
Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática

# MODELAGEM DA MPRIME ONTOLOGY EM LÓGICA DESCRITIVA E IMPLEMENTAÇÃO

TRABALHO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

**Aluno:** Fernando Valeriano de Almeida Lins (fval@cin.ufpe.br)

**Orientador:** Frederico Luiz Gonçalves de Freitas (fred@cin.ufpe.br)

**Co-Orientadora:** Cristine Martins Gomes de Gusmão (cristine@dsc.upe.br)

**Recife**, 23 de Agosto de 2007.

Sonho que se sonha só é só sonho, sonho que se sonha junto é realidade.
(John W. Lennon)

# Dedicatória

Dedico este trabalho, que simboliza minha formatura, a todos que sonharam comigo e fizeram este sonho se tornar realidade.

# Agradecimentos

Agradeço inicialmente a meus pais e meu irmão, por todo amor, apoio e compreensão, que mesmo a distância, sempre me deram a base para minha vida.
A minha namorada por compreender minha ausência durante esse trabalho, e por todo carinho e apoio.
A todos os amigos e familiares de Maceió que souberam compreender minha ausência, e que sempre me receberam de braços abertos.
Aos amigos de Recife, que constituíram uma segunda família para mim, principalmente Gleifer e Pablo amigos de todas as horas, e Vinícius, Thiago e Julio meus colegas de apartamento, todos meus irmãos de Recife.
A todos os colegas de turma e da turma de engenharia, que sempre ajudaram nos momentos difíceis, durante toda a graduação.
Ao pessoal do Sig@, que me apoiou e soube compreender minhas faltas devido a esse trabalho.
Ao Centro de Informática e toda sua equipe, por proverem a estrutura e tudo o mais para a finalização do meu curso.
A Cristine e Hermano pela ajuda e compreensão dos últimos anos, no trabalho no mPRIME e da OntoPRIME.
Ao professor Fred por me acolher num momento de dificuldade e fazer possível a finalização deste trabalho.
E por fim agradeço ao grande amigo João Paulo (pirulito) que teve a paciência de passar o corretor ortográfico neste trabalho.

# Resumo

Este trabalho tem o intuito de redefinir a OntoPRIME, Ontologia de Riscos para Ambientes de Desenvolvimento de Software Multiprojetos [GUSMÃO et al, 2004]. A OntoPRIME foi construída usando Lógica de Primeira Ordem (LPO), essa escolha se deu inicialmente pelo conhecimento que a equipe já tinha da mesma e também pelo reconhecimento que tem na comunidade acadêmica, mas com o passar do tempo viu-se que a ontologia necessitava de uma maior expressividade, e também de uma linguagem mais popular em ontologias. Por esses motivos foi proposto esse trabalho de redefinição da OntoPRIME, dessa vez em lógica descritiva, dando-se o nome dessa nova versão de mPRIME Ontology (Multiple Project RIsk ManagEment Ontology).

Para fins de testes e provas de conceito, este trabalho também apresentará a definição da ontologia na ferramenta Protégé [PROTEGE], que é uma ferramenta *Open Source*, muito utilizada na construção de Ontologias e que traz uma série de funcionalidades bastante úteis. Com esta definição espera-se ter uma maior facilidade de edição e correções no futuro, além de ajudar no compartilhamento da mesma.

**Palavras chaves:** Gerência de Riscos, Riscos, Ontologia, OntoPRIME, mPRIME Ontology.

# Índice

# Índice de Figuras

# Índice de Tabelas

# 1. Introdução

Desde que Barry Boehm [BOEHM, 1991] e Robert Charette [CHARETTE, 1990], trouxeram a atenção da comunidade da engenharia de software para a gerência de riscos, várias novas abordagens foram propostas e a Gerência de Riscos foi se tornando cada vez mais popular. Mas mesmo com essa popularização ainda é bastante comum encontrar gerentes que realizam apenas a gerência reativa, reagindo aos riscos apenas quando eles ocorrem, enquanto o mais indicado seria uma gerência pró-ativa, que identificasse o risco antes que ele ocorresse.

Muitos desses problemas se devem ao fato de que ainda não existe uma cultura de Gerência de Riscos dentro das empresas, a maioria dos gerentes tem um conhecimento limitado da área e não se sentem seguros, nem aptos a realizar essa gerência. Para difundir os conhecimentos acerca da Gestão de Riscos e ao mesmo tempo prover uma maior segurança para o gerente de projetos é necessária a definição de padrões, técnicas, ferramentas e metodologias sobre o tema.

Dessa necessidade surge a OntoPRIME, uma ontologia de riscos, criada com o intuito de dar suporte ao processo de Gerência de Riscos em Ambientes de Desenvolvimento de Software de Múltiplos Projetos, além de realizar um estudo científico sobre o relacionamento dos riscos dentro dos projetos. Ela provê um vocabulário único para representação de conhecimento, além de fornecer uma estrutura que permite a organização do conhecimento sobre riscos.

A proposta deste trabalho é realizar uma análise sobre a Onto*P*RIME, revendo toda sua estrutura e seus axiomas, propondo melhorias e correções, e a partir daí redefinir a mesma, que inicialmente foi construída em Lógica de Primeira Ordem, agora em Lógica Descritiva, uma família de linguagens bem mais expressivas e por isso mais difundida na descrição de ontologias.

Mais especificamente a redefinição será realizada em OWL-DL (*Web Ontology Language,* o DL vem de Lógica Descritiva) linguagem recomendada pela W3C. Nesta redefinição deverá ser levada em conta a nova arquitetura que a linguagem propõe para a

ontologia, e também serão analisadas expansões tirando o proveito da maior expressividade da mesma.

Além disso, a ontologia será definida na Ferramenta Protégé [PROTEGE], ferramenta bastante difundia no meio acadêmico, sendo o editor de ontologias mais popular na atualidade, com esta definição pretende-se facilitar a expansão, correções e compartilhamento da ontologia.

## 1.1 Objetivos da Monografia

Neste trabalho serão discutidos, inicialmente os conceitos e estudos necessários, para que ele fosse realizado, para que em seguida seja descrito todo o processo de desenvolvimento da mPRIME Ontology, mostrando os problemas encontrados, soluções tomadas, chegando ao estado final da ontologia. Por fim serão apresentadas as conclusões sobre o trabalho, assim como os trabalhos futuros sugeridos.

## 1.2 Metodologia empregada

Vamos abaixo descrever todas as fases necessárias para o desenvolvimento deste trabalho, detalhando como cada fase deverá ser executada.

***Fase 1:*** **Estudo do estado atual da *OntoP*RIME e de como riscos são tratados na Engenharia de Software.**

Ver todas as fases por que passou a OntoPRIME, analisando inicialmente sua origem na Taxonomia do SEI, estudando cada um dos axiomas gerados durante sua construção, e por fim entender o funcionamento da mesma dentro do protótipo do *Ontology Risk Assistant* e da mPRIME *tool*.

Nesta fase também deverá ser analisada e revistas, a evolução e o estado atual da gerência de riscos na engenharia de software, discutindo técnicas, métodos e ferramentas usadas na gerência de riscos.

