o domínio funcional e o físico relaciona um requisito funcional com um parâmetro de projeto, usando a matriz de projeto.

FIGURA 1 DOMÍNIOS DO PROJETO AXIOMÁTICO



Uma definição mais formal para requisito funcional (FR, do inglês, *functional requirement*) é dada por SUH (2001). Os requisitos funcionais (FRs) são o conjunto mínimo de requisitos independentes que especificam completamente as necessidades do produto

no domínio funcional. Segundo SUH (2001), os parâmetros de projeto (DP, do inglês, *design parameter*) são as variáveis chaves no domínio físico que caracterizam que o projeto satisfaz os requisitos funcionais (FRs) especificados. Variáveis de processo (PV, do inglês, *process variable*) são as variáveis chaves no domínio de processo que caracterizam que o processo pode gerar, ou realizar, os parâmetros de projeto (DP) especificados. As Restrições (C, do inglês, *constraint*) são limites para as soluções aceitáveis. Existem dois tipos de restrições: de entrada e de sistema. As restrições de entrada são colocadas como parte das especificações do projeto. As restrições de sistema são impostas pelo sistema no qual a solução do projeto deverá funcionar (SUH, 2001).

2.2 AXIOMA 1 - AXIOMA DA INDEPENDÊNCIA

Apesar do desenvolvimento ser um processo que passa pelos quatro domínios, a atividade de projeto em si é representada pelos mapeamentos entre o domínio funcional e o domínio físico. Neste mapeamento, para cada requisito funcional (FR), é identificado um parâmetro de projeto (DP) correspondente, de forma a satisfazer o axioma 1 ou axioma da independência. O enunciado deste axioma é “Manter a independência dos requisitos funcionais” (SUH, 2001). Um enunciado alternativo para este axioma é “Em um *projeto* aceitável, os parâmetros de projeto (DPs) e os requisitos funcionais (FRs) estão relacionados de tal forma que um DP específico pode ser ajustado para satisfazer um FR sem afetar outros FRs” (SUH, 2001).

O axioma da independência estabelece que quando existem dois ou mais requisitos funcionais (FRs), a solução deve ser tal que cada requisito funcional (FR) deve ser satisfeito sem afetar os outros (SUH, 2001). Isto significa que o conjunto de parâmetros de projeto (DPs) deve ser identificado de forma a satisfazer os requisitos funcionais (FRs) e manter sua independência. A identificação de um conjunto de parâmetros de projeto (DP) que satisfaçam o axioma da independência nem sempre é possível. Neste caso, o conjunto

de requisitos funcionais (FRs) deve ser modificado, ou seja, deve ser encontrado um novo conjunto de requisitos funcionais (FRs), de forma a conseguir um projeto mais independente.

2.2.1 Matriz de projeto

Um dos instrumentos usados no projeto axiomático é a matriz de projeto. Ela representa o mapeamento entre dois domínios, principalmente entre o domínio funcional e o físico. Em uma das dimensões da matriz estão representados os requisitos funcionais (FRs) e na outra, os parâmetros de projeto (DPs). Cada elemento da matriz representa que um determinado parâmetro de projeto (DP) está relacionado a um determinado requisito funcional (FR). A relação que representa a matriz de projeto é apresentada na expressão (1) (SUH, 1990).

$$\{FR\} = [A]\{DP\} \tag{1}$$

A relação apresentada em (1) pode ser representada sob a forma de um sistema de equações, como em (2) (SUH, 1990). Neste sistema, os elementos A_{ij} representam o relacionamento entre o requisito funcional FR_i e o parâmetro de projeto DP_j . Por exemplo, se o elemento A_{12} for diferente de zero, significa que o parâmetro de projeto DP_2 é usado para satisfazer o requisito funcional FR_1.

$$\begin{aligned} FR_1 &= A_{11}\,DP_1 + A_{12}\,DP_2 + A_{13}\,DP_3 \\ FR_2 &= A_{21}\,DP_1 + A_{22}\,DP_2 + A_{23}\,DP_3 \\ FR_3 &= A_{31}\,DP_1 + A_{32}\,DP_2 + A_{33}\,DP_3 \end{aligned} \tag{2}$$

Do ponto de vista do axioma de independência existem três tipos de matrizes de projeto. Em (3), cada parâmetro de projeto (DP) é usado para satisfazer um único requisito funcional (FR). Esta configuração satisfaz completamente o axioma da independência. Neste caso, o projeto ou solução é dito “desacoplado”, do inglês, *uncoupled*. Este é considerado o melhor tipo de projeto, pois um ajuste em um parâmetro de projeto (DP) não

afeta outros requisitos funcionais (FRs) (SUH, 1990).

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix} \quad (3)$$

O tipo de projeto mais comum é o projeto dito “semi-acoplado”[1] (SUH, 1990). Neste caso, a matriz de projeto assume o formato de uma matriz triangular inferior como apresentado em (4) (SUH, 1990). Cada parâmetro de projeto (DP) é usado para satisfazer o requisito funcional (FR) correspondente e os requisitos funcionais (FRs) que estão representados na matriz, abaixo deste. Este fato indica que irá existir uma ordem na qual os parâmetros de projeto (DPs) deverão ser realizados.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix} \quad (4)$$

A situação a ser evitada em um projeto é quando os parâmetros de projeto não só estão relacionados com os requisitos funcionais (FRs) que estão representados na matriz, abaixo deste, mas, também, com os que estão representados na matriz, acima deste. Neste caso, a construção de cada parâmetro de projeto será dependente da construção de outros parâmetros de projeto, o que envolve muita sincronização no desenvolvimento. Não existe, portanto, uma ordem de na realização dos parâmetros de projeto (DPs) que resolva este problema. Este tipo de projeto é o chamado acoplado e a matriz assume um formato cheio, como em (5) (SUH, 1990).

[1] Traduzido do original *decoupled* (SUH, 1990)

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix} \quad (5)$$

Existem casos onde a matriz de projeto não assume a forma quadrada. O número de requisitos funcionais (FR) é maior que o número de parâmetro de projetos (DP). Isto resulta em um projeto acoplado. Segundo SUH (1990), este tipo de projeto é resultado de fatores que poderiam ser especificados como requisitos funcionais (FRs) e que não foram identificados. Nestes casos, deve-se identificar mais parâmetros de projeto (DP) para tornar a matriz desacoplada ou, pelo menos, semi-acoplada (SUH, 1990).

2.2.2 Medidas para o cálculo da independência funcional

É possível encontrar uma medida quantitativa de independência funcional. Em (SUH, 1990) são descritas duas medidas de independência funcional com base na matriz de projeto. São elas a “reangularidade”, R, (do inglês *reangularity*) e a “semangularidade”, S (do inglês *semangularity*). “Reangularidade” mede a ortogonalidade entre os parâmetros de projeto (DPs) e pode ser considerada como uma medida de interdependência entre os parâmetros de projeto (DPs) (OLEWNIK; LEWIS, 2003). Uma fórmula para o cálculo da “reangularidade” é apresentada na equação (6).

$$R = \prod_{\substack{i=1,n-1 \\ j=1+i,\ n}} \left(1 - \frac{\left(\sum_{k=1}^{n} A_{ki} A_{kj}\right)^2}{\left(\sum_{k=1}^{n} A_{ki}^2\right)\left(\sum_{k=1}^{n} A_{kj}^2\right)} \right)^{1/2} \quad (6)$$

A “semangularidade” mede o relacionamento angular entre os eixos correspondentes de parâmetros de projeto (DPs) e de requisitos funcionais (FRs) e pode ser considerada como sendo uma medida da correlação entre um requisito funcional (FR) e qualquer conjunto de parâmetros de projeto (DPs) (OLEWNIK; LEWIS, 2003). Uma fórmula

para o cálculo da “semangularidade” é apresentada na equação (7). Ambas, possuem um valor máximo de 1, que corresponde a um projeto desacoplado (ideal). Quando o nível de acoplamento aumenta, o valor de “reangularidade” e de “semangularidade” decresce (OLEWNIK; LEWIS, 2003).

$$S = \prod_{j=1}^{n} \left(\frac{\left| A_{jj} \right|}{\left(\sum_{k=1}^{n} A_{kj}^{2} \right)^{1/2}} \right) \tag{7}$$

Em (SUH, 2001) são enunciados alguns corolários e teoremas relacionados com a teoria de projeto axiomático. Estes teoremas e corolários são apresentados no anexo 1 deste documento.

2.2.3 Hierarquia funcional e decomposição funcional

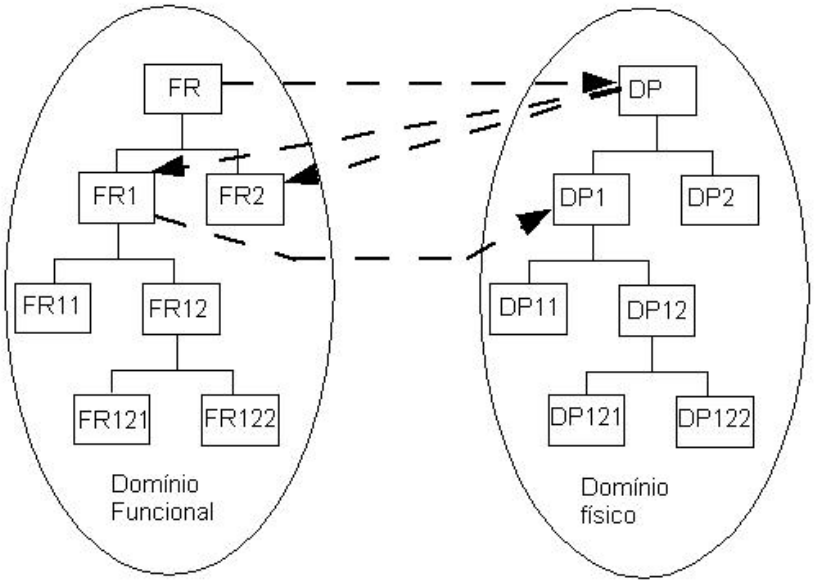
Quando os requisitos funcionais (FR) são identificados, eles fornecem as especificações necessárias para o projeto no nível conceitual. Mas, para a realização do projeto, é necessário um nível de detalhamento maior. É necessário decompor esses requisitos em unidades com maior detalhamento. Para isso, os requisitos funcionais (FR) são estruturados em uma hierarquia funcional. Esta hierarquia é obtida através da decomposição funcional de cada requisito funcional (FR) em sub-requisitos. A decomposição funcional é um processo no qual, após o mapeamento de todos os requisitos funcionais em parâmetros de projeto de um nível da hierarquia, cada requisito funcional (FR) é dividido em funcionalidades menores, chamadas sub-requisitos (sub-FR). Esta decomposição deve seguir algumas restrições, de forma a não alterar as relações de independência funcional representadas na matriz de projeto (TATE, 1999).

Para cada requisito funcional identificado em um determinado nível de decomposição, é identificado um parâmetro de projeto. Após a identificação de todos os parâmetros de projeto de um nível da hierarquia funcional, é feita a decomposição de cada requisito funcional em sub-requisitos, fazendo surgir um novo nível da hierarquia funcional.

Então, o processo é realizado novamente, agora para este novo nível. A decomposição é feita até que o projeto esteja completo (SUH, 2001).

Para cada novo nível de decomposição dos requisitos funcionais encontrados, é realizado um mapeamento entre os domínios e após o mapeamento é feita uma nova decomposição. Este processo de passar do domínio funcional para o domínio físico e novamente para o funcional em um outro nível da hierarquia é chamado de "ziguezagueamento" (SUH, 2001). A Figura 2 ilustra este processo.

FIGURA 2 "ZIGUEZAGUEAMENTO"



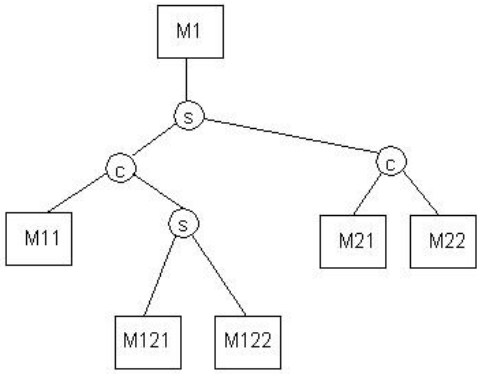
Não somente os requisitos funcionais (FRs) orientam a definição ou invenção dos parâmetros de projeto (DPs), mas também, os parâmetros de projeto orientam a decomposição dos requisitos funcionais. Isto acontece devido ao fato que para decompor os requisitos funcionais (FRs) é necessário que os parâmetros de projeto (DPs) correspondentes sejam identificados. A identificação do conjunto de sub-requisitos funcionais (sub-FR) terá como ponto de partida os parâmetros projeto (DPs) identificados no nível anterior.

2.2.4 Diagrama de fluxo e Diagrama de junção de módulos

Grandes sistemas são caracterizados por possuírem um grande número de componentes, sejam eles de *hardware* ou *software*. Para o projeto de grandes sistemas existe a necessidade de se entender e representar a arquitetura do sistema e cada um de seus módulos. Para este tipo de representação foram desenvolvidas duas ferramentas: o diagrama de fluxo e o diagrama de junção de módulos (SUH, 2001).

Um módulo é definido como sendo uma linha da matriz de projeto e contém o requisito funcional (FR) e os parâmetros de projeto (DP) que o satisfazem. O diagrama de junção de módulos representa o relacionamento entre os módulos através dos níveis da hierarquia. Um exemplo de diagrama de junção de módulos é apresentado na Figura 3. Neste diagrama, quando os módulos aparecem relacionados por um conectivo “s”, eles são independentes e a matriz de projeto que os relaciona é desacoplada. Quando os módulos estão relacionados por um conectivo “c”, existe uma relação de controle entre eles, o que significa uma matriz de projeto semi-acoplada (SUH, 2001).

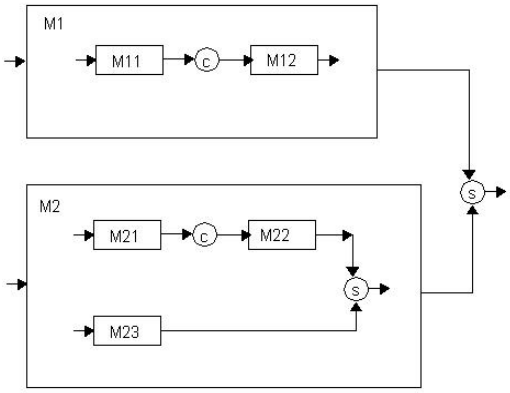
FIGURA 3 DIAGRAMA DE JUNÇÃO DE MÓDULO



O diagrama de fluxo permite fazer uma representação da arquitetura do sistema. Esta representação é baseada na independência entre os módulos. Quando a matriz de projeto é desacoplada os módulos são representados em paralelo, indicando a independência entre eles, e relacionados pelo conector “s”, como na Figura 4. Quando a matriz de projeto é semi-acoplada existe uma dependência, que é representada colocando

os módulos em seqüência, conectados por “c” (SUH, 2001). Neste tipo de representação não são consideradas matrizes de projeto acopladas. Neste caso o projeto deverá ser modificado. Um exemplo deste tipo de diagrama é apresentado na Figura 4.

FIGURA 4 DIAGRAMA DE FLUXO



2.3 AXIOMA 2 - AXIOMA DA INFORMAÇÃO

O axioma 2 estabelece que “o melhor projeto é um projeto funcionalmente desacoplado que tem o conteúdo de informação mínimo” (SUH, 1990). Ou, em um enunciado mais simples, o axioma 2 consiste em “Minimizar o conteúdo de informação do projeto” (SUH, 1990). O conteúdo de informação é um parâmetro importante para a definição de um bom projeto. Segundo SUH (2001), “o axioma da informação provê uma medida quantitativa de mérito de um dado projeto, além de prover uma base teórica para a otimização e robustez do projeto”. O conteúdo de informação pode ser usado como parâmetro para auxiliar na decisão de qual conjunto de parâmetros de projeto usar para satisfazer um conjunto de requisitos funcionais.

“Informação é a medida do conhecimento requerido para satisfazer um dado FR em um determinado nível da hierarquia de FR” (SUH, 1990, p. 65). Isto significa que quanto

mais informação for necessária para satisfazer um requisito funcional, maior será o conteúdo de informação. Por exemplo, supondo que uma determinada peça foi projetada para satisfazer o requisito funcional “cortar folhas de papel”. Se a tolerância de erro no corte for de um centímetro, não é necessária muita precisão no projeto e na fabricação da peça. Mas no caso da tolerância do corte ser de um centésimo de centímetro, serão necessários parâmetros mais precisos para o projeto da peça, além de testes mais elaborados e, principalmente, ferramentas de fabricação mais sofisticadas que exigem mais informação para sua operação. Neste caso, o conteúdo de informação é bem maior e a dificuldade de projeto e fabricação também. Assim, o conteúdo de informação está intimamente relacionado com a probabilidade de sucesso no projeto e fabricação de um produto. Esta relação é ilustrada em (SUH, 2001), onde é dito que “o axioma de informação estabelece que o projeto que tem a maior probabilidade de sucesso é o melhor projeto”.

O conteúdo de informação é calculado com base na probabilidade de que o parâmetro de projeto (DP) satisfaça o requisito funcional (FR) correspondente. Segundo SUH (1990), sendo *p* a probabilidade de que o DP satisfaça o FR, o conteúdo de informação *I* pode ser calculado como indicado na expressão (8):

$$I = \frac{1}{p} \tag{8}$$

A expressão (8) mostra que o conteúdo de informação é inversamente proporcional à probabilidade de sucesso. Para facilitar o cálculo de probabilidades relacionadas, o conteúdo de informação será calculado de forma logarítmica conforme apresentado em (SUH, 1990).

$$I = \log\left(\frac{1}{p}\right) \tag{9}$$

O inverso da probabilidade de sucesso pode ser interpretado de diversas formas. Uma forma de interpretá-la é como a dificuldade ou complexidade em projetar e construir o

produto. Por exemplo, quanto mais precisos forem os requisitos funcionais e restrições, mais difícil será projetar e construir um produto ou peça. Então, o inverso da probabilidade de sucesso é definido em termos da precisão requerida dos parâmetros de projeto. Neste caso, um parâmetro de projeto pode assumir uma determinada faixa de valores, chamada em (SUH, 1990) de *system range*. Para este mesmo parâmetro de projeto existe uma restrição para que os valores assumidos sejam considerados aceitáveis. Esta outra faixa de valores é chamada de "tolerância" (*tolerance*). Então, a dificuldade em projetar e fabricar o produto pode ser considerada em termos da razão entre a faixa de valores que o parâmetro pode assumir sobre a faixa de valores aceitáveis. Assim, foi proposto em (SUH, 1990) a fórmula mostrada na expressão (10) para o cálculo do conteúdo de informação.

$$I = \log\left(\frac{system\ range}{tolerance}\right) \tag{10}$$

O conteúdo de informação pode ser calculado como a composição dos conteúdos de informação dos requisitos funcionais do sistema. Esta composição é feita através da composição das probabilidades. Como o conteúdo de informação é calculado de forma logarítmica, a composição dos conteúdos de informação dos requisitos funcionais do sistema, se estes forem independentes, pode ser dada como a soma destes conteúdos[2].

$$I = -\sum_{i=1}^{n} p_i \log p_i \tag{11}$$

Neste caso o cálculo do conteúdo de informação não usa fatores ponderados. Um dos motivos é que se forem usados fatores ponderados na soma do conteúdo de informação, este não representará mais a probabilidade total de sucesso. Também é considerado que o cálculo da informação do requisito funcional (FR) é baseado na faixa de variação de projeto deste requisito funcional. Segundo SUH (2001, p 42), "Se esta faixa de variação for corretamente especificada, ela já definirá a importância relativa de cada

[2] ver Teorema 13, no anexo 1

requisito funcional (FR)".

2.4 TRABALHOS RELACIONADOS

Nesta seção serão apresentados trabalhos que se relacionam com a teoria de projeto axiomático e com projeto de *software* orientado a objetos, passando por temas como ciclo de vida, decomposição funcional e teoria de projeto.

2.4.1 Projeto axiomático de *software* orientado a objetos

Um dos primeiros trabalhos envolvendo projeto de *software* orientado a objetos usando o projeto axiomático foi apresentado por DO e SUH (1999). Neste trabalho, a teoria axiomática é usada para o desenvolvimento de programas orientados a objetos. É proposto que, partindo das necessidades do cliente, deve-se encontrar os componentes a serem implementados, definindo assim as classes e objetos que farão parte do sistema (DO e SUH, 1999).

HINTERSTEINER e NAIN (1999) usaram a teoria de projeto axiomático para o desenvolvimento de *software* com foco em sistemas integrados de *software* e *hardware*. É proposta uma metodologia para facilitar o desenvolvimento de *software* de controle em conjunto com seu *hardware* correspondente. Cada requisito funcional é mapeado em um item do domínio de *software* e um item do domínio de *hardware*, o mesmo acontecendo com cada parâmetro de projeto. (HINTERSTEINER e NAIN, 1999)

Em (SUH e DO, 2000) é apresentada uma abordagem para projeto de *software* orientado a objetos chamada "*Axiomatic Design of Object-Oriented Software Systems*" (ADo-oSS). Trata-se de uma abordagem baseada na metodologia OMT (RUMBAUGH et al., 1991). Neste trabalho não são utilizados conceitos da UML (BOOCH, RUMBAUGH, e JACOBSON, 1999) como, por exemplo, casos de uso. É apresentado um mapeamento entre a matriz de projeto e o diagrama de classes onde os requisitos funcionais (FRs)

representam objetos ou classes, os parâmetros de projeto representam os atributos destes objetos e as células da matriz representam os métodos dos objetos (SUH e DO, 2000).

O trabalho apresentado em (SUH e DO, 2000) serviu de base para o capítulo 5 do livro "*Axiomatic Design: Advances and Applications*" (SUH, 2001), onde é descrita com detalhes a metodologia ADo-oSS. É apresentada com detalhes a utilização da matriz de projeto, do diagrama de classes, do "ziguezagueamento", do diagrama de junção de módulos e o diagrama de fluxo e as relações entre estes artefatos. Além disso, é apresentado como estudo de caso o projeto do sistema *ACCLARO*[3].

A ferramenta de projeto *ACCLARO* foi desenvolvida para auxiliar na elaboração de projetos usando a teoria de projeto axiomático. Ela foi dividida em duas ferramentas, o *ACCLARO Designer* e o *ACCLARO Scheduller*. O *ACCLARO Designer* permite que o projeto de produtos e sistemas seja feito com o auxílio desta ferramenta de *software*. Permite a decomposição funcional e constrói automaticamente a matriz de projeto. Permite, também, o rastreamento dos requisitos funcionais devido a mudanças, a análise de acoplamento do projeto, entre outras características. O *ACCLARO Scheduller* é uma ferramenta integrada com o *IBM Rational Rose*[4] para gerenciar o andamento do projeto. Permite, entre outras coisas, identificar o impacto das mudanças de projeto, gerar um plano de desenvolvimento para o projeto, estimar custos e identificar possíveis gargalos no andamento do projeto.

Em (DO, 2004) foi apresentada a utilização da teoria de projeto axiomático para o gerenciamento do processo de desenvolvimento de *software* visando garantir a qualidade do processo e do produto. É apresentada uma extensão da teoria de projeto axiomático, que consiste em domínios complementares. Estes domínios complementares são usados para representar conceitos próprios do domínio do projeto. Um domínio complementar importante é o domínio de casos de uso usado para modelar os requisitos funcionais. Além disso, é apresentada a utilização da matriz de estrutura de projeto (DSM) (ULRICH e EPPINGER,

[3] *ACCLARO* é marca registrada de *Axiomatic Design Software, Inc.*.
[4] IBM *Rational Rose* é marca registrada de IBM

2003) para possibilitar análises de interdependência dentro de um mesmo domínio (DO, 2004).

2.4.2 Decomposição funcional

Baseado, também, na estrutura de informação de NORLUND (1996), TATE e NORLUND (1996) criaram um modelo de ciclo de vida de projeto para o projeto axiomático. O modelo apresentado contempla características como tomada de decisão, métricas de desempenho, iterações, seqüência de atividades, níveis de abstração e gerenciamento de informação. O modelo tem como objetivo servir como um guia para qualquer processo de projeto de produto (TATE; NORLUND, 1996).

Um requisito funcional pode ser decomposto em sub-requisitos de vários tipos. Segundo HINTERSTEINER (1999), eles podem ser classificados em funções de processo, de comando e controle, e de suporte e integração. As funções de processo efetuam o processamento propriamente dito dos operandos além de prover o transporte destes através do sistema. As funções de comando e controle representam a lógica necessária para coordenar os diferentes processos no subsistema. As funções de suporte e de integração mantém o subsistema funcionando de forma coordenada (HINTERSTEINER, 1999).

Em (TATE, 1999) foi definida uma seqüência de etapas a serem consideradas para realizar as decomposições funcionais durante um projeto. Além disso, foram definidas restrições a serem consideradas para manter a coerência entre as decomposições, nos diversos níveis da hierarquia funcional, durante o processo de “ziguezagueamento” (TATE, 1999). Essas restrições dizem respeito à escolha apropriada dos sub-requisitos funcionais (sub-FR) de tal forma que a matriz de projeto mantenha o mesmo grau de acoplamento do nível anterior. Em outras palavras, se a matriz de projeto do nível anterior era desacoplada, ela manterá essa característica no próximo nível.

Em (SCHREYER e TSENG, 2000) é apresentada uma notação para representar a decomposição funcional e o processo de “ziguezagueamento”. Esta notação é baseada nos *statecharts* apresentados em (HAREL, 1987). Esta notação ajuda a identificar estados

principais, além de outros parâmetros importantes, como tempo, sinais de sensores e informação de memória compartilhada. É apresentada a matriz de transições de estados, que relaciona as condições de entrada com as ações de saída para cada estado (SCHREYER e TSENG, 2000).

Em (CAPPETTI, NADDEO e PELLEGRINO, 2004) é apresentada uma metodologia para reduzir o acoplamento da matriz de projeto. Esta metodologia é baseada na lógica *Fuzzy* e na teoria da representação (ZADEH, L. A. et al., 1974). É proposto que associando-se valores de função de pertinência para cada célula da matriz de projeto, é possível reduzir o grau de dependência entre parâmetros de projeto (DP) e requisitos funcionais (FR). Com Isto é possível reduzir o grau de acoplamento da matriz de projeto, otimizando um projeto acoplado, quando não é possível obter um projeto desacoplado (CAPPETTI, NADDEO e PELLEGRINO, 2004).

2.4.3 Tópicos Gerais em Teoria de Projeto

Em (NORLUND, 1996) foi estabelecida uma estrutura de informação para o projeto axiomático. Esta estrutura de informação permite identificar quais as fontes de informação que o projetista irá usar durante o processo de projeto axiomático e como a informação obtida durante o processo pode ser usada nos estágios finais do projeto (NORLUND, 1996).

HARUTUNIAN; NORLUND e TATE (1996) estudaram a tomada de decisões de projeto baseada na teoria de projeto axiomático usando a estrutura de informação de projeto apresentada em (NORLUND, 1996) e criaram um *software* de desenvolvimento de produtos que usa a teoria de projeto axiomático para tomar decisões de projeto.

Quando decisões de projeto são tomadas, podem surgir conseqüências para estas decisões. Segundo LINDHOLM, TATE e HARUTUNIAN (1999) estas conseqüências podem gerar a necessidade de subsistemas adicionais ao produto. Por exemplo, se um parâmetro de projeto escolhido gera calor como conseqüência, aparecerá a necessidade de um subsistema de resfriamento. Em (LINDHOLM, TATE e HARUTUNIAN, 1999) foi apresentada

uma classificação das conseqüências, em conseqüências originadas de parâmetros de projeto, de variáveis de processo ou da configuração. Além disso, foi apresentado um processo para identificar estas conseqüências e gerenciá-las (LINDHOLM, TATE e HARUTUNIAN, 1999).

Em "*On a Mathematical Foundation of Axiomatic Design"*, RUDOLPH (1996) derivou a teoria de projeto axiomático com um caso especial da chamada hipótese de avaliação. A hipótese de avaliação é baseada na técnica de análise dimensional conhecida da física e que permite a derivação direta de expressões que em casos especiais correspondem aos axiomas de independência e informação (RUDOLPH, 1996).

Em (SHIN et al., 2004) são apresentadas formas de se calcular o conteúdo de informação de um projeto. São apresentados dois métodos, o método gráfico, para projetos com dois requisitos funcionais e o por integração, para projetos com mais de dois requisitos funcionais. O cálculo é feito com base na associação entre o *system range* e o *functional range*. O método gráfico usa uma distribuição uniforme de probabilidade para a associação. No método por integração podem ser usados quaisquer tipos de distribuição de probabilidade (SHIN et al., 2004).

3 METODOLOGIA DE DESENVOLVIMENTO PROPOSTA

Neste capítulo é feita a descrição da metodologia de projeto de *software* proposta neste trabalho. É apresentada uma analogia entre os conceitos da teoria de projeto axiomático e os conceitos de projeto de *software* orientado a objetos usando UML. É feita uma descrição dos domínios complementares que serão usados nesta metodologia, em adição aos domínios do projeto axiomático. Então é apresentada uma descrição de cada uma das etapas da metodologia indicando os artefatos originados e as ações a serem executadas.

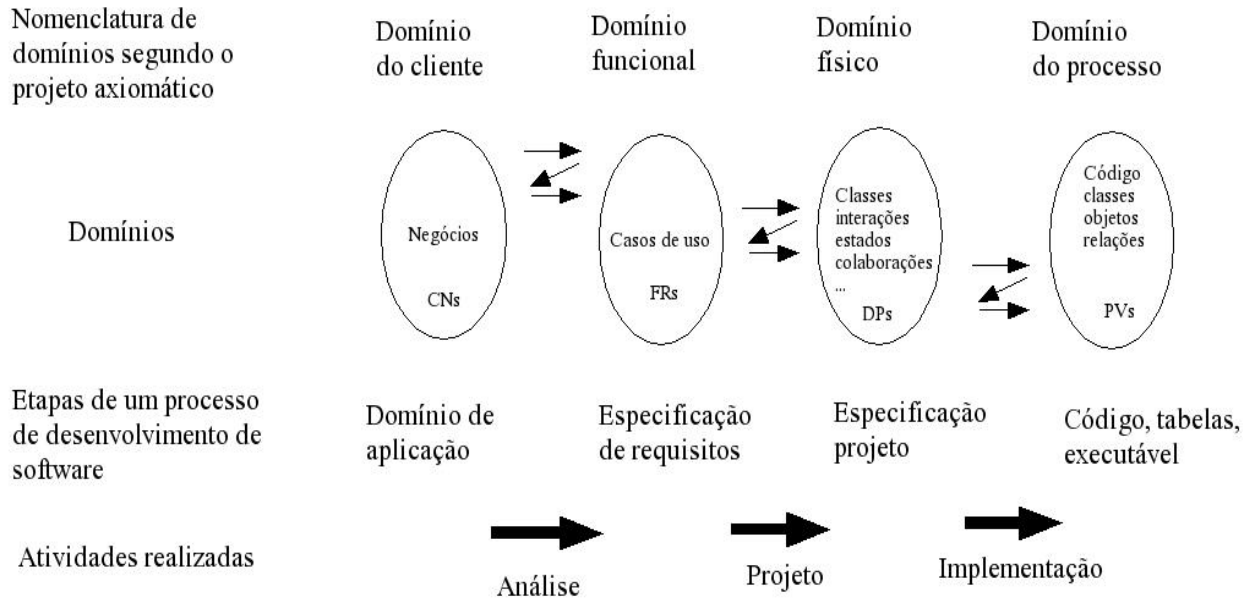
3.1 RELACIONAMENTO ENTRE PROJETO AXIOMÁTICO E PROJETO DE *SOFTWARE*

Como apresentado no capítulo anterior, a teoria de projeto axiomático é uma teoria que se aplica a todo o tipo de projeto, desde o projeto de um carro até o projeto de um departamento acadêmico (SUH, 2001). Para a sua aplicação no desenvolvimento de *software* orientado a objetos, existe a necessidade de se estabelecer uma relação entre os principais conceitos do projeto axiomático e os principais conceitos de projeto de *software*. Na Figura 5 são mostradas estas relações. No projeto axiomático existem quatro domínios principais que podem ser entendidos no projeto de *software* como sendo o modelo ou conjunto de modelos que estão sendo desenvolvidos em cada fase.

De acordo com a Figura 5, o domínio do cliente, que representa as necessidades do cliente no projeto axiomático, corresponde ao modelo do negócio (*business model*) do projeto de *software*. O domínio funcional, que representa os requisitos funcionais, corresponde à especificação de requisitos no projeto de *software*. O domínio físico, que representa os parâmetros de projeto, corresponde aos modelos de projeto de *software*,

como modelo de classes, modelo de interações e modelo de estados. O domínio de processo representa as variáveis de processo que correspondem aos modelos relativos à implementação e implantação, como o código fonte.

FIGURA 5 RELAÇÃO ENTRE CONCEITOS DO PROJETO AXIOMÁTICO E DE PROJETO DE *SOFTWARE*



Além das correspondências entre os domínios e os modelos, existe a correspondência entre as fases do processo de desenvolvimento e os mapeamentos (“ziguezagueamento”) entre os domínios. O mapeamento entre os domínios do cliente e funcional corresponde, no projeto de *software*, à atividade de análise de requisitos. Nesta atividade, as necessidades dos clientes, usuários e *stakeholders* (BITTNER e SPENCE, 2003) são identificadas e mapeadas em requisitos funcionais e requisitos não funcionais. Segundo LEFFINGWELL e WIDRIG (2003), o modelo do negócio é usado para auxiliar na definição do sistema e suas aplicações. O negócio representa o conjunto das atividades, entidades e seus relacionamentos para os quais o sistema computacional será implementado.

O mapeamento entre os domínios funcional e físico corresponde à atividade de

projeto propriamente dito. Nesta atividade, partindo dos requisitos funcionais do sistema, são identificados os parâmetros de projeto que satisfazem estes requisitos. A especificação de requisitos de um sistema é usada para definir quais funcionalidades o sistema deverá oferecer e, também, serve de base para validações e testes. A partir dos requisitos funcionais identificados são construídos os parâmetros de projeto. Estes parâmetros podem ser processos, classes, objetos, funções, estados. Neste caso, cada parâmetro de projeto pode estar relacionado com um ou mais requisitos funcionais.