***Fase 2:*** **Estudo de ontologias e da Lógica Descritiva.**

Conhecer melhor os conceitos e definições inerentes a ontologias, analisando seus componentes, classificações e linguagens de desenvolvimento. Dentro deste objetivo deverão ser estudados não só livros e artigos como também ontologias já existentes.

Em seguida haverá a pesquisa em Lógica Descritiva, estudando cada minúcia dessa linguagem, assim como sua forma de definir ontologias.

***Fase 3:*** **Estudo e análise da ferramenta para edição de ontologias Protégé [PROTEGE].**

O Protégé, é um dos editores de ontologias mais populares na comunidade acadêmica, por isso foi escolhido para a definição da mPRIME Ontology, nessa fase serão estudadas suas funcionalidades, e pretende-se ao final estar bem familiarizado com a ferramenta.

***Fase 4:*** **Definição da ontologia.**

Inicialmente serão passados todos os axiomas da lógica de primeira ordem para lógica descritiva (mais especificamente OWL), fazendo uma cuidadosa análise desses axiomas para que se possa tirar proveito da maior expressividade da lógica descritiva.

Em seguida esses axiomas deverão ser definidos no Protégé.

## 1.3 Estrutura do Trabalho

Os capítulos seguintes desse trabalho estão estruturados da forma que segue:

- Capítulo 2 – **Riscos na Engenharia de Software** - Neste capítulo são apresentados os conceitos fundamentais de riscos e é feita uma breve análise sobre a gerência de riscos em projetos de software.
- Capítulo 3 – **Ontologias** - Neste capítulo serão apresentados os conceitos sobre ontologias, mostrando sua origem etimológica, a evolução do termo dentro da informática, além da composição de uma ontologia, das classificações adotadas na história, e também mostrar as linguagens usadas para definir ontologias, por fim

daremos uma ênfase maior na linguagem OWL, que acabou sendo a escolhida para a construção da mPRIME Ontology.

- Capítulo 4 – **OntoPRIME** - Neste capítulo será descrito o estado atual da OntoPRIME, mostrando desde sua motivação até a sua construção, para que com esse estudo seja possível redefini-la da melhor forma possível.
- Capítulo 5 – **mPRIME Ontology** - Neste Capítulo será apresentada a mPRIME Ontology, assim como as soluções adotadas para sua construção, os problemas ultrapassados ou não.
- Capítulo 6 – **Conclusões** - Neste capítulo serão apresentadas as conclusões tiradas do trabalho, e propostos trabalhos futuros para a mPRIME Ontology .

# 2. Riscos na Engenharia de Software

Existem várias definições e usos diferentes para a palavra risco, talvez a mais conhecida tenha sido proposta por Frank Knight [KNIGHT, 1921] onde risco é a exposição a eventos incertos com probabilidades conhecidas, enquanto que incerteza seria a exposição a eventos incertos com probabilidades também incertas. Já segundo Peter Bernstein [BERNSTEIN, 1997], a palavra risco tem origem na língua italiana, e vem de *riscare* que significa ousar, a partir desse conceito risco deixa de ser o destino para ser uma escolha, e a capacidade de controlar riscos, saber correr riscos e evitá-los, podem diferenciar o sucesso do fracasso.

Na Engenharia de Software, o conceito de risco vem deixando de ser usado como exposição a perigo e impacto negativo. Está associado também à "exposição a conseqüências da incerteza", a partir desse momento passa a implicar que um risco tanto pode ter conseqüência negativa como positiva, criando uma idéia que de início parece estranha, mas pode ser considerada bem cabível que é a de riscos positivos.

Se um projeto está sendo desenvolvido por um grupo de pessoas, é factível pensar que esse projeto, durante todo seu ciclo (definição, implementação, aplicação...), deve estar relacionado a uma série de riscos. Citando como exemplo há o risco de a equipe ser inexperiente ou de o prazo ser muito curto. Quanto mais for possível evitar os riscos negativos, maximizar as chances de riscos positivos e mitigar os riscos que vierem a acontecer, maior será a chance de o projeto obter sucesso. A partir dessa necessidade, surge a idéia da Gerência de Riscos.

Barry Boehm [BOEHM, 1991] e Robert Charette [CHARETTE, 1990], foram os responsáveis por trazer a atenção da comunidade de Engenharia de Software para a necessidade de gerir riscos, e apesar de em seguida terem surgidos várias abordagens para tratar riscos dentro de um projeto, a realidade é que na prática a maioria das empresas ainda não tem uma aplicação efetiva da Gerência de Riscos em seus projetos. Tom De

Marco [DeMARCO, 2003], em seu livro “Waltzing with Bears”, diz que o processo de gerência de risco ainda está no jardim da infância e que muitas organizações não têm a preocupação de institucionalizar o gerenciamento de risco.

A maioria dos gerentes de projeto de software utiliza a estratégia de gerenciamento de risco reativo, que nada mais é do que reagir ao risco de acordo com a ocorrência. Mais barato e lógico é ter atitude pró-ativa. A estratégia de gerenciamento de risco pró-ativa começa pelo planejamento da gerência de risco e é um processo de âmbito organizacional.

Para ser bem-sucedida, a organização deve estar comprometida com uma abordagem de riscos proativa e consistente durante todo o projeto [PMI 2004]. Com esta citação o Guia PMBOK demonstra a crescente preocupação da comunidade de Engenharia de Software com a gestão de riscos como parte fundamental da gerência de projetos.

## 2.1 Atividades de um Processo de Gestão de Riscos

Apesar de vários modelos diferentes terem sido propostos para a gerência de riscos, na área de Engenharia de Software [HIGUERA, 1994], parece haver um consenso entre uma lista de atividades básicas que sempre devem ser seguidas. Vale salientar que estas atividades devem ser seguidas de forma cíclica e contínua.

- Planejar a gerência de riscos – Atividade relacionada à definição de uma estratégia para a gestão de riscos.
- Identificar riscos – Atividade onde se levantam todas as possibilidades de riscos do projeto, e a partir desse levantamento é elaborada uma planilha inicial de riscos, que deverá ser atualizada ao longo do projeto.
- Analisar riscos – Atividade onde cada risco é caracterizado, quantitativa e qualitativamente, com o intuito de tratá-los da melhor forma possível. A forma como o risco é caracterizado e a estratégia para tratamento dos riscos devem ser definidos na atividade de planejamento da gerência de riscos.

- Planejar respostas aos riscos – Levando-se em conta as estratégias e o nível de tolerância definidos, nesta atividade analisa-se as possíveis atitudes que devem ser tomadas caso um risco aconteça, esta análise é feita baseada no custo da resposta e do acontecimento do risco. Por exemplo, pode ser mais caro para uma empresa se um risco acontecer do que tratar para que ele não aconteça.
- Monitorar Riscos – Esta atividade trata de atualizar a caracterização dos riscos de acordo com o planejado, para que a gerência de riscos possa atuar de forma preventiva e não reativa. Com a realização dessa atividade é possível que toda a equipe entenda com maior exatidão o que está acontecendo no projeto.
- Controlar Riscos – Envolve a alteração das estratégias de mitigação, utilização de ações, previamente planejadas, de contingência e encerramento de trabalhos em relação a um risco quando o mesmo deixar de existir.
- Comunicar os riscos – Trata-se da comunicação entre os membros da equipe, uma planilha compartilhada pode ser uma forma de comunicar os riscos a toda equipe, comunicação tem se mostrado um dos fatores mais importantes para uma gerência de riscos bem sucedida.