O mapeamento entre o domínio físico e de processo corresponde à atividade de implementação. Os parâmetros de projeto, classes, objetos, interações e estados são usados como base para a implementação do *software*.

3.2 DOMÍNIOS COMPLEMENTARES

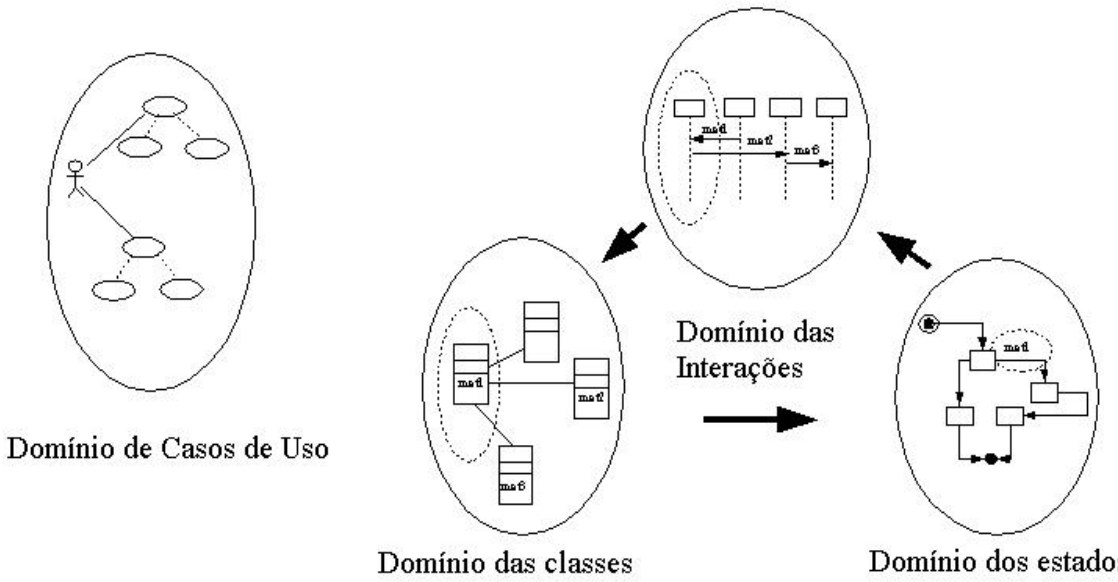
Os domínios básicos do projeto axiomático não são suficientes para representar todas as informações utilizadas em um projeto de *software* que utiliza o processo unificado (JACOBSON, BOOCH e RUMBAUGH, 1999). De acordo com (DO, 2004) podem ser adicionados domínios complementares ao processo. Estes domínios representam características e informações específicas da natureza do projeto. No caso de projeto de *software* serão usados os domínios de casos de uso, de classes, de estados e de interações. Os domínios complementares serão usados em conjunto com os domínios básicos do projeto axiomático.

Pela importância do mapeamento entre o domínio funcional e o físico, como sugerido em (DO, 2004), estes dois domínios serão tratados como domínios abstratos e o mapeamento entre eles será usado para a aplicação dos axiomas 1 e 2 (SUH, 1990) e dos principais conceitos do projeto axiomático. Para isso será feito um mapeamento direto entre estes domínios e os domínios complementares.

De acordo com a Figura 6 serão utilizados os seguintes domínios

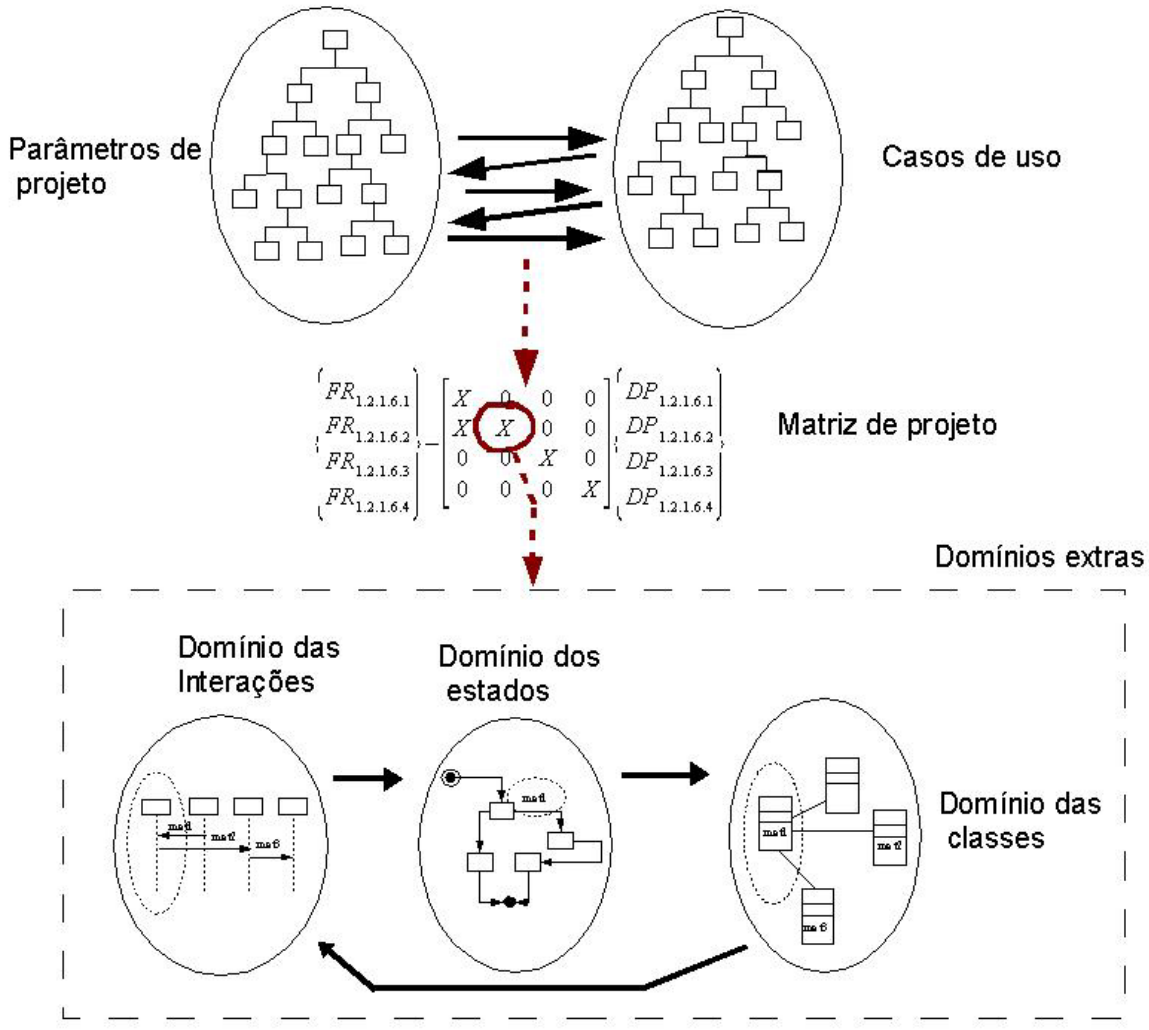
complementares: o domínio de casos de uso, o domínio de classes, o domínio de interações e o domínio de estados. O domínio de casos de uso será composto pelo modelo de casos de uso (JACOBSON, BOOCH e RUMBAUGH, 1999). O modelo de casos de uso é formado pelo diagrama de casos de uso e as especificações de caso de uso, e é usado para identificar os requisitos funcionais do sistema. O domínio de classes é formado pelo diagrama de classes. Este domínio representa aspectos estruturais das classes do sistema, como as classes, seus atributos, seus métodos e os relacionamentos entre elas (BOOCH, RUMBAUGH, e JACOBSON, 1999). O domínio de interações é formado principalmente pelos diagramas de seqüência. Este domínio representa principalmente a interação entre os objetos do sistema de forma a satisfazer as funcionalidades desejadas. O domínio de estados é formado pelos diagramas de estado. Este domínio representa as mudanças de estados dos objetos do sistema causadas, entre outros, pelas interações entre os objetos. Os domínios de classes, interações e estados serão usados como auxílio na representação dos parâmetros de projeto.

FIGURA 6 DOMÍNIOS COMPLEMENTARES SUGERIDOS



O mapeamento entre os domínios do cliente e funcional será feito através do domínio de casos de uso, como sugerido por DO (2004). Neste caso haverá uma relação direta entre cada requisito funcional (FR) do domínio funcional e um caso de uso do domínio de casos de uso. Os requisitos funcionais serão mapeados em casos de uso, subcasos de uso, atividades e serviços técnicos.

FIGURA 7 DOMÍNIOS DE CLASSES, DE INTERAÇÕES E DE ESTADOS



Os parâmetros de projeto serão mapeados em colaborações, sub-colaborações que são decompostas até atingir o nível onde serão representados os objetos que fazem parte da colaboração. Este mapeamento será feito usando a matriz de projeto. Nesta matriz

cada uma das células representará um item de projeto que é composto por um conjunto de parâmetros resultantes da composição das visões estática (domínio de classes) dinâmica (domínio de estados) e de interação (domínio de seqüência). A relação entre o mapeamento dos requisitos funcionais em parâmetros de projeto e os domínios de classes, de interações e de estados está ilustrada na Figura 7.

3.3 DESCRIÇÃO DAS ETAPAS DA METODOLOGIA PROPOSTA

Nesta seção será feita uma descrição das etapas da metodologia de desenvolvimento de *software* proposta neste trabalho. Esta descrição será feita com base nas atividades executadas durante o processo de desenvolvimento de *software*. As etapas são basicamente as atividades de mapeamento entre os domínios do projeto, incluindo os domínios básicos do projeto axiomático e os domínios complementares próprios do projeto de *software*. Neste trabalho serão consideradas três etapas: o mapeamento entre os domínios do cliente e funcional, o mapeamento entre os domínios funcional e físico e o mapeamento entre o domínio físico e de processo. Fazendo-se uma analogia com o processo unificado, o mapeamento entre os domínios do cliente e funcional corresponde à fase de elaboração, a fase de construção corresponde aos mapeamentos entre os domínios funcional e físico e entre o domínio físico e de processo.

3.3.1 Mapeamento entre domínios do cliente e funcional

Na primeira etapa são identificados os requisitos funcionais (FR) a partir das necessidades dos clientes (CN). Esta etapa corresponde à análise de requisitos em um processo de desenvolvimento *software* convencional. As necessidades dos clientes podem ser representadas através do modelo de negócio. Com este tipo de modelo é possível representar o processo a ser automatizado e como este se enquadra no processo da organização. O modelo de negócio permite avaliar como o futuro *software* irá afetar os

usuários e o negócio em si (LEFFINGWELL e WIDRIG, 2003).

Segundo apresentado em (DO, 2004), “o modelo do processo axiomático recomenda que seja estabelecido um domínio de casos de uso para modelar os aspectos dinâmicos dos requisitos em *software*”. Segundo apresentado em (SUH, 1990), os requisitos funcionais (FR) são definidos como sendo o conjunto mínimo de requisitos independentes que caracterizam completamente o objetivo do projeto para uma necessidade específica. Uma forma de se encontrar um conjunto mínimo de requisitos funcionais independentes é através dos casos de uso. Estes requisitos funcionais são identificados a partir das necessidades do cliente e/ou dos *stakeholders*. Entre as necessidades dos clientes são encontrados requisitos não funcionais, requisitos funcionais e restrições. A decomposição baseada em casos de uso permite que sejam identificadas as principais funcionalidades do sistema (serviços desejados), separando-as de requisitos funcionais de menor importância.

3.3.2 Mapeamento entre domínios funcional e físico.

A partir do momento em que os requisitos funcionais estejam identificados como casos de uso e validados com o cliente, é dado início à etapa que, segundo apresentado em (SUH, 1990), corresponde à atividade de projeto propriamente dita. Esta etapa é o mapeamento entre o domínio funcional e o domínio físico. Como visto anteriormente, este mapeamento é feito a partir dos requisitos funcionais (FR), obtendo-se os parâmetros de projeto (DP).

A decomposição funcional realizada nesta etapa será dividida em níveis, de acordo com o tipo de resultado da decomposição. A cada nível de decomposição podem ser feitas quantas decomposições forem necessárias para que o projeto seja satisfeito para este nível de abstração. Cada nível será caracterizado pelo tipo de resultado da decomposição. No primeiro nível são obtidos os casos de uso. No segundo nível são obtidos os subcasos de uso. No terceiro, são obtidas as atividades e no quarto nível, os serviços técnicos.

Na metodologia proposta neste trabalho a decomposição funcional será feita de acordo com as características de um projeto de *software*. No primeiro nível de

decomposição são feitas decomposições até que todos os casos de uso sejam identificados. A partir dos casos de uso, identificados na etapa anterior, são realizadas decomposições funcionais de forma a obter os subcasos de uso no segundo nível de decomposição. A partir dos subcasos de uso, são realizadas decomposições e chega-se ao terceiro nível de decomposição onde são obtidas as atividades, ou segundo a definição apresentada em (BOOCH, RUMBAUGH, e JACOBSON, 1999), o chamado fluxo de eventos. De cada atividade são efetuadas decomposições funcionais e são encontradas funções de serviço (STADZISZ, 1997) ou, como serão chamados neste trabalho, os serviços técnicos. Este é o quarto nível de decomposição.

A teoria de projeto axiomático propõe que a decomposição seja feita em amplitude (SUH, 2001). Isto significa que antes de ser efetuada uma decomposição funcional, todos os parâmetros de projeto deste nível já deverão ter sido identificados e mapeados na matriz de projeto.

A decomposição pode ser realizada também em profundidade. Isto significa que um caso de uso pode ser decomposto até que seja atingido o nível de serviços técnicos, sem a necessidade da decomposição dos outros casos de uso ter sido realizada. Este fato impede que a matriz de projeto completa de cada nível de decomposição seja obtida, mas podem ser obtidas matrizes parciais, para cada ramo da árvore de hierarquia funcional, a cada nível, que servirão para avaliar parcialmente a independência funcional. Esta avaliação parcial pode ser útil já que no primeiro nível foram identificadas as dependências funcionais entre os casos de uso, e esta dependência será refletida na decomposição.

3.3.2.1 Primeiro nível de decomposição.

Neste nível de hierarquia estão os casos de uso representando os requisitos funcionais (FRs). Então, para cada requisito funcional (FR) serão identificados os parâmetros de projeto (DPs) correspondentes, que neste caso será representado por uma colaboração. De acordo com a UML 2.0 (OBJECT MANAGEMENT GROUP, 2003) uma colaboração descreve uma estrutura de elementos (papéis) que colaboram entre si para

realizar uma determinada funcionalidade. A colaboração é útil, pois seus detalhes podem ser mostrados ou ocultados dependendo do nível de abstração desejado. Para este nível, uma colaboração irá representar toda uma estrutura de papéis para realizar um caso de uso. Esta estrutura pode ser decomposta em sub-colaborações em outros níveis. Com esse mapeamento obtém-se uma matriz de projeto onde as linhas representam casos de uso e as colunas representam colaborações.

Quando este mapeamento é feito, são identificadas algumas relações entre os casos de uso (FR) que não são identificadas durante a decomposição funcional. Por exemplo, um caso de uso “mostrar relatório com os dados do cliente” é dependente de um outro caso de uso “cadastrar cliente”. Isto indica uma relação de dependência de projeto. Esta dependência é resolvida no processo unificado (JACOBSON, BOOCH e RUMBAUGH, 1999), de maneira informal, através de uma ordenação de prioridades dos casos de uso (LARMAN, 1997).

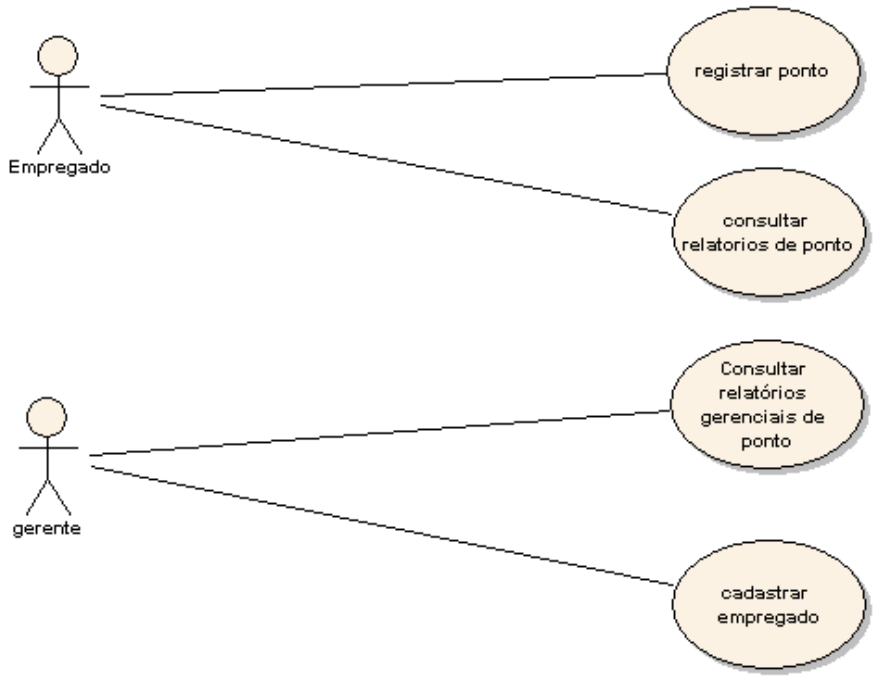
TABELA 1 MATRIZ DE PROJETO DE PRIMEIRO NÍVEL

	Colaboração cadastrar empregado	Colaboração registrar ponto	Colaboração consultar relatórios gerenciais	Colaboração consultar relatórios de ponto
1.1 Cadastrar Empregado	X			
1.2 Registrar Ponto	R	X		
1.3Consultar relatórios gerenciais de ponto	R	R	X	
1.4 Consultar relatórios de Ponto	R	R		X

Esta dependência entre casos de uso é assinalada na matriz de projeto, como na Tabela 1. Como exemplo é mostrado na 0 o modelo de casos de uso identificados para um sistema de registro de pontos. Para este sistema, neste nível de decomposição funcional, é

construída a matriz de projeto, na qual as linhas representando os casos de uso e as colunas representando colaborações. Esta matriz de projeto é mostrada na Tabela 1.

FIGURA 8 CASOS DE USO DE UM SISTEMA DE REGISTRO DE PONTOS



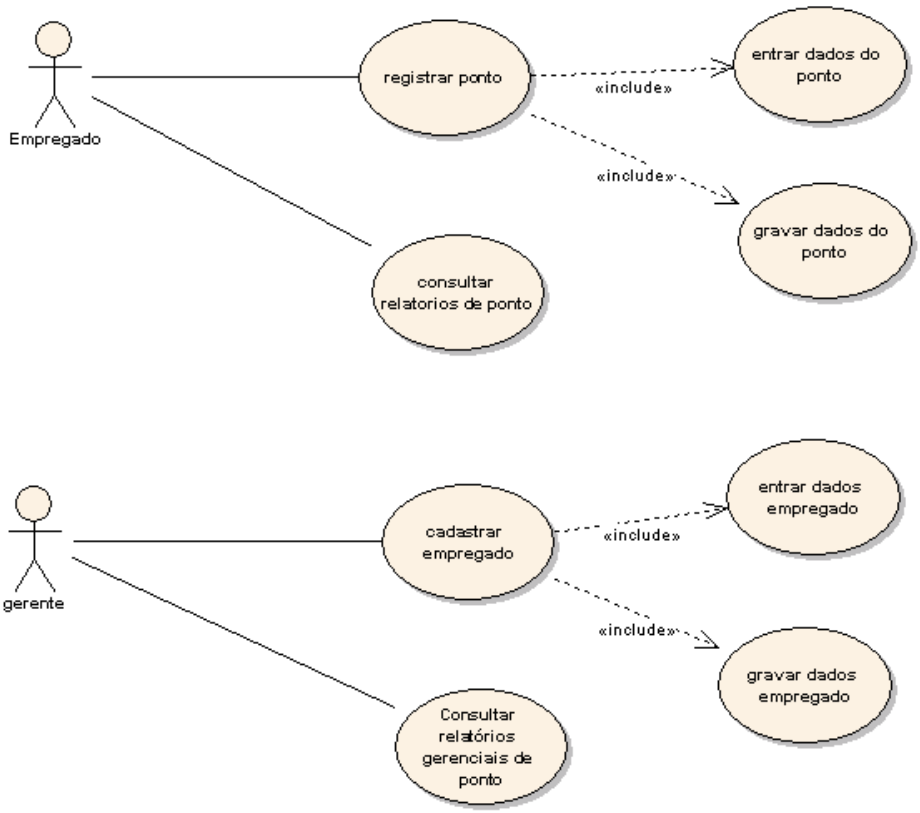
Na matriz de projeto da Tabela 1, o símbolo “R” representa que uma determinada FR irá usar na colaboração (DP) uma classe (papel da colaboração) já usada em outra colaboração. No caso ilustrado na Tabela 1, para se fazer o projeto do caso de uso registrar ponto é necessário que dados da colaboração cadastrar empregado já estejam definidos. Ou seja, existe uma dependência de projeto entre os dois casos de uso. Portanto, haverá a necessidade de existir uma ordem na qual os casos de uso serão implementados, que é resultante do fato da matriz de projeto ser semi-acoplada.

3.3.2.2 Segundo nível de decomposição

No 2º nível de decomposição, são identificados o que será chamado neste trabalho de “subcasos de uso”. No nível hierárquico anterior tem-se casos de uso, serviços completos prestados pelo sistema aos atores. Cada um destes serviços pode ser

decomposto em unidades de serviço chamadas subcasos de uso. Um subcaso de uso é um serviço prestado pelo sistema, que não é completo e está incluso em algum caso de uso. É uma parte da funcionalidade do caso de uso. Nem todos os casos de uso serão decompostos em subcasos, pois apresentam apenas uma funcionalidade menor, como se pode ver na Figura 9.

FIGURA 9 SUBCASOS DE USO



Na Tabela 2 é apresentada a matriz de projeto referente ao segundo nível da hierarquia funcional para o exemplo apresentado na Figura 9. Nesta tabela é possível ver que existe uma dependência funcional entre alguns subcasos de uso. Por exemplo, o subcaso de uso “entrar dados de empregado” é mapeado para a colaboração “entrar dados de empregado” que usa o mesmo papel que a colaboração “gravar empregado”. Esta dependência está de acordo com o mapeamento do nível anterior, apresentado na Tabela 1.

TABELA 2 MATRIZ DE PROJETO DE 2^{O}. NÍVEL

	Colaboração entrar dados de empregado	Colaboração gravar empregado	Colaboração entrar dados de ponto	Colaboração gravar ponto	Colaboração consultar relatórios gerenciais ponto	Colaboração consultar relatórios
1.1.1 entrar dados de empregado	X					
1.1.2 gravar empregado	R	X				
1.2.1 entrar dados de ponto	R		X			
1.2.2 gravar ponto	R		R	X		
1.3.1Consultar relatórios gerenciais de ponto	R		R		X	
1.4.1 Consultar relatórios de Ponto	R		R			X

3.3.2.3 Terceiro nível de decomposição

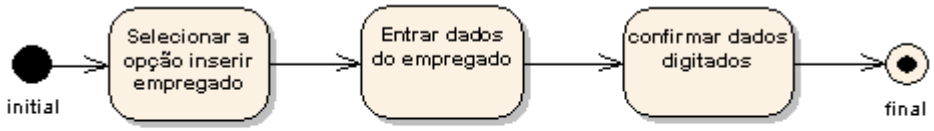
Os fluxos de eventos são uma forma de descrever os casos de uso (BOOCH, RUMBAUGH e JACOBSON, 1999). Eles descrevem os passos de interação que ocorrem entre o sistema e o ator. Cada um destes passos corresponde a uma ação do ator sobre o sistema e a reação do sistema a esta ação. O conjunto destes eventos pode ser representado por um fluxo, ou um diagrama de atividades.

Neste nível de decomposição cada requisito funcional será representado por uma atividade. As decomposições serão feitas da seguinte forma. Para cada subcaso de uso encontrado no nível anterior serão identificadas as atividades correspondentes. Estas atividades são identificadas através da descrição da interação do ator com o sistema. Estas atividades poderão ser decompostas enquanto for necessário para este nível de abstração.

No exemplo apresentado na Tabela 2, o subcaso de uso “Entra dados de Empregado” (1.1.1) pode ser decomposto no seguinte fluxo de atividades: 1.1.1.1

“selecionar a opção inserir”; 1.1.1.2 “receber dados do empregado”; 1.1.1.3 confirmar dados digitados. Esta decomposição pode ser representada pelo diagrama de atividades da Figura 10.

FIGURA 10 DIAGRAMA DE ATIVIDADES PARA O SUBCASO DE USO “ENTRAR DADOS DE EMPREGADO”



Neste nível de decomposição, cada atividade será mapeada em uma sub-colaboração da colaboração anterior. Formando uma matriz de projeto que representará nas linhas as atividades e nas colunas as colaborações correspondentes. Novamente, as dependências funcionais têm que estar de acordo com as dependências nos níveis anteriores para manter a consistência no processo de decomposição (TATE, 1999).

3.3.2.4 Quarto nível de decomposição

Logo após a identificação de todas as atividades (FR) e respectivas colaborações (DP) para cada atividade, são identificados os serviços técnicos (FR) resultados da decomposição da atividade em sub-requisitos. Para cada serviço técnico é encontrado um parâmetro de projeto (DP) que é um objeto usado para satisfazer o requisito funcional (FR). A célula FR x DP corresponde a uma tripla que é composta de um elemento do domínio de interação, um do domínio de classes e um do domínio de estados.

Para definir um serviço técnico será usada a definição de função de serviço (STADZISZ, 1997). Segundo STADZISZ (1997), “Uma função de serviço exprime uma ação esperada do produto sobre um elemento do meio exterior, em benefício de outro elemento deste meio, dentro de uma fase de utilização”. Se um objeto do sistema for considerado

como um produto e os outros objetos, com os quais interage, seu meio externo, então, um serviço técnico é um serviço esperado de um objeto por outro, para realizar a colaboração.

Um requisito funcional pode ser decomposto em sub-requisitos de vários tipos. Segundo HINTERSTEINER (1999), eles podem ser classificados em funções de processo, de comando e controle, e de suporte e integração. Esta classificação é similar aos conceitos da arquitetura Modelo-Visão-Controlador (MVC) (KRASNER e POPE, 1988) (PRESSMAN, 2004) que pode ser usado como base para a decomposição neste nível.

No caso de sistemas de *software*, são encontrados três tipos de funcionalidades. Em uma atividade são comumente encontradas funcionalidades para trocar informações entre o sistema e atores (humanos ou não), funcionalidades para manipular os dados do sistema e funcionalidades para controlar o processo realizado na atividade. Devido a este fato, pode-se representar cada uma destas funcionalidades como sendo um serviço técnico que, a rigor, é um requisito funcional de um nível hierárquico mais baixo. Neste trabalho, os serviços técnicos serão classificados nas seguintes categorias:

- serviços técnicos de fronteira;
- serviços técnicos de entidade;
- serviços técnicos de controle;

Serviços técnicos de fronteira são serviços para fazer a interface do sistema com os atores. Serviços técnicos de entidade são serviços para manipular as informações do sistema. Serviços técnicos de controle são serviços para controlar o processo realizado na atividade.

Na metodologia apresentada neste trabalho, a decomposição funcional de uma atividade será feita identificando-se os serviços técnicos de fronteira, controle e entidade que compõem a atividade. Por exemplo, para a atividade “receber dados de empregado”, serão identificados os serviços técnicos “mostrar formulário” e “receber dados usuário”.

Para cada serviço técnico identificado serão escolhidos os parâmetros de projeto que serão variáveis (objetos) que participam da colaboração identificada no nível anterior.

Com o mapeamento FR x DP, cada serviço técnico será mapeado em um objeto. De acordo com PRESSMAN (2004) um objeto que executa serviço de fronteira recebe o estereótipo de <<fronteira>>, o que executa serviços de entidade recebe o estereótipo <<entidade>> e o que executa serviços de controle recebe o estereótipo <<controle>>.

TABELA 3 MATRIZ DE PROJETO DE 4^{O}. NÍVEL

	DP 1																							
	?	2																						
	?	1																						
	?	1		2		3			4		5											6		
	Interface banco de dados	tela de escolha de opção	variável opção	tela receber id exemplar	exemplar	colaboração Iniciar leitor barras	colaboração ler código	colaboração mostrar código lido	tela receber id usuário	variável usuário	consulta à tabela usuário	variável usuário	consulta à tabela empréstimos	variável número de atrasos	consulta à tabela exemplar	variável exemplar	consulta à tabela obra	variável obra	tela confirmação empréstimo	variável confirmação	consulta à tabela empréstimos	tela sucesso gravação
$FR_{?.?.?}$ - conectar com a base de	X																							
$FR_{1.2.1.1.1}$ - mostrar tela escolha		X																						
$FR_{1.2.1.1.2}$ - receber opção			X																					
$FR_{1.2.1.2.1}$ - mostrar tela receber id exemplar				X																				
$FR_{1.2.1.2.2}$ - receber id exemplar					X																			
$FR_{1.2.1.3.1}$ - Iniciar leitor de cod barras						X																		
$FR_{1.2.1.3.2}$ – ler código							X																	
$FR_{1.2.1.3.3}$ - mostrar código lido								X																
$FR_{1.2.1.4.1}$ - mostrar tela receber id usuário									X															
$FR_{1.2.1.4.2}$ - receber id usuário										X														
$FR_{1.2.1.5.1}$ - buscar dados Usuário	X										X													
$FR_{1.2.1.5.2}$ - receber busca dados Usuário	X									R		X												
$FR_{1.2.1.5.3}$ - verificar na base empréstimos atrasados	X												X											
$FR_{1.2.1.5.4}$ - receber busca empréstimos atrasados	X													X										
$FR_{1.2.1.5.5}$ - buscar dados exemplar	X														X									
$FR_{1.2.1.5.6}$ - receber busca dados exemplar	X				R											X								
$FR_{1.2.1.5.7}$ - buscar dados obra	X																X							
$FR_{1.2.1.5.8}$ - receber busca dados obra	X																	X						
$FR_{1.2.1.5.9}$ - mostrar tela confirmação					R					R		X		X		X		X	X					
$FR_{1.2.1.5.10}$ - receber confirmação																				X				
$FR_{1.2.1.6.1}$ - gravar dados de empréstimo na base	X																				X			
$FR_{1.2.1.6.2}$ - mostrar mensagem de sucesso																						X		

A matriz de projeto resultante terá representado nas linhas os serviços técnicos e nas colunas os objetos (variáveis) que irão executá-los. Como exemplo, na Tabela 3 o

serviço técnico “receber id usuário” (FR 1.2.1.4.2) é mapeado no parâmetro de projeto “variável usuário” (DP 1.2.1.4.2). Nesta forma de mapeamento, quando uma variável (objeto) é usada em mais de um requisito funcional (FR), esta relação é marcada com um “X”. Quando não é o mesmo objeto, mas uma outra instância da mesma classe aparece em outro requisito funcional (FR), esta relação é marcada com um “R”. Então, das relações “X” e “R” pode-se concluir em quais requisitos funcionais (FRs) aparecem objetos de uma classe. Com isso, pode-se identificar as principais utilizações de cada objeto e seus métodos principais.

Na célula da matriz de projeto que representa o mapeamento FR x DP é representado não apenas o método chamado, mas uma tripla. Esta tripla (interação, estrutura, mudança de estado) representa uma composição das visões interativa, estrutural e dinâmica do projeto. Estas visões estão de acordo com a divisão da modelagem em estrutural e comportamental, proposta em (BOOCH, RUMBAUGH, e JACOBSON, 1999). Cada tripla representa a interação onde o envio da mensagem é realizado, para qual objeto é feito, a mudança na estrutura da classe que o envio da mensagem acarreta e a mudança de estados do objeto que este envio ocasiona.

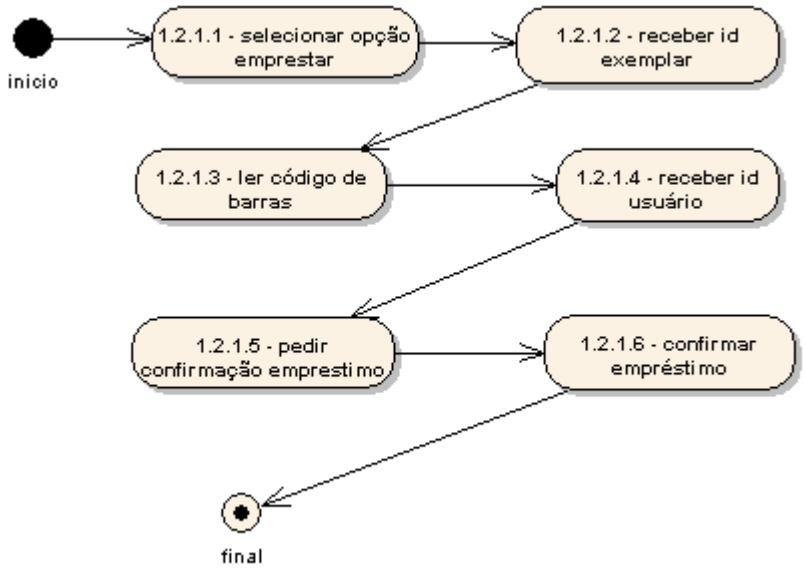
A representação destas visões na forma de uma célula resultante do mapeamento FR x DP permite que seja feito o rastreamento do local de utilização de cada método e, seja visualizado qual o efeito de cada chamada deste método. Além disso, permite que todos os métodos sejam rastreados segundo os requisitos funcionais que eles satisfazem. Este tipo de rastreamento se torna importante em projetos grandes e de risco.

Após o mapeamento do serviço técnico e da variável (objeto) correspondente na matriz de projeto, os elementos da tripla serão representados nos respectivos diagramas. O envio da mensagem para o objeto correspondente será representado em um diagrama de seqüência. A definição do método, correspondente ao envio dessa mensagem, será representada no diagrama de classes. O envio da mensagem para o objeto irá ocasionar a mudança de estado do objeto. Esta mudança de estado será representada no diagrama de

estados para o objeto.

Para ilustrar esta representação será apresentada como exemplo a decomposição da atividade “confirmar empréstimo” referente a um sistema de gerenciamento de bibliotecas. Esta atividade é resultante da decomposição de terceiro nível de um subcaso de uso, representada na Figura 11.