## 2.2 Evolução de Modelos e Abordagens

Durante a história, diversas abordagens, foram propostas para a gerência de riscos, a figura abaixo mostra algumas das mais reconhecidas pela comunidade de engenharia de software, ela acompanha o surgimento em relação ao tempo. Podemos ver toda a evolução dos modelos, desde o modelo espiral proposto por Barry Boehm, até os mais modernos propostos pelo SEI e pela ISO, passando por todos os marcos que fizeram parte do desenvolvimento dos modelos.

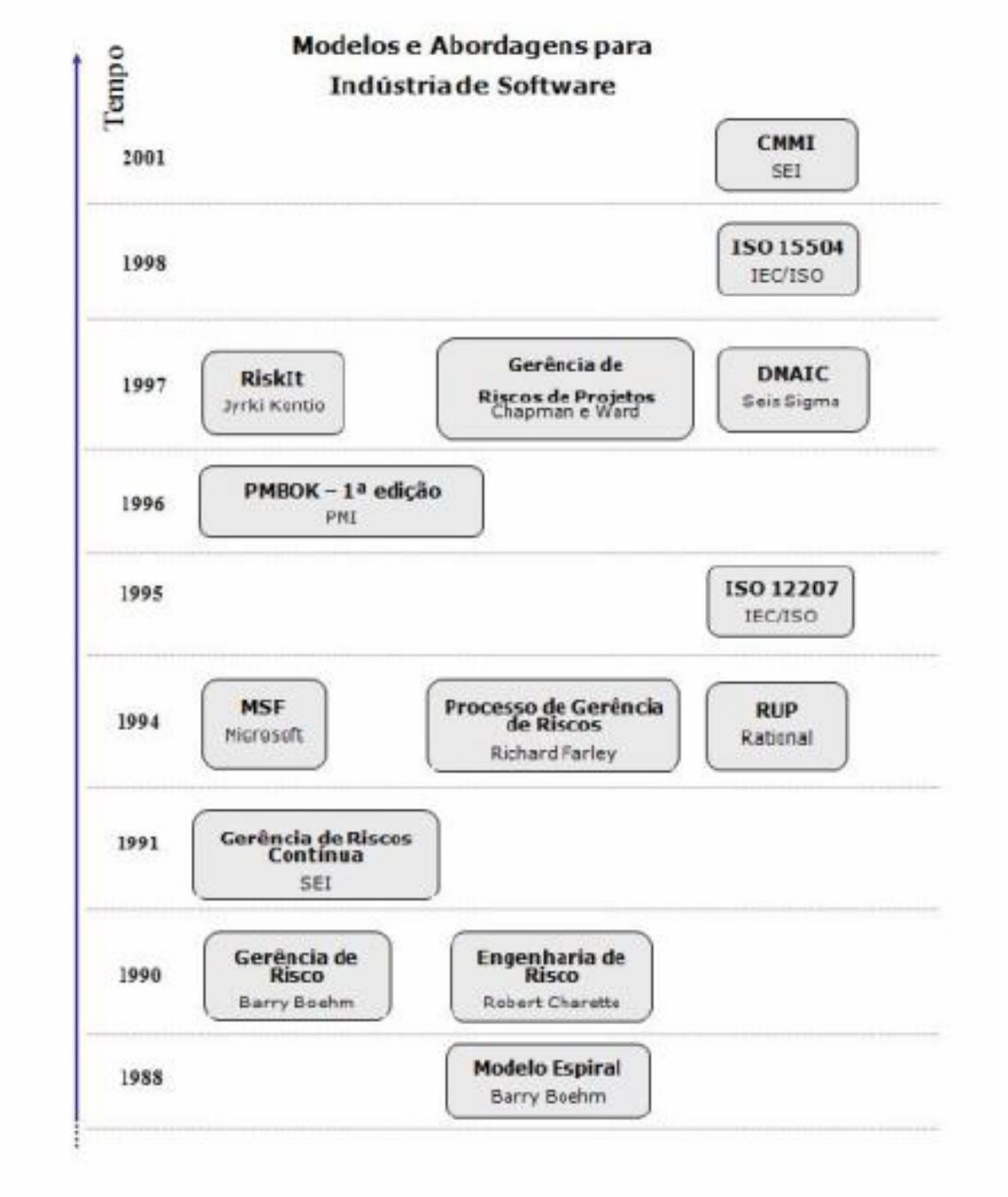


**Ilustração 2.1 - Históricos de abordagens para a gerência de riscos [Gusmão 2007].**

Incluindo abordagens propostas pela academia, como o processo de sete passos proposto pelo professor Richard Fairley [FAIRLEY, 1994] ou o de nove passos proposto por Chris Chapman e Stephen Ward [CHAPMAN & WARD, 1997]. Abordagens em processos descritivos como os modelos MSF (Microsoft Solutions Framework) [ROBIN

et al 2002] e o método *RiskIt* [KONTIO 2001]. Também abordagens definidas em modelos de desenvolvimento como a do RUP (Rational Unified Process).

Os mais populares atualmente são os modelos do PMI (Project Management Institute) composto por seis processos [PMI, 2004], os modelos CMMI (Capability Maturity Model Integration) definidos possuindo três fases (análise, controle e relatórios) [SEI, 2001] e os modelos ISO.

## 2.3 Métodos, Técnicas e Ferramentas

### 2.3.1 Métodos para identificação de riscos

Uma grande variedade de métodos, para a identificação de riscos, está disponível na literatura de Engenharia de Software [BOEHM 1991, HIGUERA 1994, MOYNIHAM 1997, MACHADO 2002, PRESSMAN 2006]. Alguns mais simples outros mais complexos, cabe ao gerente da equipe, analisar qual dos métodos é o mais indicado para ser usado.

Estando entre as mais populares, segundo pesquisa realizada pelo PMI [PMI, 2006], *Brainstorming*, Entrevista, Revisão de Documentos, Listas de Verificação, Técnica Delphi, Análise SWOT e Análise Causal.

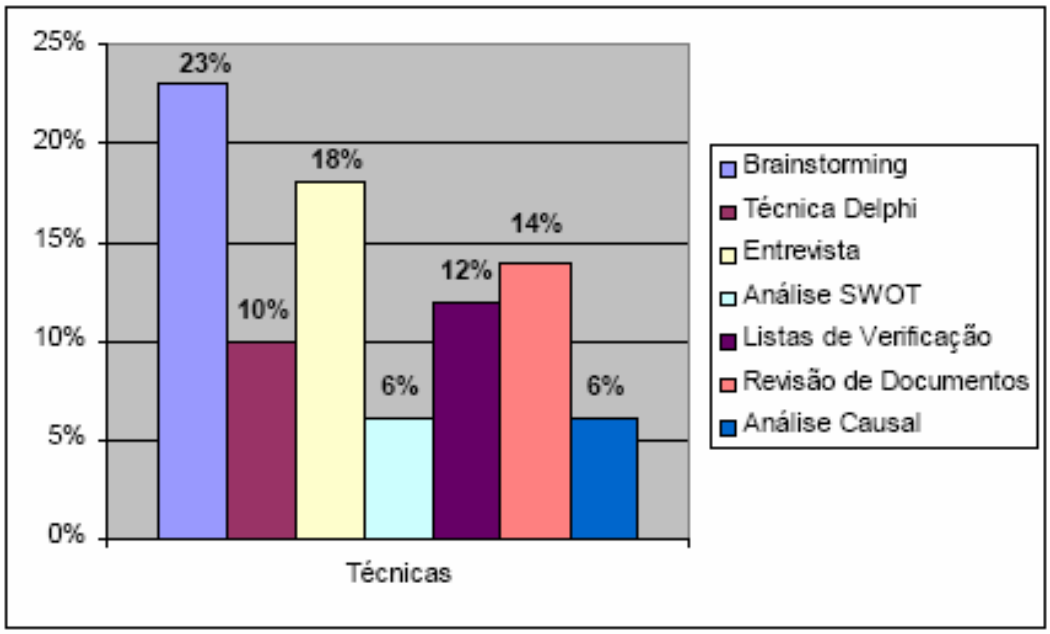


**Ilustração 2.2 - Pesquisa de popularidade de métodos de gerência de riscos [Gusmão 2007].**

Abaixo temos uma descrição de alguns dos métodos mais populares:

- *Brainstorming* – Atividade realizada com vários componentes da equipe, onde cada um deverá sugerir riscos e ao final todas as sugestões são documentadas para posterior análise.
- Entrevista – Questionário realizado para levantamento e identificação de riscos, podendo realizar as entrevistas individualmente ou através de grupos focais [VICTORIA et al, 2000], tem as desvantagens de o questionário limitar a análise, e de ter grande dependência do entrevistador.
- Listas de verificação – são comumente usadas para identificar os riscos associados a um processo e para assegurar a concordância entre as atividades desenvolvidas e os procedimentos operacionais padronizados. Limita a identificação aos itens da lista de riscos.
- Técnica Delphi – Usada com o intuito de chegar a um consenso, entre especialistas, sobre um determinado evento futuro, tem três condições básicas: o anonimato dos participantes, a representação estatística dos resultados, e o *feedback* das respostas para subseqüente re-avalização [MARTINO, 1993]. Trata-se de um questionário interativo, respondido anonimamente, onde cada participante responde quantitativamente às perguntas, com justificativas, e essas respostas são analisadas

estatisticamente e retornadas aos participantes que devem rever suas respostas em busca de um consenso.

- Análise SWAT - A análise SWOT (Strength, Weakness, Opportunity and Threats), costuma ser dividida em ambiente interno (forças e fraquezas) e ambiente externo (oportunidades e ameaças) e tem o objetivo de identificar forças e fraquezas da empresa, para tirar proveito das forças, e amenizar as fraquezas; ao mesmo tempo em que monitora o mercado com o intuito de aproveitar oportunidades e evitar ameaças. Por exemplo, se uma empresa tem como fraqueza o desenvolvimento de hardware, e surge um novo projeto nessa área, os riscos relativos ao projeto devem ser potencializados.

### 2.3.2 Técnicas de Análise de riscos

Após a identificação dos riscos de um projeto, seguindo os passos apresentados em 2.1, partimos para a análise dos mesmos. Esta análise costuma ser dividida em quantitativa e qualitativa, sendo quantitativa a análise baseada em números, estatística. Já a qualitativa é baseada no conhecimento de quem a realiza, por esse motivo, em geral, a análise qualitativa é feita por especialistas da área.

Existe uma série de técnicas para análise de riscos, sendo usadas pela indústria de software, a seguir temos uma breve descrição das mais populares, tanto quantitativas quanto qualitativas.

#### 2.3.2.1 Técnicas Quantitativas

Algumas das principais técnicas quantitativas citadas, para uso na gerência de riscos, são árvores de decisão e simulações [BOEHM, 1991], [PMI, 2004].

- Árvores de decisão – Esta técnica baseia-se em dividir o problema em um número de subconjuntos de forma que seja facilitada a solução desses problemas menores e por conseqüência, tem-se facilitada a tomada de decisão.
- Simulações – Técnicas de simulação costumam ser feitas de forma automática, logo se faz necessária uma ferramenta que realize o processo. Uma das técnicas de simulações mais populares é a análise de Monte Carlo [CSEP 1995], [HELDMAN 2005], ela é realizada das atividades do projeto sob o critério de quantidade de horas e

custos de execução, realizando uma série de simulações com o uso de variáveis definidas, obtendo um conjunto de valores das cargas horárias e custos das atividades do projeto.

#### 2.3.2.2 Técnicas Qualitativas

Técnicas Qualitativas vêm se popularizando, na gerência de riscos, devido ao fato de que a análise quantitativa não leva em conta uma série de pequenos detalhes que influenciam cada projeto. A análise qualitativa leva em conta a interpretação de especialistas, que estão presenciando o desenvolvimento do projeto, logo podem perceber minúcias que uma análise quantitativa deixaria passar.

Abaixo veremos algumas técnicas populares da análise qualitativa.

- **Cálculo da exposição ao risco** – Este se configura na multiplicação da probabilidade pelo impacto de cada risco, sendo probabilidade e impacto, variáveis definidas pelo gerente do projeto. Com a exposição de cada risco da nossa matriz podemos priorizá-los e definir planos de contingência e mitigação para cada um deles, de acordo com seu grau de exposição.
- **Escalas de probabilidade e impacto** – A definição da probabilidade e do impacto de um risco costuma ser feita pelo gerente do projeto, de forma subjetiva. Para facilitar essa definição podemos definir uma escala, com valores agregados, diminuindo a quantidade de possibilidades, logo simplifica o entendimento da equipe sobre as variáveis.

### 2.3.3 Ferramentas

A gerência de riscos tem se popularizado cada vez mais em projetos de engenharia de software, mas a execução de uma gerência de riscos bem elaborada, seguindo padrões e técnicas, muitas vezes não é fácil. Principalmente para projetos muito grandes e equipes não preparadas. A partir desse problema, surge uma série de soluções em forma de ferramentas, para auxiliar o gerente durante a gerência de riscos.

Muitas ferramentas garantem aderência a modelos de gerência de riscos, algumas são voltadas para especialistas, outras para usuários inexperientes. Cabe ao gerente escolher a ferramenta que melhor soluciona seus problemas.

Abaixo temos uma tabela comparativa entre algumas das ferramentas acadêmicas, para gerência de riscos.