FIGURA 11 DIAGRAMA DE ATIVIDADES DA DECOMPOSIÇÃO DO SUBCASO EMPRESTAR EXEMPLAR



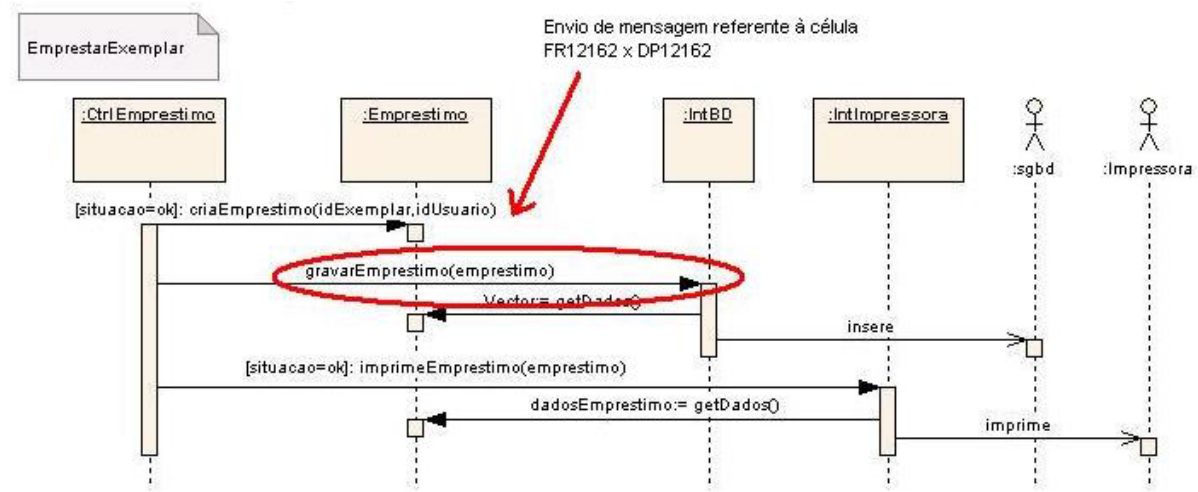
A decomposição da atividade “confirmar empréstimo” (1.2.1.6) resulta nos serviços técnicos “criar novo empréstimo” (FR 1.2.1.6.1), “gravar dados do empréstimo na base” (FR 1.2.1.6.2) e “imprimir comprovante empréstimo” (FR 1.2.1.6.3). Estes serviços técnicos serão mapeados nos seus respectivos parâmetros de projeto que são: variável “empréstimo” (DP 1.2.1.6.1), variável “consulta à tabela empréstimo” (DP 1.2.1.6.2) e variável “interface com a impressora” (DP 1.2.1.6.3). Este mapeamento está representado em uma matriz de projeto parcial, ilustrada na Tabela 4.

TABELA 4 MATRIZ DE PROJETO REFERENTE À DECOMPOSIÇÃO DE CONFIRMAR EMPRÉSTIMO

	DP 1.2.1.6.1 - variável empréstimo	DP 1.2.1.6.2 - variável de consulta à tabela empréstimo	DP 1.2.1.6.3 – variável interface com a impressora
FR 1.2.1.6.1 – criar novo empréstimo	X		
FR 1.2.1.6.2 – gravar dados do empréstimo na base	X	X	
FR 1.2.1.6.3 – imprimir comprovante empréstimo			X

O serviço técnico “gravar dados do empréstimo na base” (FR 1.2.1.6.2) utiliza o parâmetro de projeto variável “consulta à tabela empréstimo” (DP 1.2.1.6.2). Para representar os envios de mensagens referentes à colaboração confirmar empréstimo (DP 1216) é feito um diagrama de seqüência para esta colaboração. Neste diagrama é representado o envio da mensagem “gravarEmprestimo(emprestimo)” referente ao mapeamento do serviço técnico “gravar dados do empréstimo na base”, ilustrado no diagrama de seqüência da Figura 12.

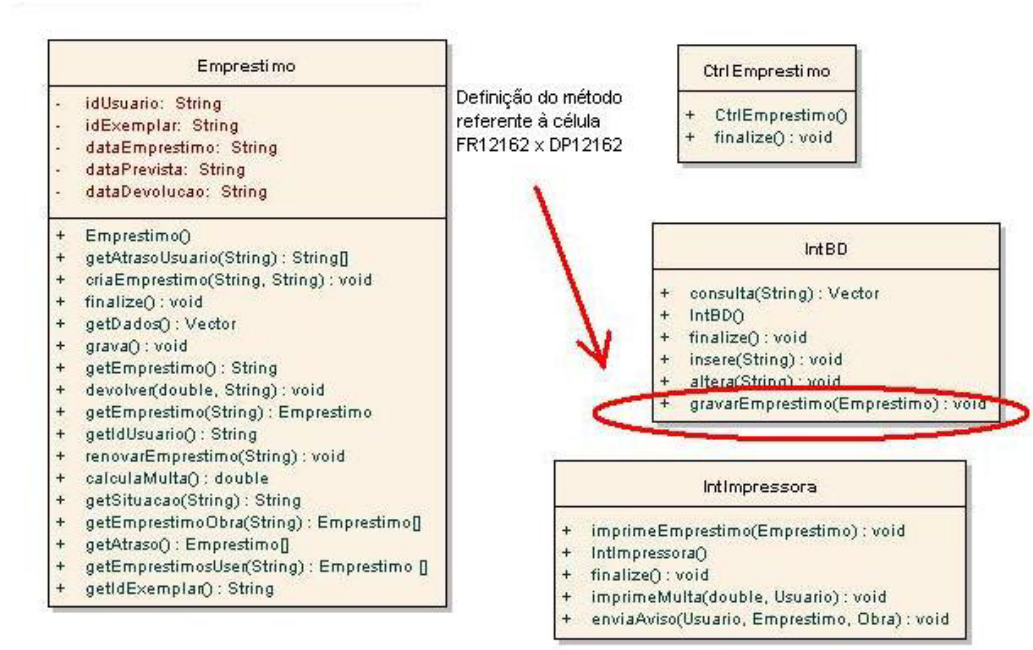
FIGURA 12 ENVIO DE MENSAGEM REFERENTE A FR12162 X DP12162



O mapeamento do serviço técnico “gravar dados do empréstimo na base”, além do envio da mensagem “gravarEmprestimo(empréstimo)” deve produzir a descrição de um

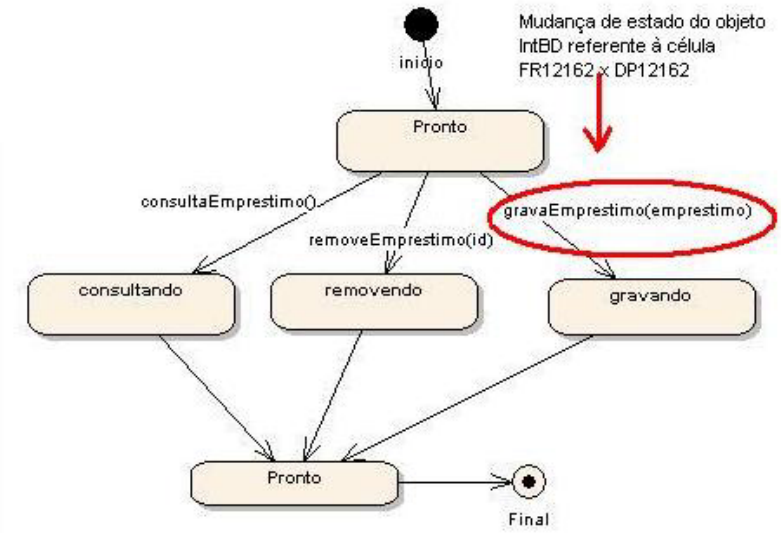
método novo no diagrama de classes. Esta descrição está representada no diagrama da Figura 13. Mesmo no caso da utilização de um método que já existe, a tripla representa que este método está sendo usado para satisfazer o requisito funcional “gravar dados do empréstimo na base” (FR 1.2.1.6.2).

FIGURA 13 DEFINIÇÃO DE MÉTODO REFERENTE A FR12162 X DP12162



Na tripla também é representada a mudança de estado que o envio da mensagem “gravarEmprestimo(empréstimo)” causa no objeto “IntBD”. Esta mudança de estados é representada no diagrama de estados da classe “IntBD” apresentado na Figura 14.

FIGURA 14 MUDANÇA DE ESTADO DO OBJETO INTBD REFERENTE À CÉLULA FR12162 X DP12162



3.3.3 Mapeamento entre domínios físico e de processo.

A diferença básica entre as variáveis do domínio físico e as do domínio de processo é que as primeiras têm um nível de abstração e uma importância maior para o projeto, são independentes de plataforma de implementação e normalmente representam objetos globais. As variáveis do domínio de processo são normalmente variáveis que existem em um nível de abstração menor, são usadas para realizar os parâmetros de projeto e normalmente representam variáveis locais.

A decomposição dos requisitos funcionais pode ser feita até o nível de abstração desejado no projeto. Este nível é atingido, nesta metodologia, quando são obtidos os serviços técnicos. Apesar disso, esses serviços técnicos podem ser decompostos em sub-serviços e assim por diante até que seja atingido o nível equivalente a uma linha de código do sistema. Esta decomposição pode parecer excessiva a princípio, mas quando é exigido que o projeto tenha a capacidade de rastrear cada item da implementação com relação aos requisitos funcionais, ela se torna muito útil. Esta capacidade de rastreamento é particularmente útil quando são feitos testes no sistema.

Segundo LEFFINGWELL e WIDRIG (2003) “a experiência tem mostrado que a habilidade de rastrear os artefatos de requisitos através dos estágios de especificação, arquitetura, projeto, implementação e testes é um fator significativo em assegurar a qualidade da implementação do *software*“. Testes em sistemas de *software* podem ser feitos de varias formas. A forma mais comum é a feita com base nos casos de uso (*test cases*) (BITTNER e SPENCE, 2003). Este tipo de teste requer que o trecho de código se deseja corrigir seja rastreado a partir do erro encontrado na utilização do sistema (requisito funcional).

3.4 -CÁLCULO DO CONTEÚDO DE INFORMAÇÃO

Durante o processo de desenvolvimento o conteúdo de informação é calculado. Este conteúdo é definido com sendo a probabilidade do projeto não satisfazer os requisitos funcionais. Segundo o axioma dois (SUH, 1990) e alguns teoremas e corolários relacionados, quanto menor o conteúdo de informação melhor será o projeto (SUH, 1990). O conteúdo de informação normalmente é calculado quando se está fazendo o mapeamento entre o domínio físico e o de processo, mas ele pode ser calculado a qualquer etapa do desenvolvimento.

A utilidade do cálculo do conteúdo é permitir que seja possível identificar, entre soluções possíveis para o problema proposto, que tenham independência funcional, qual delas é a melhor solução. Por exemplo, quando se está fazendo uma decomposição em um determinado nível e aparecem duas decomposições possíveis, pode-se decidir qual delas adotar. Apesar da fórmula para o cálculo do conteúdo de informação ser independente do domínio do problema (SUH, 1990), seus parâmetros devem ser adaptados para o projeto de *software*. Esta adaptação será estudada no próximo capítulo.

4 MEDIDA DE CONTEÚDO DE INFORMAÇÃO PARA SISTEMAS DE *SOFTWARE*

Neste capítulo são definidos os parâmetros para o cálculo do conteúdo de informação para projetos de *software*. Para esta definição serão usadas métricas de projeto de *software*. Estas métricas refletem a complexidade referente ao projeto e a complexidade referente à implementação do sistema computacional. As métricas adotadas consideram as seguintes características: número e complexidade dos diagramas de projeto, tamanho do código (Linhas de código), complexidade do algoritmo, número de chamadas a outros módulos (métodos) dentro de um método, número de parâmetros recebidos por um método, número e complexidade de tabelas, número e complexidade das telas e outros aspectos que influenciam no projeto e construção do *software*.

4.1 CONTEÚDO DE INFORMAÇÃO DE SISTEMAS COMPUTACIONAIS

As medidas de conteúdo de informação costumam levar em consideração faixas de valores que as variáveis de projeto podem atingir ou a probabilidade de que estas faixas de valores estejam dentro de uma tolerância. Em termos de projeto de *software*, as necessidades são diferentes, a variabilidade das variáveis de projeto em termos de valores não tem uma importância tão grande como a complexidade destas variáveis. Devido a essa diferença, serão estabelecidas medidas de conteúdo de informação baseada em medidas de complexidade de componentes de sistemas de *software* que possam ser aplicadas nos diversos níveis de decomposição de um projeto.

O conteúdo de informação é calculado a cada decomposição dos requisitos funcionais (FRs). Isto se deve ao fato de que é necessário verificar se os sub-requisitos funcionais (sub-FRs) encontrados e os parâmetros de projeto (DPs) correspondentes

satisfazem os axiomas 1 e 2. Além disso, o cálculo do conteúdo de informação mostra a vantagem da escolha de um conjunto de sub-requisitos funcionais (sub-FRs) ao invés de um outro conjunto através do axioma 2 (SUH, 1990). No caso, se os dois conjuntos de sub-requisitos funcionais (sub-FR) tiverem independência funcional, o melhor conjunto é o que possuir o menor conteúdo de informação. Como visto anteriormente o conteúdo de informação pode ser calculado pela fórmula em (10).

No caso de sistemas computacionais, será feita uma adaptação destes conceitos. O cálculo do conteúdo de informação será feito com base no tamanho e na complexidade do sistema. Os sistemas computacionais, em sua grande maioria, ainda são projetados e construídos manualmente. Não é uma máquina ou sistema computacional que irá construir o *software*. Portanto, o fator humano deve ser levado em consideração quando o inverso da probabilidade de sucesso é calculado. Por esta razão, neste trabalho, o valor da tolerância será calculado através de uma estimativa de complexidade baseada em registros da complexidade de sistemas construídos anteriormente pela organização. Isto será feito pelo fato de que se o valor de complexidade obtido for maior que a estimativa feita com base no registro histórico da organização, a probabilidade da construção do *software* ser mais difícil para esta organização é maior.

4.2 CONTEÚDO DE INFORMAÇÃO NOS NÍVEIS DE DECOMPOSIÇÃO

O conteúdo de informação será medido a cada decomposição dos requisitos funcionais (FRs). Segundo a metodologia de desenvolvimento adotada, as decomposições são:

- 1º nível de decomposição - identificação dos casos de uso;
- 2º nível de decomposição - identificação dos subcasos de uso;
- 3º nível de decomposição - identificação dos “eventos” ou atividades;
- 4º nível de decomposição - identificação dos serviços técnicos.

Como visto no capítulo anterior, a cada nível de composição podem ser realizadas varias decomposições, enquanto houver necessidade para este nível de abstração do projeto. A Tabela 5 apresenta as relações entre os níveis de decomposição, os tipos de requisitos funcionais e as métricas de complexidade usadas em cada um dos níveis.

TABELA 5 MÉTRICAS DE COMPLEXIDADE E TIPOS DE REQUISITOS FUNCIONAIS

Nível de decomposição	Requisitos funcionais	Métrica de complexidade
1º nível de decomposição	Casos de uso	*Use case points*
2º nível de decomposição	Subcasos	*Use case points*
3º nível de decomposição	Atividades	Pontos por função
4º nível de decomposição	Serviços técnicos	Conjunto de Métricas C K (CHIDAMBER e KEMERER, 1994)

4.2.1 Conteúdo de informação nos níveis 1 e 2

No primeiro e no segundo nível de decomposição (identificação dos casos de uso e subcasos de uso) o conteúdo de informação é calculado com base nos *use case points* (KARNER, 1993) (ANDA et al., 2001). Os *use case points* são originados a partir de uma métrica de sistemas computacionais amplamente utilizada que é a de pontos por função (*Function Points*) (VAZQUEZ; SIMÕES; ALBERT, 2003). Tal como os pontos por função, os *use case points* permitem que a complexidade do sistema seja mensurada a partir dos seus requisitos funcionais. Além disso, podem ser medidas as complexidades de partes do sistema, e em diferentes níveis de decomposição funcional e abstração.

Os *use case points* são calculados da seguinte maneira. A complexidade de cada ator é avaliada e é atribuído um peso. Estes pesos são somados e é obtido o peso não ajustado dos atores. A seguir, a complexidade de cada caso de uso é avaliada e é atribuído um peso. Estes pesos também são somados e é obtido o peso não ajustado dos casos de uso. Estes valores são somados e é obtido o valor para os *use case points* não ajustados. A este valor são aplicados os fatores de ajuste técnico e ambiental, obtendo-se assim o valor para os *use case points*. Os fatores de ajuste técnico e de ambiente levam em consideração

fatores como usabilidade, instalabilidade, reusabilidade, complexidade de processamento, concorrência de processos, experiência da equipe, motivação, entre outros.

Adaptando a fórmula em (10), o *system range* é dado pela contagem dos *use case points* de cada caso de uso e a tolerância é dada pela estimativa baseada na média histórica das contagens de *use case points* da organização.

$$I = \log\left(\frac{ucpc}{eucp}\right) \tag{12}$$

A fórmula em (12) representa o cálculo do conteúdo de informação de um requisito funcional (FR), onde *ucpc* representa os *use case points* contados para o requisito funcional (FR) específico e *eucp* a estimativa baseada nos *use case points* médios da organização para requisitos funcionais (FR) da mesma complexidade.