**Tabela 2.1 - Comparativo de ferramentas para gerência de riscos [GUSMÃO, 2007].**

| FERRAMENTAS | ATIVIDADES DO PROCESSO DE GERENCIAMENTO DE RISCOS | | | | | | |
|---|---|---|---|---|---|---|---|
| | Planejar Gerência de Riscos | Identificar Riscos | Analisar Riscos | Planejar Respostas aos Riscos | Monitorar Riscos | Controlar Riscos | Comunicar Riscos |
| RISKMAN | NC | C | NC | NC | NC | NC | C |
| RITO | NC | C | C | C | NC | NC | C |
| RAMP | NC | C | C | NC | NC | NC | C |
| SERIM | NC | C | C | C | NC | NC | C |
| SAGRES | NC | C | C | C | C | C | C |
| RISKGUIDE | NC | C | C | C | NC | NC | C |
| RISCPLAN | NC | C | C | C | C | NC | C |
| RISKFREE | C | C | C | C | C | C | NC |
| RISK RADAR | NC | C | C | NC | C | NC | C |

**Legenda: Funcionalidade Contemplada (C) e Não Contemplada (NC)**

Esta tabela faz um comparativo entre as ferramentas existentes no mercado mostrando se elas contemplam ou não, cada uma das atividades do processo de gerenciamento de riscos, podemos ver, por exemplo, que a ferramenta Sagres contempla quase que a totalidade das atividades, deixando de fora apenas a fase do planejamento. Enquanto que a Riskman, contempla apenas as atividades de identificação e comunicação dos riscos. Cabe ao gerente do projeto, identificar qual ferramenta pode lhe ajudar da melhor forma.

# 3. Ontologias

O termo ontologia é originário da filosofia (grego *ontos*+*logoi* = "conhecimento do ser"), foi introduzido por Aristóteles dando nome a um ramo da metafísica que se importa com a existência e a organização do ser. O termo veio a ser adaptado pela comunidade de inteligência artificial e acabou assumindo um significado diferente dentro da ciência da computação.

Na visão de Uschold e Gruninger (1996), ontologia é um termo usado para se referir a uma concepção compartilhada de algum domínio de interesse, e pode ser utilizada para unificar o processo de solução de problemas relativos ao domínio em questão. Semelhantemente Fensel afirma que uma ontologia é uma especificação explicita de uma conceitualização compartilhada [FENSEL, 2001].

Uma definição complementar às passadas é a dada por Goméz-Perez, de que uma ontologia é um conjunto de termos ordenados hierarquicamente para descrever um domínio que pode ser usado como um esqueleto para a base de conhecimentos [GOMEZ & PEREZ, 1999].

Um fator que vem popularizando as ontologias é o aumento exponencial que a troca de informações vem tendo, principalmente por causa da internet, e um grande problema enfrentado nessas trocas, é a falta de uma padronização, muitas vezes termos diferentes são usados impossibilitando o reconhecimento da informação dos dois lados. Daí entram as ontologias definindo um vocabulário comum para determinado domínio, para que o mesmo possa ser reutilizado e compartilhado.

Ontologias vêm sendo usadas em várias áreas diferentes, como web semântica, inteligência artificial ou bioinformática como uma forma de representar conhecimento sobre o mundo ou parte dele.

## 3.1 Porque usar uma Ontologia?

Uma Ontologia pode ser criada com diversos fins, e posteriormente ser usada de diversas formas diferentes, são úteis para apoiar a especificação e implementação de qualquer sistema de computação complexo. Em geral ontologias fornecem uma infra-estrutura para integrar sistemas inteligentes no nível do conhecimento, trazendo para os mesmos uma série de benefícios como:

- A criação de um vocabulário comum para a representação do conhecimento. Esse vocabulário traz uma conceitualização que o sustenta, evitando interpretações ambíguas e o uso de definições diferentes para o mesmo conceito.
- Uma ontologia permite o compartilhamento de conhecimento entre membros interdisciplinares de uma equipe e também entre membros de equipes diferentes. Por exemplo, se temos dois sistemas desenvolvidos para prontuários de hospitais, e esses sistemas fizeram uso de uma mesma ontologia durante o desenvolvimento, um membro da equipe de um hospital terá maior facilidade para entender o sistema do outro hospital.
- Facilitam a integração de informação entre diferentes bases de dados, caso seja necessário dois hospitais integrarem sua base de dados, o problema terá uma solução bem mais fácil caso os dois sistemas tenha sido desenvolvidos com o uso da mesma ontologia.
- O desenvolvimento de uma ontologia traz como produto uma base de conhecimento sobre o domínio escolhido, servindo assim como base para estudos de pessoas que queiram entender melhor sobre o assunto.
- Ontologias são feitas com o objetivo de serem flexíveis, para não só poderem ser usadas pelas mais diversas aplicações, como também para poderem ser estendida das mais diversas formas, além de poderem ser passadas para linguagens diferentes.

Uma ontologia de qualidade deve levar em conta, todos esses propósitos, é normal que uma ontologia tenha um foco maior em algum específico, mas é sempre importante

levar os outros em consideração, para que no futuro a ontologia possa ser reusada e compartilhada.

## 3.2 Composição de uma Ontologia

Existem várias descrições diferentes sobre a composição das ontologias, podemos citar duas com maior importância, [MAEDCHE, 2001] e [GÓMEZ e PEREZ, 1999]. Sendo a segunda mais completa por aceitar indivíduos como parte da ontologia, logo podemos definir que uma ontologia é composta por:

- Indivíduos são objetos no domínio da ontologia, esses objetos serão classificados dentro da Ontologia. Podemos dizer que "João" é um individuo dentro de uma classe de Pessoas.
- Classes, também conhecidas como conceitos, e uma hierarquia envolvendo essas classes que chamamos de taxonomia. Podemos considerar a classe Pessoa como sendo a classe de todas as pessoas do mundo, e a classe Homem como sendo uma hierarquia da classe Pessoa.
- Podemos descrever relações como atributos onde o valor é outro objeto dentro da ontologia, por exemplo, podemos definir um relacionamento "serMarido" ligando um Homem a uma Mulher.
- Funções são um caso específico de relacionamento onde um conjunto de elementos têm uma relação única com outro elemento.
- Um conjunto de axiomas, sendo axioma uma regra que é sempre verdade, como por exemplo podemos citar o axioma de que toda pessoa tem uma mãe.