4.2.2 Conteúdo de informação do nível 3

No terceiro nível de decomposição são definidas as atividades (ver seção 3.3.2.3). A complexidade relativa às atividades não pode ser calculada através dos *use case points*. Neste caso, a métrica que se mostra mais adequada é a de pontos por função.

A métrica de pontos por função (*Function Points*) (VAZQUEZ; SIMÕES; ALBERT, 2003) é amplamente utilizada para a estimativa e medida de tamanho de sistemas computacionais. A contagem de pontos por função permite que o sistema seja mensurado a partir dos seus requisitos funcionais. Além disso, podem ser medidas partes do sistema, e em diferentes níveis de decomposição funcional e abstração. Então, se para *system range* for adotado a contagem de pontos por função do requisito funcional (FR) e para tolerância for adotada a estimativa feita com base na média histórica para requisitos funcionais (FRs) de mesma complexidade, obtém-se a seguinte fórmula.

$$I = \log\left(\frac{pfc}{epf}\right) \tag{13}$$

Em (13) é representado o cálculo do conteúdo de informação de um requisito

funcional (FR), onde *pfc* representa os pontos por função contados para o requisito funcional (FR) específico e e*pf*, a estimativa dos pontos por função com base na média histórica da organização para requisitos funcionais (FRs) da mesma complexidade.

Com essa fórmula consegue-se calcular a razão fundamental do conteúdo de informação que é calcular a probabilidade de que o parâmetro de projeto (DP) identificado satisfaça o respectivo requisito funcional (FR). No caso de projetos de *software*, a organização que faz os projetos (Por ex: *software houses*) possui um registro histórico das contagens de pontos por função. Através desse registro historico é possível se fazer uma estimativa para o valor da contagem de pontos por função. Se a contagem de pontos por função da FR for maior que a estimativa, o conteúdo de informação será positivo e a probabilidade total de insucesso será menor. Se for menor, *I* será negativo e a probabilidade total de insucesso diminuirá. O fato de a contagem de pontos por função do requisito funcional (FR) ser menor que a estimativa feita com base na média histórica de contagens também significa que este requisito funcional (FR) tem uma complexidade menor que a complexidade média dos projetos produzidos na empresa, ou seja, teoricamente é mais fácil se ser realizado. No caso contrário, ele seria mais difícil de ser construído pela empresa.

4.2.3 Conteúdo de informação para o nível 4

Para o quarto nível de decomposição, o cálculo de pontos por função não é mais satisfatório. Neste nível precisam ser medidas as complexidades de classes, métodos, interações entre objetos, mudanças de estados, hierarquia de herança, entre outros. Para satisfazer essas necessidades será adotada uma medida de complexidade mais voltada para esses aspectos, como a definida em (CHIDAMBER e KEMERER, 1994).

Como os requisitos funcionais neste nível são serviços técnicos e os parâmetros de projeto são objetos ou variáveis é necessário usar uma métrica que contemple características de classes, atributos e métodos. Nesta métrica são definidas seis grandezas que serão medidas: Métodos ponderados por classe, profundidade da árvore de herança, número de subclasses, acoplamento entre objetos, respostas para a classe e falta de

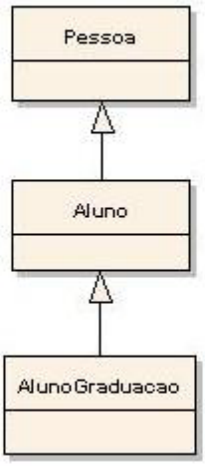
coesão nos métodos (CHIDAMBER e KEMERER, 1994).

A grandeza “métodos ponderados por classe” (WMC, do inglês *weighted methods per class*) representa o total das complexidades dos métodos de uma classe. O valor é dado pela soma das complexidades de cada método. Uma forma simplificada de calcular o WMC é considerar os métodos com o mesmo valor para complexidade, sendo este valor igual a 1. Neste caso, o valor do WMC é igual ao número de métodos da classe. Segundo CHIDAMBER e KEMERER (1994), o número de métodos de uma classe e a complexidade destes ajuda a estimar o tempo e o esforço para desenvolver e manter a classe. Classes com um número grande de métodos, são muito específicas e limitam a possibilidade de reuso. Sendo c_i a complexidade do método i de uma classe, WMC pode ser calculado pela relação em (14).

$$WMC = \sum_{i=1}^{n} c_i \qquad (14)$$

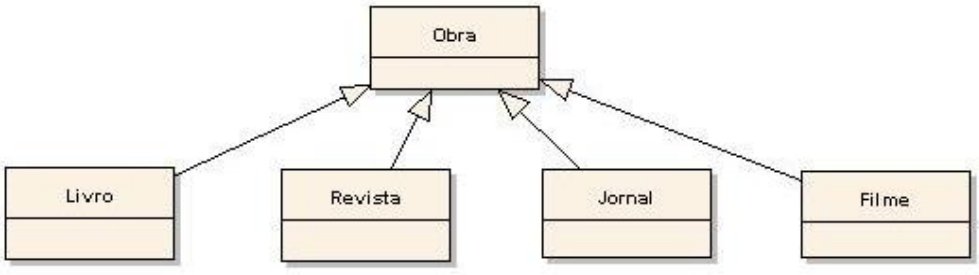
A profundidade da árvore de herança (DIT, do inglês *depth of the inheritance tree*), para uma classe é definida como sendo o comprimento máximo do nó que representa a classe até a raiz da árvore. Árvores de herança muito profundas geram muita complexidade no projeto pelo fato de mais métodos e classes estarem envolvidos (CHIDAMBER e KEMERER, 1994). A Figura 15 ilustra uma árvore de herança. Neste exemplo, o valor de DIT para a classe “AlunoGraduacao” é igual a 2.

FIGURA 15 PROFUNDIDADE DA ÁRVORE DE HERANÇA (DIT)



O número de subclasses (NOC, do inglês *number of children*) é o número de subclasses diretas de uma classe. Segundo CHIDAMBER e KEMERER (1994), quanto maior o número de subclasses diretas de uma classe, maior a possibilidade de que a herança tenha sido usada incorretamente para esta classe. A Figura 16 ilustra uma árvore de hierarquia. O valor para NOC para a classe “Obra” é igual a 4 pois ela tem 4 subclasses diretas.

FIGURA 16 NÚMERO DE SUBCLASSES (NOC)



A grandeza “acoplamento entre objetos” (CBO, do inglês *coupling between*

objects) de uma classe é definida como sendo o número de outras classes com as quais esta classe se relaciona (CHIDAMBER e KEMERER, 1994). Em outras palavras, o número de classes das quais esta classe usa métodos ou variáveis de instância. Uma maneira prática para determinar o CBO é usando os cartões classe-responsabilidade-colaborador (CRC) definidos por BECK e CUNNINGHAM (1989). Um colaborador é uma classe que presta serviço para que uma outra classe consiga realizar suas responsabilidades.

A grandeza "resposta para a classe" (RFC, do inglês *response for a class*) é definida como a cardinalidade do conjunto de respostas de uma classe, como na relação em (15).

$$RFC = |RS| \tag{15}$$

Segundo CHIDAMBER e KEMERER (1994), o conjunto de respostas de uma classe é o conjunto de métodos de uma classe que podem potencialmente ser executados em resposta a uma mensagem recebida. Segundo CHIDAMBER e KEMERER (1994), quanto maior o conjunto de métodos que podem ser chamados de uma classe, maior a complexidade da classe. Se um grande número de métodos pode ser chamado de uma classe, as operações de teste e correção ficam muito mais complexas, exigindo maior experiência por parte do desenvolvedor. (CHIDAMBER e KEMERER, 1994) Sendo *RS*, o conjunto de resposta da classe, { R_i }, o conjunto dos métodos da classe chamados pelo método *i* e { *M* }, o conjunto de todos os métodos da classe. O conjunto de resposta de uma classe, *RS,* pode ser definido pela relação em (16).

$$RS = \{M\} \cup_{all\ i} \{R_i\} \tag{16}$$

A grandeza "falta de coesão nos métodos" (LCOM, do inglês *lack of cohesion of methods*) é o número de métodos da classe que acessam um ou mais dos mesmos atributos. Se dois métodos de uma classe acessam um mesmo atributo da classe, é dito que eles compartilham o atributo. LCOM de uma classe é o número de métodos que compartilham atributos. Se LCOM é alto, métodos podem estar acoplados através dos

atributos. Isto aumenta a complexidade do projeto da classe (PRESSMAN, 2004). A grandeza “falta de coesão nos métodos” provê uma medida de desigualdade dos métodos de uma classe. Segundo CHIDAMBER e KEMERER (1994), qualquer medida de desigualdade dos métodos ajuda a identificar falhas no projeto de classes.

O valor do conteúdo de informação do 4º nível de decomposição é obtido pela soma dos valores do conteúdo de informação para cada classe. O valor do conteúdo de informação de cada classe é calculado pela soma dos conteúdos de informação relativos a cada umas das grandezas definidas por CHIDAMBER e KEMERER (1994), como em (17).

$$I_{classe} = I_{WMC} + I_{DIT} + I_{NOC} + I_{CBO} + I_{RFC} + I_{LCOM} \quad (17)$$

Este cálculo pode ser feito, também, usando-se fatores de ponderação para cada grandeza, como na relação (18). Nesta relação $\alpha, \beta, \delta, \gamma, \theta$ e ω são os fatores de ponderação. Estes fatores são definidos, pelo projetista, com base na importância do conteúdo de informação de cada grandeza.

$$I_{classe} = \alpha I_{WMC} + \beta I_{DIT} + \delta I_{NOC} + \gamma I_{CBO} + \theta I_{RFC} + \omega I_{LCOM} \quad (18)$$

Os valores dos conteúdos de informação são calculados pelas relações entre os valores encontrados para cada classe, para cada grandeza, como em (19), (20), (21), (22), (23) e (24).

$$I_{WMC} = \log\left(\frac{WMC}{EWMC}\right) \quad (19)$$

Onde I_{WMC} é o conteúdo de informação relativo a “métodos ponderados por classe” (WMC) e *EWMC* é o valor estimado de *WMC* para a classe com base na média histórica.

$$I_{DIT} = \log\left(\frac{DIT}{EDIT}\right) \quad (20)$$

Onde I_{DIT} é o conteúdo de informação relativo a “profundidade da árvore de herança” (DIT) e *EDIT* é o valor estimado de *DIT* para a classe com base na média histórica.

$$I_{NOC} = \log\left(\frac{NOC}{ENOC}\right) \quad (21)$$

Onde I_{NOC} é o conteúdo de informação relativo a “número de subclasses” (NOC) e *ENOC* é o valor estimado de *NOC* para a classe com base na média histórica.

$$I_{CBO} = \log\left(\frac{CBO}{ECBO}\right) \quad (22)$$

Onde I_{CBO} é o conteúdo de informação relativo a “acoplamento entre objetos” (CBO) e *ECBO* é o valor estimado de *CBO* para a classe com base na média histórica.

$$I_{RFC} = \log\left(\frac{RFC}{ERFC}\right) \quad (23)$$

Onde I_{RFC} é o conteúdo de informação relativo a “resposta para a classe” (RFC) e *ERFC* é o valor estimado de *RFC* para a classe com base na média histórica.

$$I_{LCOM} = \log\left(\frac{LCOM}{ELCOM}\right) \quad (24)$$

Onde I_{LCOM} é o conteúdo de informação relativo a “falta de coesão nos métodos” (LCOM) e *ELCOM* é o valor estimado de *LCOM* para a classe com base na média histórica.

4.3 APLICAÇÃO DO CONTEÚDO DE INFORMAÇÃO NA METODOLOGIA

O conteúdo de informação é um parâmetro importante para se escolher entre as diferentes soluções possíveis para um projeto. A metodologia apresentada neste trabalho propõe o cálculo do conteúdo de informação a cada decomposição feita para, no caso de surgir mais de uma possibilidade para a decomposição, facilitar a decisão de qual adotar.

Os *use case points*, usados para calcular o conteúdo de informação para o primeiro e o segundo nível de decomposição, podem ser calculados facilmente com a ajuda

de ferramentas CASE (*Computer Aided Software Engineering*) como a *Enterprise Architect*[5]. No terceiro nível de decomposição, o conteúdo de informação é calculado através dos pontos por função. Para isto também existem várias opções de ferramentas de *software*. Mas para se calcular o conteúdo de informação é necessário que seja possível estimar os valores com base em registros históricos da empresa que vai realizar o projeto. Então, é necessário que a empresa mantenha controle e registros dos projetos realizados, o que é uma característica de maturidade de processo de *software*. (PAULK et al., 1993)

Para o quarto nível de decomposição será usado o conjunto de métricas definido por CHIDAMBER e KEMERER (1994). A grande vantagem deste conjunto de métricas é a facilidade em se calcular seus valores. Estes valores podem ser calculados durante o projeto, usando-se os diagramas de classe e seqüência da UML 2.0 (OBJECT MANAGEMENT GROUP, 2003), ou diretamente do código através de uma análise simples, que pode ser automatizada. Isto permite que projetos anteriores da organização sejam analisados e seus valores obtidos, auxiliando a obtenção da estimativa.

TABELA 6 CONTEÚDO DE INFORMAÇÃO DA CLASSE “CINTBD”

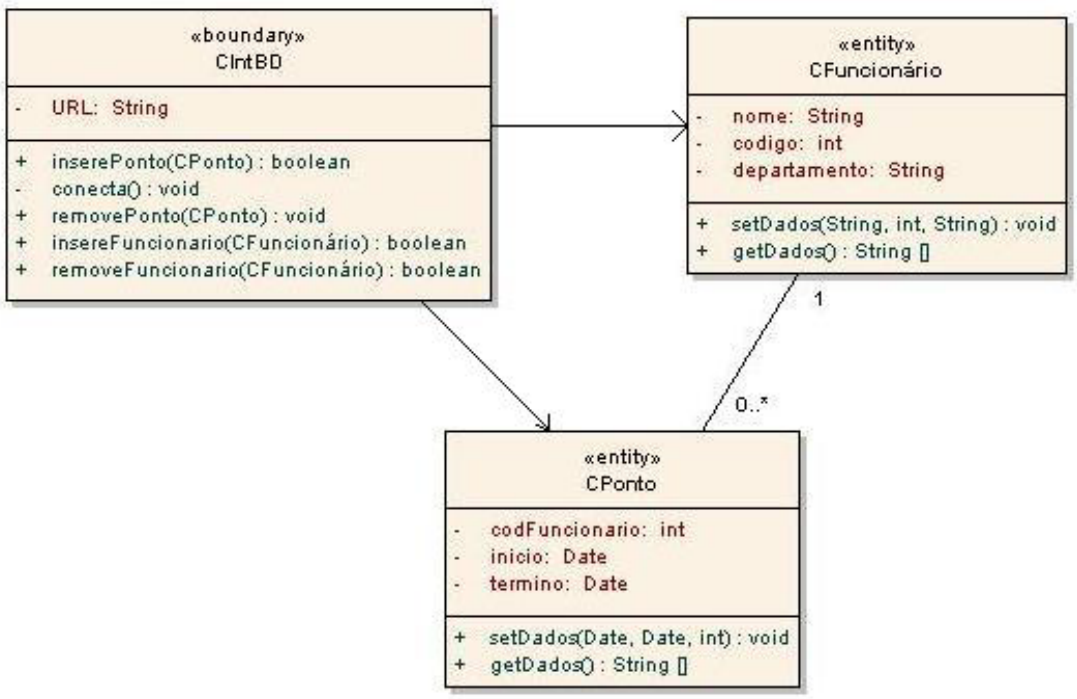
Grandeza	Valor medido	Estimativas	Conteúdo de Informação
WMC	5	8	-0,67807
DIT	0	0	0
NOC	0	0	0
CBO	2	2	0
RFC	4	6	-0,58496
LCOM	5	3	0,73696

Para ilustrar como este cálculo do conteúdo de informação é realizado, será apresentado um exemplo. No diagrama de classes da Figura 17, são apresentadas as classes “CIntBD”, “CPonto” e CFuncionário”. Será apresentado o cálculo do conteúdo de informação para a classe “CIntBD”. Na Tabela 6 é apresentado, na primeira coluna, o nome da grandeza e, na segunda coluna, é exibido o valor da grandeza calculado para a classe

[5] O *software Enterprise Arquitect* é marca registrada da *Sparx Systems*

"CIntBD". Na terceira coluna é apresentada a estimativa do valor para cada grandeza. Na quarta coluna é apresentado o valor do conteúdo de informação para cada grandeza.

FIGURA 17 CLASSES PARA O CÁLCULO DO CONTEÚDO DE INFORMAÇÃO



O conteúdo de informação da classe "CIntBD" é calculado pela soma dos conteúdos de informação de cada grandeza, apresentados na Tabela 6. O valor obtido é $I_{CIntBD} = -0,53056$. Um valor negativo para o conteúdo de informação representa um decréscimo no conteúdo de informação total. Isto pode indicar que esta classe pode ser uma boa solução. Mas, só é possível ter certeza de que a solução é melhor que outra com base no conteúdo de informação total, isto é, quando todas as classes já tiverem sido projetadas e avaliadas.