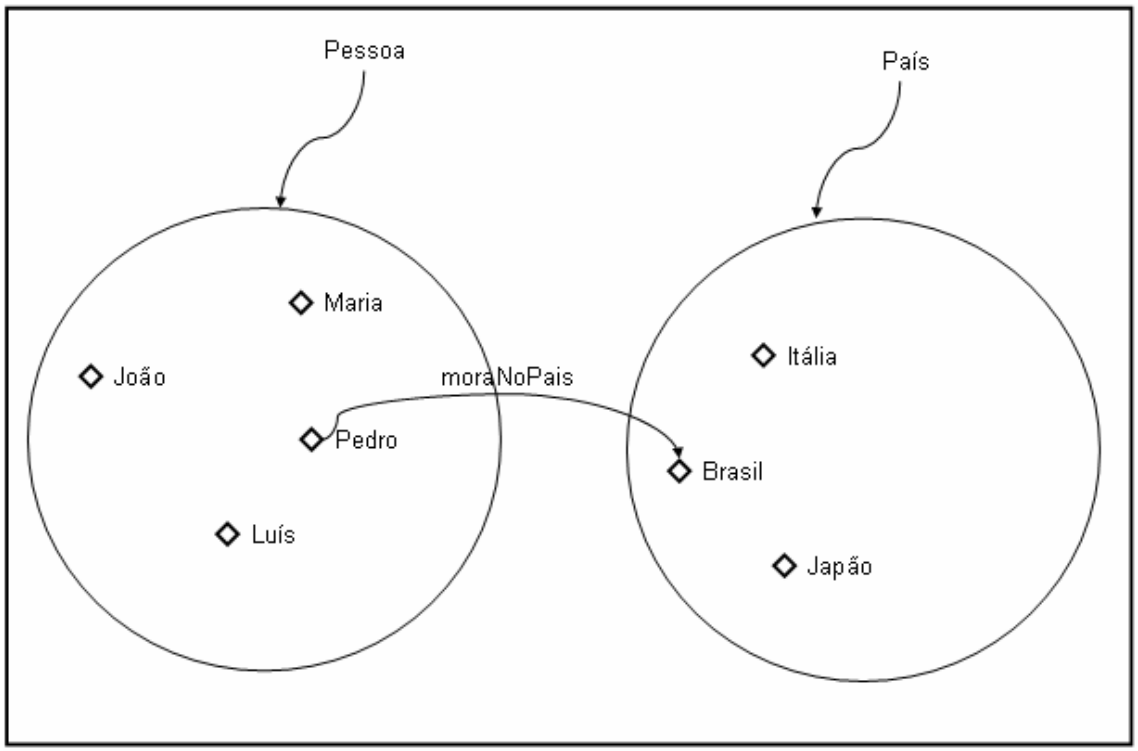


**Ilustração 3.1 - Exemplo de ontologia**

Na figura podemos ver o exemplo de uma ontologia que define as classes Pessoa e País, dentro de cada uma das classes há uma série de indivíduos como Pedro em Pessoa e Brasil na classe País, está também expressa a relação moraNoPais, ligando o individuo Pedro ao individuo Brasil, expressando que Pedro mora no Brasil.

## 3.3 Classificação de Ontologias

Ao longo do tempo foram feitas inúmeras classificações de ontologias, por diferentes autores e levando em consideração diferentes parâmetros e abordagens. Na tabela abaixo temos algumas das mais importantes, assim como a descrição de cada classificação.

**Tabela 3.1 - Tipos de Ontologias [ALMEIDA & BAX, 2003].**

| Abordagem | Classificação | Descrição |
|---|---|---|
| **Quanto à função** [MIZOGUCHI et al, 1995] | Ontologias de Domínio | Reutilizáveis no domínio, fornecem vocabulário sobre conceitos, seus relacionamentos, sobre atividades e regras que os governam. |
| | Ontologias de Tarefa | Fornecem um vocabulário sistematizado de termos, especificando tarefas que podem ou não estar no mesmo domínio. |
| | Ontologias Gerais | Incluem um vocabulário relacionado a coisas, eventos, tempo, espaço, casualidade, comportamento, funções, etc. |
| **Quanto ao grau de formalismo** [USCHOLD & GRUNINGER, 1996] | Ontologias altamente informais | Expressa livremente em linguagem natural. |
| | Ontologias semi-informais | Expressa em linguagem natural de forma restrita e estruturada. |
| | Ontologias semiformais | Expressa em uma linguagem artificial definida formalmente. |
| | Ontologia rigorosamente formal | Os termos são definidos com semântica formal, teoremas e provas. |
| **Quanto à aplicação** [JASPER & USCHOLD, 1999] | Ontologias de autoria neutra | Um aplicativo é escrito em uma única língua e depois convertido para uso em diversos sistemas, reutilizando-se as informações. |
| | Ontologias como especificação | Cria-se uma ontologia para um domínio, a qual é usada para documentação e manutenção no desenvolvimento de softwares. |
| | Ontologias de acesso comum à informação | Quando o vocabulário é inacessível, a ontologia torna a informação inteligível, proporcionando conhecimento compartilhado dos termos. |
| **Quanto à estrutura** [HAAV & LUBI, 2001] | Ontologias de alto nível | Descrevem conceitos gerais relacionados a todos os elementos da ontologia (espaço, tempo, matéria, objeto, evento, ação, etc.) os quais são independentes do problema ou domínio. |
| | Ontologias de domínio | Descrevem o vocabulário relacionado a um domínio, como por exemplo, medicina ou automóveis. |

| | | |
|---|---|---|
| | Ontologias de tarefa | Descrevem uma tarefa ou atividade, como por exemplo, diagnósticos ou compras, mediante inserção de termos especializados na ontologia |
| **Quanto ao conteúdo** [VAN-HEIJIST et al, 1997] | Ontologias terminológicas | Especificam termos que serão usados para representar o conhecimento em um domínio (por exemplo, os léxicos). |
| | Ontologias de informação | Especificam a estrutura de registros de bancos de dados (por exemplo, os esquemas de bancos de dados) |
| | Ontologias de modelagem do conhecimento | Especificam conceitualizações do conhecimento, têm uma estrutura interna semanticamente rica e são refinadas para uso no domínio do conhecimento que descrevem. |
| | Ontologias de aplicação | Contêm as definições necessárias para modelar o conhecimento em uma aplicação. |
| | Ontologias de domínio | Expressam conceitualizações que são específicas para um determinado domínio do conhecimento. |
| | Ontologias genéricas | Similares às ontologias de domínio, mas os conceitos que as definem são considerados genéricos e comuns a vários campos. |
| | Ontologias de representação | Explicam as conceitualizações que estão por trás dos formalismos de representação do conhecimento |

## 3.4 Linguagens para Ontologias

Linguagens formais são usadas para construir ontologias, essas linguagens permitem, ao desenvolvedor, codificar conhecimento sobre um certo domínio, algumas linguagens também possuem regras de raciocínio, permitindo assim que se processe o conhecimento gerado.

Para escolher qual linguagem usar, deve-se primeiro analisar o problema, algumas linguagens são mais expressivas enquanto outras permitem o uso de regras de raciocínio, algumas são mais voltadas pra aplicações web. Cabe ao desenvolvedor, antes de iniciar a construção da ontologia, analisar bem suas necessidades e qual das linguagens disponíveis se adequa melhor.

Podemos mostrar alguns exemplos de linguagens de ontologias na tabela abaixo.

**Tabela 3.2 - Linguagens para Ontologias**

| Tipo | Linguagens |
|---|---|
| **Linguagens tradicionais** | Ontolingua, Cycl, F-Logic, CML, OCML, KIF, Loom |
| **Linguagens padrão para *web*** | XML, RDF |
| **Linguagens *web-based*** | OIL, DAML+OIL, SHOE, XOL, OWL |

### 3.4.1 Lógica Descritiva

A Lógica Descritiva é uma família de linguagens para representação de conhecimento que surgiu como uma evolução das redes semânticas e frames. Seu intuito era de resolver alguns problemas e limitações que estes encontravam, como por exemplo a falta de uma semântica formal, que permitisse a quebra de ambigüidades através de raciocinadores.

Inicialmente, final dos anos setenta, foi chamada de Linguagens Terminológicas, que buscavam representações com mais engajamento ontológico, nessa época surgiu pela primeira vez a modelagem com o uso de TBox e ABox, e foram desenvolvidos alguns sistemas como o KL-ONE [BRACHMAN & SCHMOLZE, 1985] e o KRYPTON [BRACHMAN et al, 1983]. Chegou a se chamar Linguagens Conceituais, chegando até o começo dos anos oitenta quando surgiu a expressão Lógica Descritiva.

Bases de conhecimento, em lógica descritiva, são compostas basicamente por conceitos que representam um conjunto de indivíduos, também chamados de classes, e por propriedades que representam uma relação binária entre dois indivíduos. A partir dai os sistemas permitem aos usuários a construção de descrições complexas sobre conceitos e propriedades. A modelagem em lógica descritiva é feita através das TBox (Terminological Box) e da ABox (Assertional Box) geralmente a TBox traz sentenças que descrevem hierarquias, ou seja, relações entre conceitos, enquanto a ABox traz sentenças representando o individuo dentro da hierarquia, ou seja, relações entre conceitos e indivíduos. Por exemplo, a TBox traria uma expressão do tipo, “todo homem é um ser-humano”, já uma sentença da Abdus seria, “João é um homem”.

Abaixo podemos ver como ficaria uma modelagem de uma base de conhecimento em lógica descritiva, com a base de conhecimento dividida em TBox e ABox fazendo uso de uma linguagem descritiva para a construção das duas e de um raciocinador para inferir relacionamentos e evitar ambigüidades.

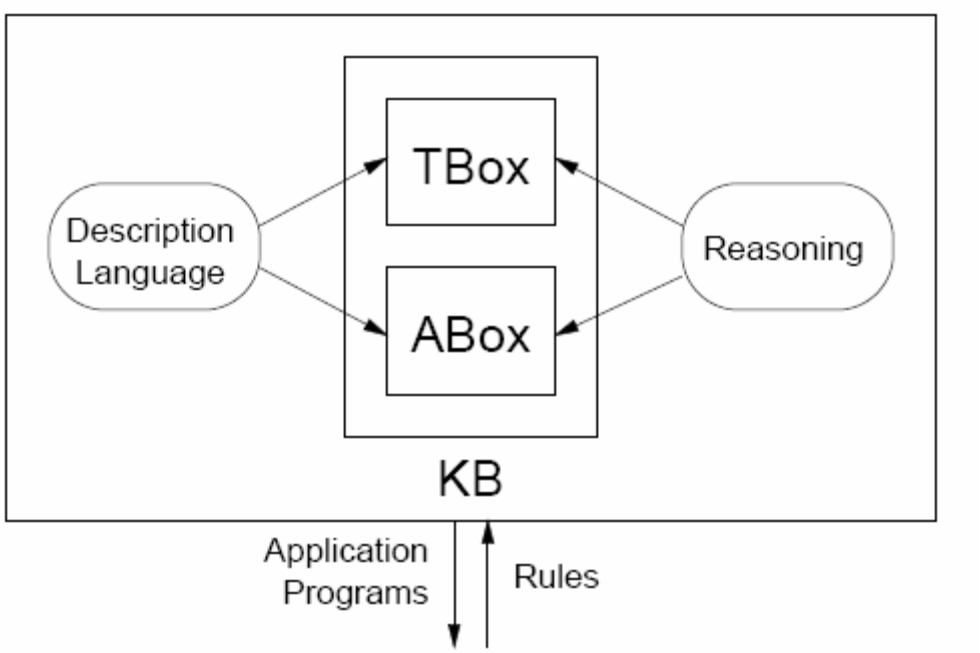


**Ilustração 3.2 - Arquitetura base de conhecimento em Lógica Descritiva [BAADER & NUTT, 2002].**

A Lógica Descritiva traz uma série de linguagens, cada uma abrangendo uma certa quantidade de regras sintáticas, na tabela abaixo podemos ver as linguagens FL-, AL*, H, I e Q. A junção de FL- e AL* forma a linguagem S, mais a frente vamos ver que a linguagem definida para uso neste trabalho é a OWL-DL, que é uma linguagem SHIQ, ou seja traz a união das regras sintáticas de S, H, I e Q.

**Tabela 3.3 - Linguagens da família da Lógica Descritiva.**

| Construct | Syntax | Language |
|---|---|---|
| Concept | A | FL⁻ |
| Role name | R | |
| conjunction | $C \cap D$ | |
| value restriction | $\forall R.C$ | |
| existensial quantification | $\exists R$ | |
| top | $\top$ | AL* |
| bottom | $\bot$ | |
| negation (C) | $\neg A \neg C$ | |
| disjunction (U) | $C \cup D$ | |
| existential restriction (E) | $\exists R.C$ | |
| number restrictions (N) | $(\geq n\ R)\ \ (\leq n\ R)$ | |
| collection of individuals (O) | $\{a_1 \ldots a_n\}$ | |
| Role heirarchy | $R \subseteq S$ | H |
| Inverse role | $R^-$ | I |
| Qualified number restriction | $(\geq n\ R.C)\ \ (\leq n\ R.C)$ | Q |

### 3.4.2 OWL – *Web Ontology Language*

OWL foi desenvolvida pela *World Wide Web Consortium* (W3C) objetivando a definição de ontologias para a *web*, além de bases de conhecimento associadas a elas. Em OWL uma ontologia é definida através de um conjunto de classes, de propriedades e de restrições. Um conjunto de sentenças OWL em um sistema de dedução forma uma base de conhecimento.

A linguagem traz um rico conjunto de operadores como, por exemplo, o *and* e *and negation*. Além disso, traz a possibilidade de trabalhar com máquina de raciocínio para checar consistência e ajudar a manter a hierarquia correta, bem útil para classes com mais de um pai.

OWL pode ser classificado em três sub-lingugens, sendo elas OWL-Lite, OWL-DL e OWL-Full. A diferenciação entre as três pode ser feita principalmente através da sua expressividade, sendo OWL-Lite a menos expressiva, enquanto OWL-Full é a mais expressiva, e a expressividade de OWL-DL fica entre as outras duas. Assim podemos Considerar que OWL-DL é uma extensão de OWL-Lite e que OWL-Full é uma extensão de OWL-DL.

- OWL-Lite – é a sub-linguagem sintaticamente mais simples e deve ser usada quando só se necessita de uma hierarquia de classes simples.
- OWL-DL – bem mais expressiva que OWL-Lite, é baseada em lógica descritiva e permite o uso de máquinas de raciocínio.
- OWL-Full – ainda mais expressiva que OWL-DL, é indicada quando se necessita de muita expressividade, não é possível realizar raciocínio.

# 4. OntoPRIME

Apesar da gerência de riscos estar cada vez mais presente nos projetos de engenharia de software, ainda é comum existirem projetos com uma gerência reativa ou que sequer possuam uma gerência, a falta de padrões e de conhecimento explicito legado, são alguns dos motivos para se negligenciar uma gerência de riscos.

Torna-se necessária um padrão que faça com que seja possível o entendimento da gerência de riscos por todas as partes do projeto, e inclusive por pessoas externas ao projeto, e que também permita a representação do conhecimento adquirido com o tempo.

Nesse contexto é proposta a OntoPRIME (Ontologia de Riscos para Ambientes de Desenvolvimento de Software Multiprojetos).