Supondo os mesmos requisitos funcionais (FRs) do exemplo anterior, será considerado que o projetista criou apenas uma classe, chamada "CIntBDNova", para receber dados de ponto e empregado, retornar dados de ponto e empregado e gravar e

remover esses dados. A classe “CIntBDNova” é apresentada na Figura 18. O conteúdo de informação da classe “CIntBDNova” é calculado pela soma dos conteúdos de informação de cada grandeza, apresentados na Tabela 7. O valor obtido é $I_{CIntBDNova}$ = 1,5539. Um valor positivo para o conteúdo de informação representa um acréscimo no conteúdo de informação total. A classe “CIntBDNova” possui um conteúdo de informação maior que a classe “CIntBD”. Neste caso a classe “CIntBD” pode ser considerada uma solução melhor que a classe “CIntBDNova”, pelo axioma 2.

FIGURA 18 CLASSE CINTBDNOVA

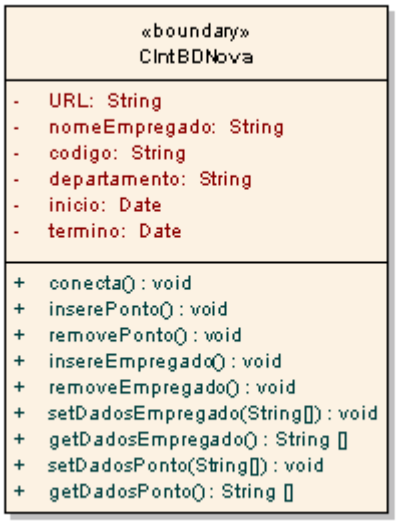


TABELA 7 CONTEÚDO DE INFORMAÇÃO DA CLASSE “CINTBDNOVA”

Grandeza	Valor medido	Estimativas	Conteúdo de Informação
WMC	9	8	0,1177
DIT	0	0	0
NOC	0	0	0
CBO	0	0	0
RFC	9	6	0,4554
LCOM	8	3	0,9808

5 PLANO DE TRABALHO PROPOSTO

Este capítulo apresenta o plano de trabalho proposto para a continuidade do trabalho de doutorado. A primeira seção apresenta as atividades realizadas nos dois primeiros anos de trabalho e as atividades propostas para o término do trabalho. A segunda seção apresenta o cronograma para as atividades propostas.

5.1 ATIVIDADES PLANEJADAS

Durante os dois primeiros anos deste trabalho de doutorado foram realizadas, as atividades de:

- Obtenção de créditos;
- Estudo sobre a teoria de projeto axiomático;
- Estudo sobre as relações entre os conceitos do projeto axiomático, projeto de *software* e UML;
- A elaboração de uma metodologia de desenvolvimento envolvendo a teoria de projeto axiomático e a UML.

A revisão da teoria do projeto axiomático visou a assimilação dos conceitos teóricos envolvidos no projeto axiomático, além da visualização de detalhes da aplicação do projeto axiomático em estudos de caso de projetos de engenharia e, principalmente, em projetos de *software*. Nesta etapa foram efetuadas as seguintes atividades:

- Estudo aprofundado sobre a teoria de projeto axiomático envolvendo a aquisição dos conceitos teóricos do projeto axiomático;
- Revisão bibliográfica envolvendo a revisão da literatura das principais contribuições para a teoria de projeto axiomático e sua aplicação no

desenvolvimento de *software*.

- Estudo da aplicação desta teoria no desenvolvimento de *software* envolvendo o estudo sobre a aplicação do processo axiomático no desenvolvimento de *software* através de estudos de caso;

A elaboração da metodologia envolveu a avaliação da aplicação do projeto axiomático no desenvolvimento de *software* e a identificação e criação de adaptações e melhorias para a utilização do projeto axiomático no projeto de *software* orientado a objetos. Esta etapa envolveu as seguintes atividades:

- Identificação das lacunas do projeto axiomático para o desenvolvimento de *software* orientado a objetos;
- Identificação das modificações necessárias para a aplicação do projeto axiomático no desenvolvimento de *software* orientado a objetos;
- Proposição e criação de um *framework* com novos domínios e artefatos para possibilitar a utilização proposta acima;

Para a continuidade deste trabalho de doutorado, as seguintes atividades serão realizadas:

- Realização de estudos de caso, aplicando a metodologia proposta;
- Aprimoramento e ajuste da metodologia proposta;
- Investigação sobre a aplicabilidade da metodologia;
- Investigação sobre a aplicação da metodologia auxiliada por ferramentas de *software;*
- Investigação sobre a influência da metodologia sobre a qualidade de *software.*

Uma das atividades importantes será a realização de estudos de caso para avaliar a aplicabilidade da metodologia proposta. A aplicação da metodologia proposta pode revelar alguns gargalos no processo, além de ocasionar alguns efeitos colaterais, desejáveis ou indesejáveis. A realização de estudos de caso da aplicação da metodologia servirá para

identificar os ganhos, gargalos e efeitos colaterais. Estes estudos de caso serão importantes também para realizar ajustes nos parâmetros usados e na maneira de efetuar o cálculo do conteúdo de informação.

Com base nos estudos de caso, a atividade seguinte será realizar ajustes na metodologia. Estes ajustes ou modificações são importantes para que a metodologia consiga atingir os objetivos desejados. Neste caso, poderão ser modificadas etapas, a maneira de se utilizar os artefatos. A forma de calcular o conteúdo de informação poderá sofrer adaptações, principalmente nos valores dos fatores de ponderação. Isto para permitir que a aplicação da metodologia seja mais fácil e intuitiva.

Uma outra atividade importante é a investigação sobre a aplicação da metodologia auxiliada por ferramentas de *software*. Com esta atividade pretende-se identificar quais componentes da metodologia poderiam ser automatizados. Já existem ferramentas de desenvolvimento de *software* auxiliado por computador que usam aspectos do projeto axiomático, como a *IBM Rational Rose*[6] integrada com o sistema *Acclaro Scheduler*[7]. Será investigado se artefatos como a matriz de projeto e o diagrama de fluxo podem ser gerados automaticamente. Será investigado, também, se o cálculo do conteúdo de informação poderá ser feito com a ajuda de um *software*. Mas, principalmente, será investigado se a tomada de decisão sobre qual solução é a melhor poderia ser feita com a ajuda de um *software*, inclusive usando técnicas de inteligência artificial no processo.

A investigação sobre a aplicabilidade e relacionamento com outras tecnologias de desenvolvimento de *software* consistirá em analisar a aplicação da teoria de projeto axiomático em tecnologias como *Extreme Programming* (BECK, 1999), desenvolvimento orientado a aspectos (KICZALES et. al., 1997), padrões (GAMMA et al., 2000), entre outros.

É também importante investigar a influência da metodologia em aspectos de qualidade de *software* como reutilização, manutenibilidade, testabilidade, entre outros.

---

[6] IBM *Rational Rose* é marca registrada de IBM

[7] *ACCLARO* é marca registrada de *Axiomatic Design Software, Inc*..

5.2 CRONOGRAMA GERAL DAS ATIVIDADES A SEREM DESENVOLVIDAS

A Tabela 8 a seguir contém a relação das atividades a serem desenvolvidas durante o doutoramento, bem como a respectiva previsão de tempo para sua realização. A escala de tempo é mostrada em trimestres em cada célula.

TABELA 8 CRONOGRAMA DE ATIVIDADES

Atividades / Trimestres	01o. 2005	02o. 2005	03o. 2005	04o. 2005	01o. 2006	02o. 2006	03o. 2006	04o. 2006
Desenvolvimento e ajuste da Metodologia	**X**	**X**	**X**					
Estudos de caso	**X**	**X**	**X**	**X**				
Investigação sobre a aplicabilidade da metodologia			**X**	**X**				
Investigação sobre a aplicação da metodologia auxiliada por ferramentas de *software*				**X**	**X**			
Investigação sobre a influência da metodologia sobre a qualidade de *software*					**X**	**X**		
Elaboração de artigos científicos / Participação em Congressos	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Escrita da Tese							**X**	**X**
Defesa da Tese								**X**

6 CONCLUSÕES

Este trabalho propõe a criação de uma metodologia de desenvolvimento de *software* baseada na adaptação da teoria de projeto axiomático ao projeto de *software* orientado a objetos e na UML. Esta adaptação foi efetuada considerando as peculiaridades do projeto de *software* orientado a objetos, as principais ferramentas de projeto providas pela UML e os principais conceitos do projeto axiomático.

A motivação para a realização deste trabalho é tornar o projeto de *software* mais preciso e confiável de forma a garantir a qualidade dos produtos (*software*) construídos através deste processo. A teoria de projeto axiomático (SUH, 2001) foi estabelecida para garantir a qualidade de um projeto, através do uso de seus axiomas, domínios, ferramentas e processo. A teoria de projeto axiomático foi criada com o objetivo de ser independente de domínio, podendo ser aplicada em todos os tipos de projeto.

Como resultado da pesquisa espera-se obter as seguintes contribuições para a ciência e tecnologia: Uma nova abordagem para a utilização da UML no desenvolvimento de *software*, usando ferramentas e processos tirados da abordagem axiomática. Uma nova abordagem para a aplicação da teoria de projeto axiomático Uma metodologia para desenvolvimento de *software* baseada na UML e na teoria de projeto axiomático que ajude a garantir a qualidade do projeto, propriamente dito, e, portanto, do *software* produzido.

No desenvolvimento de *software* orientado a objetos ocorrem freqüentemente decisões de projeto tomadas usando-se a experiência do desenvolvedor, sem parâmetros mais precisos. Isto não garante que o projeto seja um bom projeto. A formas de avaliar um projeto são baseadas na qualidade do produto que ele irá gerar. Não se leva em consideração a qualidade do projeto em si. Na teoria de projeto axiomático temos ferramentas como os axiomas, teoremas e seus corolários (SUH, 1990), que ajudam na tomada de decisões a respeito da qualidade do projeto em si, de maneira mais precisa. A

teoria de projeto axiomático, desenvolvida para projetos de engenharia pode fornecer ferramentas para ajudar na garantia da qualidade do projeto. Com este trabalho espera-se tornar o processo de projeto de *software* mais preciso e confiável de forma a garantir a qualidade dos produtos construídos através deste processo.

Este trabalho tem por objetivo estabelecer uma metodologia que poderá ser usada para garantir a qualidade do projeto de *software*. Com esta metodologia será possível identificar que um projeto não está bom, ou seja, não satisfaz os axiomas. Neste caso, sua realização seria muito difícil, pois exigiria da equipe de desenvolvimento muito mais sincronização e comunicação do que seria necessário.

Esta metodologia poderá, também, ser incorporada a ferramentas CASE adicionando a capacidade de avaliar um projeto que está sendo executado, e dando ao projetista a oportunidade de modificar ou refazer o projeto, antes do início de sua implementação. Com isto, tenta-se evitar que a implementação, atividade que gasta mais recursos, seja feita a partir de um projeto que, por exemplo, não satisfaz os axiomas.

Com esta metodologia tenta-se, também, obter uma maior robustez no projeto de *software*. Uma das grandes dificuldades de um projeto de *software* é o gerenciamento de mudanças nos requisitos durante a elaboração do projeto. Com esta metodologia é possível controlar os efeitos negativos dessas mudanças, evitando que a mudança em um requisito tenha um efeito muito grande sobre os outros requisitos.

A metodologia, proposta neste trabalho, tem por objetivo permitir que os conceitos do projeto axiomático sejam utilizados, de maneira eficaz, para o desenvolvimento de um projeto de *software*. A metodologia é dividida em etapas, baseadas nos mapeamentos entre os domínios do projeto axiomático. Os domínios originais do projeto axiomático não são suficientes para representar todas as características de um projeto de *software* orientado a objetos. Por este motivo foram propostos novos domínios complementares. Foram usados os seguintes domínios complementares: domínio de casos de uso, domínio de classes, domínio de interações e domínio de estados.

Em DO (2004), é proposta a utilização de um domínio complementar (casos de uso) para capturar os requisitos funcionais (FRs), durante a fase de especificação de requisitos. O trabalho de doutorado, aqui apresentado, estende a utilização de domínios complementares para a fase de projeto. Além disso, este trabalho propõe uma classificação dos requisitos funcionais (FRs) em relação ao nível da hierarquia funcional e também com relação ao tipo de serviço requerido. Com relação ao nível da hierarquia, foram propostos os seguintes tipos de requisitos funcionais: casos de uso, subcasos de uso, atividades e serviços técnicos. Com relação ao tipo de serviço requerido, os serviços técnicos foram classificados em: serviços técnicos de entidade, serviços técnicos de controle e serviços técnicos de fronteira. Esta classificação procura estabelecer uma ligação entre a arquitetura MVC e o projeto axiomático, aproximando-o das tecnologias usadas, atualmente, para o projeto de *software*.

Conceitos do projeto axiomático como requisitos funcionais e parâmetros de projeto foram adaptados para o projeto de *software*. Os requisitos funcionais são representados como casos de uso, subcasos de uso, atividades e serviços técnicos. Os parâmetros de projeto são representados como colaborações, subcolaborações que são decompostas até atingir um item de projeto que é equivalente a um objeto e sua representação nas visões estática (diagrama de classes) dinâmica (diagrama de estados) e interativa (diagrama de seqüência).

O processo de decomposição funcional foi adaptado para satisfazer as necessidades do projeto de *software*. Os níveis de decomposição adotados foram: a identificação dos casos de uso; a identificação dos sub casos de uso; a identificação dos “eventos” ou atividades; a identificação dos serviços técnicos.

Com relação à decomposição funcional, este trabalho, modifica a decomposição funcional proposta em (SUH e DO, 2000), que não usa o conceito de casos de uso. Este trabalho propõe uma decomposição funcional fortemente baseada em casos de uso. Cada caso de uso é decomposto em unidades menores de serviço até alcançar o nível de

serviços prestados por um objeto a outro objeto, chamados serviços técnicos. Esta proximidade torna a decomposição muito mais próxima dos conceitos usados em projeto de *software*

Foi estabelecido um *framework* para o cálculo do conteúdo de informação, adaptado para o projeto de *software* orientado a objetos. Este cálculo foi efetuado com base em métricas de complexidade de *software* orientado a objetos, conhecidas na literatura.

Para a continuidade deste trabalho de doutorado, serão realizados estudos de caso para avaliar a aplicabilidade da metodologia proposta. A aplicação da metodologia proposta deverá ser refinada para torna-la mais intuitiva e aplicável. Desta forma tenta-se evitar gargalos e efeitos colaterais, desejáveis ou indesejáveis. Estes estudos de caso servirão para identificar estes gargalos e efeitos colaterais, além de servir para realizar ajustes nos parâmetros usados e na maneira de efetuar o cálculo do conteúdo de informação. Neste caso, poderão ser modificadas etapas, a maneira de se utilizar os artefatos e a forma de calcular o conteúdo de informação.

Será feita uma investigação sobre a aplicação da metodologia auxiliada por ferramentas de *software*. Tenta-se identificar quais componentes e etapas da metodologia poderiam ser automatizados, e como seria realizada esta automação.

Será realizada uma investigação sobre a aplicabilidade e relacionamento com outras tecnologias de desenvolvimento de *software*. É também importante investigar a influencia da metodologia em aspectos de qualidade de *software* como reutilização, adaptabilidade, entre outros.

O trabalho de doutorado aqui proposto tem um forte caráter teórico, não envolvendo a aquisição de equipamentos especiais. Recursos tradicionais como microcomputador e *software* aplicativos serão suficientes. Serão necessários ao desenvolvimento do trabalho de doutorado, itens como: livros, artigos, microcomputador e *software* aplicativos, entre outros. Estes itens deverão ser adquiridos com recursos do CPGEI, do CEFET-PR e do CNPq, via bolsa de estudos e taxa de bancada, ambas já

implementadas.

6.1 AGRADECIMENTO

O presente trabalho de doutorado está sendo realizado com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – Brasil.

ANEXO 1 - TEOREMAS E COROLÁRIOS RELACIONADOS COM OS AXIOMAS

Neste anexo são apresentados corolários e teoremas relacionados com os axiomas 1 e 2. Serão apresentados os teoremas sobre projeto em geral e os teoremas relacionados com projeto de *software*. Os corolários e teoremas apresentados abaixo foram traduzidos de (SUH, 2001).

1.1 COROLÁRIOS

Corolário 1 – Desacoplamento de projetos acoplados. Desacople ou separe as partes ou aspectos de uma solução se os requisitos funcionais estiverem acoplados ou se tornarem interdependentes no projeto proposto.

Corolário 2 – Minimização dos requisitos funcionais. Minimize o número de requisitos funcionais e restrições.

Corolário 3 – Integração das partes Físicas. Integre características de projeto em uma única parte física se os requisitos funcionais puderem ser satisfeitos independentemente na solução proposta.

Corolário 4 – Uso de padronização. Use partes padronizadas ou intercambiáveis se o uso destas partes estiver consistente com os requisitos funcionais e restrições.

Corolário 5 – Uso de simetria. Use formas e componentes simétricos se eles estiverem consistentes com os requisitos funcionais e restrições.

Corolário 6 – Maior faixa de variação de projeto. Especifique a maior faixa de variação de projeto permitido quando estiver estabelecendo os requisitos funcionais.

Corolário 7 – Projeto desacoplado com menos informação. Procure um projeto desacoplado que requer menos informação que projetos acoplados em satisfazer um

conjunto de requisitos funcionais.

Corolário 8 – Reangularidade Efetiva de um escalar. A reangularidade efetiva R para uma matriz de acoplamento escalar ou elemento de matriz é um.

1.2 TEOREMAS DE PROJETOS GERAIS

Teorema 1 – Acoplamento devido a um número insuficiente de parâmetros de projeto (DPs). Quando o número de parâmetros de projeto (DPs) é menor que o número de requisitos funcionais (FRs), ou resulta num projeto acoplado ou os requisitos funcionais (FRs) não poderão ser satisfeitos.

Teorema 2 – Redução do acoplamento de um projeto acoplado. Quando um projeto é acoplado devido a um número de requisitos funcionais (FRs) maior que o número de parâmetros de projeto (DPs) ($m > n$), seu acoplamento pode ser reduzido pela adição de novos parâmetros de projeto (DPs) de tal forma a tornar o número de requisitos funcionais (FRs) igual ao número de parâmetros de projeto (DPs) se um subconjunto da matriz de projeto contendo $n \times n$ elementos constitui uma matriz triangular.

Teorema 3 – Projeto redundante. Quando existem mais parâmetros de projeto (DPs) que requisitos funcionais (FRs), o projeto ou é um projeto redundante ou é um projeto acoplado.

Teorema 4 – Projeto ideal. Em um projeto ideal, o número de parâmetros de projeto (DPs) é igual ao número de requisitos funcionais (FRs) e os requisitos funcionais são sempre mantidos independentes uns dos outros.

Teorema 5 – Necessidade de um novo projeto. Quando um conjunto de requisitos funcionais (FRs) é alterado pela adição de um novo requisito funcional (FR), pela substituição de um dos requisitos funcionais (FRs) por um novo ou pela seleção de um conjunto completamente diferente de requisitos funcionais (FRs), a solução do projeto

Teorema 6 - Independência de caminho do projeto desacoplado. O conteúdo de

informação de um projeto desacoplado é independente da seqüência pela qual os parâmetros de projeto (DPs) são mudados para satisfazer um dado conjunto de requisitos funcionais (FRs).

Teorema 7 - Dependência de caminho de projeto acoplados e semi-acoplados. O conteúdo de informação de um projeto acoplado ou semi-acoplado depende da seqüência pela qual os parâmetros de projeto (DPs) são mudados e dos caminhos específicos de mudança destes parâmetros de projeto (DPs).

Teorema 8 – Independência e a faixa de variação de projeto. Um projeto é um projeto desacoplado quando a faixa de variação especificada pelo projetista é maior que

$$\left(\sum_{\substack{i \neq j \\ j=1}}^{n} \frac{\partial FR_i}{\partial DP_j} \Delta DP_j \right) \tag{25}$$

Neste caso, os elementos fora da diagonal da matriz de projeto podem ser desconsiderados do projeto.

Teorema 9 – Projeto para “manufaturabilidade”. Para um produto ser manufaturável com confiabilidade e robustez, a matriz de projeto para o produto, [A] (que relaciona o vetor de requisitos funcionais (FRs) para o produto com o vetor dos parâmetros de projeto (DPs) do produto), multiplicada pela matriz de projeto do processo de manufatura, [B] (que relaciona o vetor de requisitos funcionais (FRs) para o processo de manufatura com o vetor dos parâmetros de projeto (DPs) do processo), deve resultar ou em uma matriz diagonal ou em uma matriz triangular. Conseqüentemente, quando tanto [A] quanto [B] representam um projeto acoplado, a independência dos requisitos funcionais (FRs) e um projeto robusto não podem ser alcançados. Quando as matrizes forem matrizes triangulares completas, tanto ambas devem ser triangulares superiores ou ambas devem ser triangulares inferiores para que o processo de manufatura satisfaça a independência dos requisitos funcionais (FRs).

Teorema 10 – Modularidade das medidas de independência. Supondo que uma

matriz de projeto [DM] possa ser particionada em sub-matrizes quadradas que somente possuam elementos diferentes de zero na diagonal principal. Então a reangularidade e a semangularidade para [DM] são iguais ao produto das suas medidas correspondentes para cada sub-matriz.

Teorema 11 – Invariância. A reangularidade e a semangularidade para uma matriz de projeto [DM] não varia com a mudança da ordem dos requisitos funcionais (FRs) e dos parâmetros de projeto (DPs) enquanto forem preservadas as associações entre cada requisito funcional (FR) e seus correspondentes parâmetros de projeto (DPs).

Teorema 12 – Soma da informação. A soma da informação para um conjunto de eventos também é informação, desde que probabilidades condicionais apropriadas sejam usadas quando os eventos não forem estatisticamente independentes.

Teorema 13 – Conteúdo de informação de todo o sistema. Se cada parâmetro de projeto (DP) for probabilisticamente independente dos outros parâmetros de projeto (DPS), o conteúdo de informação total do sistema é a soma da informação de todos os eventos individuais associados com o conjunto de requisitos funcionais (FRs) que devem ser satisfeitos.

Teorema 14 – Conteúdo de informação de projetos acoplados *versus* desacoplados. Quando o estado dos requisitos funcionais (FRs) é mudado de um estado para o outro no domínio funcional, a informação requerida para a mudança é maior para projetos acoplados que para projetos desacoplados.

Teorema 15 – Interface projeto-manufatura. Quando um sistema de manufatura compromete a independência dos requisitos funcionais (FRs) do produto, ou o projeto do produto deve ser modificado ou um novo processo de manufatura deve ser projetado e/ou utilizado para manter a independência dos requisitos funcionais (FRs) do produto.

Teorema 16 – Igualdade do conteúdo de informação. Todos os conteúdos de informação que são relevantes para a tarefa de projetar são igualmente importantes, não importando sua origem física, e nenhum fator de ponderação deverá ser aplicado a eles.

Teorema 17 – Projeto na ausência de informação completa. O projeto pode ser feito mesmo na ausência de informação completa apenas no caso de um projeto semi-acoplado se a informação faltante está relacionada com elementos fora da diagonal.

Teorema 18 – Existência de um projeto desacoplado ou semi-acoplado. Sempre irá existir um projeto desacoplado ou semi-acoplado que possua menor conteúdo de informação que um projeto acoplado.

Teorema 19 – Robustez do projeto. Um projeto desacoplado e um projeto semi-acoplado são mais robustos que um projeto acoplado no sentido que é mais fácil reduzir o conteúdo de informação de projetos que satisfazem o axioma da independência.

Teorema 20 – Faixa de variação de projeto e acoplamento. Se as faixas de variação de projeto para projetos desacoplados ou semi-acoplados forem apertadas, os projetos podem se tornar projetos acoplados. No sentido contrário, se as faixas de variação de projeto para projetos acoplados forem relaxadas, eles podem se tornar projetos desacoplados ou semi-acoplados.

Teorema 21 – Projeto robusto quando o sistema possui uma função de distribuição de probabilidade não uniforme. Se a função de distribuição de probabilidade do requisito funcional (FR) na faixa de variação de projeto não é uniforme, a probabilidade de sucesso é igual a um quando a faixa de variação do sistema está dentro da faixa de variação de projeto.

Teorema 22 – Robustez comparativa de um projeto semi-acoplado. Dadas as faixas de variação de projeto máximas para um dado conjunto de requisitos funcionais (FRs), projetos semi-acoplados não podem ser tão robustos quanto projetos desacoplados em que as tolerâncias permitidas para parâmetros de projeto (DPs) de um projeto semi-acoplado são menores que aqueles para um projeto desacoplado.

Teorema 23 – Diminuindo a robustez de um projeto semi-acoplado. A tolerância permitida e, portanto, a robustez de um projeto semi-acoplado com uma matriz triangular completa diminui com um aumento no número de requisitos funcionais (FR).

Teorema 24 – Agendamento ótimo. Antes que um agendamento para a movimentação de um robô ou agendamento de manufatura possa ser otimizado, o projeto das tarefas deve ser feito de maneira a satisfazer o axioma de independência pela adição de desacopladores para eliminar o acoplamento. Os desacopladores podem ser na forma de uma pilha de *hardware* isolado ou *buffer*.

1.3 TEOREMAS RELACIONADOS COM PROJETO DE *SOFTWARE*

Teorema *Soft* 1 – Conhecimento requerido para operar um sistema desacoplado. Sistema de *software* ou *hardware* desacoplados podem ser operados sem um conhecimento preciso sobre elementos do projeto (módulos) se o projeto é verdadeiramente um projeto desacoplado e se as saídas dos requisitos funcionais (FRs) puderem ser monitorados para permitir um controle dos requisitos funcionais (FRs).

Teorema *Soft* 2 – Tomada de decisões corretas na ausência do conhecimento completo para um projeto semi-acoplado com controle. Quando o sistema de *software* é um projeto semi-acoplado, os requisitos funcionais (FRs) podem ser satisfeitos pela mudança dos parâmetros de projeto (DPs) se a matriz de projeto é conhecida para a extensão que o conhecimento a respeito da seqüência apropriada é dada, mesmo que não haja um conhecimento preciso a respeito dos elementos do projeto.

REFERÊNCIAS

ALTSHULLER, G. S.. *Creativity as an Exact Science*. Gordon and Breach. USA, 1988.

ANDA, B.; DREIEM, D.; SJØBERG, D.; JØRGENSEN, M.. Estimating *Software* Development Effort Based on Use Cases - Experiences from Industry. In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, LNCS 2185 Springer.

BECK, K.. Extreme Programming Explained. Embrace Change. Addison-Wesley. 1999.

BECK, K.; CUNNINGHAM, W.. *A Laboratory for Teaching Object-Oriented Thinking*. In: Proceedings of OOPSLA 89. 1989.

BITTNER, K.; SPENCER, I.. *Use Case Modeling*. Addison Wesley, Reading, Massachusetts, 2003.

BJÄRNEMO, R. *Formalized Design Methods*. Thesis. Institute for Machine *Design*. University of Lund. Sweden. 1983

BOOCH, G.. Object-Oriented Analysis and *Design* with Applications. Benjamin/Cummings. 1994.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, 1999.

CAPPETTI, N.; NADDEO, A.; PELLEGRINO, A.. *Design* Decoupling Method Based on Para-Complete Logics. In Proceedings of the 3rd International conference on axiomatic *design* – ICAD2004. Seul Korea, june, 2004.

CHIDAMBER, S. R.; KEMERER, C. F.. A Métrics Suite for Object-Oriented *Design*. In IEEE transactions on *software* engineering, vol. SE-20, no. 6, junho 1994, pp. 476-493.

CLAUSING, D. P.. *Total Quality Development*. ASME press. USA, 1994.

DO, S. H.. *Software* Product Lifecycle Management Using Axiomatic Approach. In Proceedings of the 3rd International conference on axiomatic *design* – ICAD2004. Seul Korea, june, 2004.

DO, S.; SUH, N. P.. *Systematic OO Programming with Axiomatic Design.* IEEE Computer 32(10): 121-124. 1999.

D'SOUZA, D. F.; WILLS, A. C.. Objects, Components, and Frameworks With Uml: The Catalysis Approach. ADDISON-WESLEY. 1998

GANE, C.; SARSON,.T.. *Análise Estruturada de Sistemas*. LTC editora. Brasil. 1995.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J.. Padrões de Projeto. Soluções Reutilizáveis de *Software* Orientado a Objetos. Bookman editora. 2000.

HALSTEAD, M.; *Elements of Software Science*, North-Holland. 1977.

HAREL, D.. Statecharts: A Visual Formalism for Complex Systems. In: Science of computer Programming, v. 8, p. 231-274. 1987.

HARUTUNIAN, V.; NORLUND, M. and TATE, D.. *Decision Making and Software Tools for Product Development Based on Axiomatic Design Theory*. Submited to the 1996 CIRP General Assembly (CIRP Annals, vol. 45/1). Como, Italy. 1996

HINTERSTEINER, J. D.. A fractal representation for systems. In the 1999 International CIRP *Design* Seminar, Enschede, The Netherlands, 1999.

HINTERSTEINER, J. D. and NAIN, A. S.. *Integrating Software Into Systems: An Axiomatic Design Approach*. Proceedings of the 3rd. International Conference on Engineering Desing and Automation. Vancouver, Canada. 1999.

HUBKA, V.; EDER, W. E.. *Engineering Design*. Heurista. Switzerland, 1992.

JACOBSON, I.. Object-Oriented *Software* Engineering: A Use Case Driven Approach. Addison-Wesley. 1994.

JACOBSON, I.; BOOCH , G.; RUMBAUGH , J.. *The Unified Software Development Process*. Addison Wesley, Reading, Massachusetts, 1999.

KRASNER, G.; POPE, S.. A Cookbook for using the Model-View Controller user paradigm in Smalltalk-80. Journal of Object-Oriented programming, vol. 1, no. 3, august/September 1988. pp. 26-49.

KARNER, G, “Metrics for Objectory”. Diploma thesis, University of Linköping, Sweden. No. LiTH- IDA-Ex-9344:21. December 1993.

KICZALES, G. J.; LAMPING, A.; MENDHEKAR, C.; MAEDA, C.; LOPES, J.; LOINGTIER, M.; IRWIN, J.. Aspect-Oriented Programming. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland, Springer-Verlag LNCS 1241, june, 1997.

LARMAN, C.. Applying UML and patterns – An Introduction to Object-Oriented Analysis and *Design*. Prentice-Hall. 1997.

LEFFINGWELL, D.; WIDRIG, D.. *Managing Software Requirements: A use case approach*. 2nd edition. Addison-Wesley. Boston, 2003.

LINDHOLM, D.; TATE, D.; HARUTUNIAN, V.. Consequences of *Design* Decisions in Axiomatic *Design*. In: Journal of Integrated *Design* and Process Science. Society for *Design* and Process Science. December 1999, v. 3 no. 4 pp. 1-12.

MCMENAMIM, S.; PALMER, J. F.. *Análise Essencial de Sistemas.* Makron Books. Brasil. 1991.

NORLUND, M.. *An Information Framework for Engineering Design based on Axiomatic Design*. Ph.D. Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, Nov 1996.

OBJECT MANAGEMENT GROUP. *UML 2.0 super structure specification*. Disponível em http://www.omg.org/docs/ptc/03-08-02.pdf. 2003

OLEWNIK, A. T.; LEWIS, K. E.. *On validating design decision methodologies*. Proceedings of DETC 2003 ASME 2003 *Design* Engineering Technical Conferences. Chicago, IL, September 2- 6.

PAULK, M. C.; CURTIS, B.; CHRISSIS, M. B.; WEBER, C. V.. *Capability Maturity Model℠ for Software, Version 1.1*. Technical Report CMU/SEI-93-TR-024 ESC-TR-93-177. February, 1993.

PRESSMAN, R. S.. *Software Engineering: A practitioner's approach*. 6a ed. McGraw-Hill. New York, 2004.

RUDOLPH, S.. *On a Mathematical Foundation of Axiomatic Design*. Proceedings of the 1996 ASME *Design* Engineering Technical Conference and Computers in Engineering Conference. USA. 1996.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.. *Object-Oriented Modeling and Design*. Prentice Hall. 1991.

SCHREYER, M.; TSENG M. M.. Hierarchical State Decomposition for the *design* of PLC *Software* by Applying Axiomatic *Design*. In: Proceedings of ICAD2000 – First International Conference On Axiomatic *Design*. Cambridge, MA. June, 2000.

SHIN, G. S.; YI, J. W.; YI, S. I.; KWON, Y. D.; PARK, G. J.. Calculation of Information Content in Axiomatic *Design*. In Proceedings of the 3rd International conference on axiomatic *design* – ICAD2004. Seul Korea, june, 2004.

STADZISZ, P. C.. *Contribuition à une Méthodologie de Conception Intégrée des Familles de Produits pour l'Assemblage*. L'Universite de Franche-Comte. (Tese de doutorado). 1997.

SUH, N. P.. *Axiomatic design: Advances and Applications*. Oxford University Press. New York, 2001.

SUH, N. P.. *The Principles of Design*. Oxford University Press. New York, 1990.

SUH, N. P.; BELL, A. C.; GOSSARD, D. C.. *On an Axiomatic Approach to Manufactoring and Manufactoring Systems*. Journal of Engineering for Industry. Vol. 100, no. 2, p. 127-130. 1978.

SUH, N. P.; DO, S.. *Axiomatic Design of Software Systems.* .In: CIRP Annals. Vol. 49, n. 100, p. 95-100. Sydney, Australia, 2000.

TAGUCHI, G.. Systems of Engineering *Design*: Engineering Methos to Optimize Quality and Minimize Cost. American Supplier Institute. Dearborn, MI, 1987.

TATE, D.. *A Roadmap for decomposition: Activities, Theories, and Tools for System Design*. PhD thesis, Department of Mechanical Engineering at Massachusetts Institute of Technology, Cambridge, Massachusetts, 1999.

TATE, D.; NORLUND, M.. A *Design* Process Roadmap as a General Tool for Structuring and Supporting *Design* Activities. In: Proceedings of the Second World Conference on Integrated *Design* and Process Tecnology (IDPT – vol. 3). Society for *Design* and Process Science, Austin TX. Pp. 97-104. December 1996.

ULRICH, K. T.; EPPINGER, S. D.. *Product Design and Development*. McGraw-Hill. New York: 2003.

VAZQUEZ, C. E.; SIMÕES, G. S.; ALBERT, R. M.. *Análise de Pontos de Função: medição, estimativas e gerenciamento de projetos de software*. Érica. São Paulo: 2003.

ZADEH, L. A.; FU, K.S.; TANAKA, K.; SHIMURA, M.. *Fuzzy Sets and their Applications to Cognitive and Decision Processes*. Academic press.1974.