## 4.1 Relevância

A OntoPRIME tem sua relevância e conseqüente contribuição pautada na construção de uma Ontologia de Riscos para dar suporte ao processo de Gerência de Risco em Ambientes de Desenvolvimento de Software de Múltiplos Projetos e pelo desenvolvimento de um estudo científico sobre os relacionamentos dos fatores de risco de software que influenciam o sucesso dos projetos, em ambientes organizacionais. Fornecendo um vocabulário comum que pode ser utilizado para representar conhecimento útil para os desenvolvedores de software sobre os riscos e oportunidades que podem afetar um projeto de software, vários projetos e mesmo, entre projetos, dentro de uma organização desenvolvedora de software.

A OntoPRIME também fornece uma estrutura que pode ser utilizada para organizar o conhecimento sobre riscos e oportunidades, podendo ser utilizada para orientar a aquisição do conhecimento dos riscos dos projetos.

## 4.2 Construção

A OntoPRIME foi desenvolvida baseando-se na Taxonomia de riscos desenvolvida pelo SEI, que se dividia em três níveis, classes, elementos e atributos, a taxonomia divide os riscos em três superclasses, a de engenharia de produtos que engloba os fatores técnicos associados com a entrega do produto final, ambiente de desenvolvimento que engloba o ambiente do projeto e o processo utilizado e a de restrições de programa cobrindo os fatores externos aos projetos. Abaixo podemos ver uma figura com todas as classes, elementos e atributos da árvore, já com o acréscimo das partes marcadas, que foram sugeridas no desenvolvimento da mPRIME Ontology [GUSMÃO, 2005].

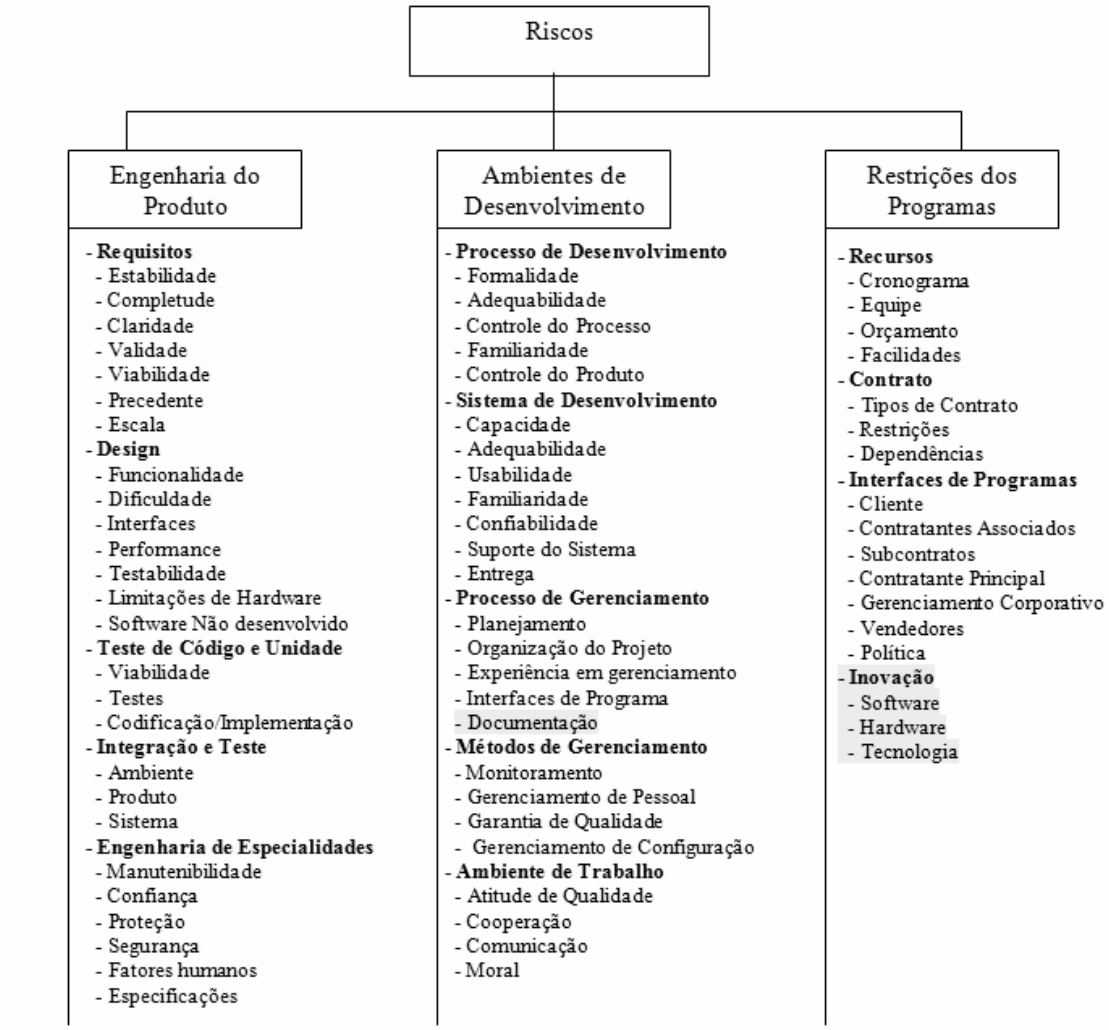


**Ilustração 4.1 - Taxonomia do SEI**

Foram identificados termos e frases através de estudo bibliográfico e análise da Taxonomia de riscos do SEI, daí deu-se início à captura da ontologia que após a captura da semântica foi dividida em sub-ontologias como representado na ilustração 2, cada conceito e restrição foi representado através de linguagem formal.

## 4.3 Aplicações práticas

### 4.3.1 Protótipo – *Ontology Risk Assistant*

Para realizar uma análise da OntoPRIME foi implementado um protótipo, contemplando os axiomas apenas da classe de Engenharia de Produto [GUSMÃO et al, 2004]. O objetivo deste protótipo era demonstrar o uso prático da linguagem proposta pela ontologia.

O protótipo consiste de  um questionário a ser respondido pelo gerente, onde cada reposta revela uma característica do projeto e pode levar à sugestão de riscos. Ao fim é retornada uma representação dos riscos gerados pelas repostas, no formato de uma árvore, obedecendo a classificação da taxonomia do SEI.

### 4.3.2 mPRIME

Uma das formas de avaliação da *Onto*PRIME utilizadas foi a aplicação da mesma em uma ferramenta de Gerência de Riscos em Múltiplos Projetos – *m*PRIME.

A *m*PRIME é uma ferramenta desenvolvida como *add-in* para o Microsoft Office Project 2003 [Lopes e França 2005] que se propõe a apoiar gerentes de projetos e equipes de desenvolvimento nas atividades de Gerência de Riscos. É aderente ao modelo CMMI, contribuindo na qualidade do processo utilizado [SEI 2001].

O processo de Gerência de Riscos utilizado para construção da mPRIME está baseado em modelos e abordagens disponibilizados na literatura de Engenharia de Software e, de uma forma geral, composto por um ciclo dividido em seis fases, como demonstrado na Figura 3.

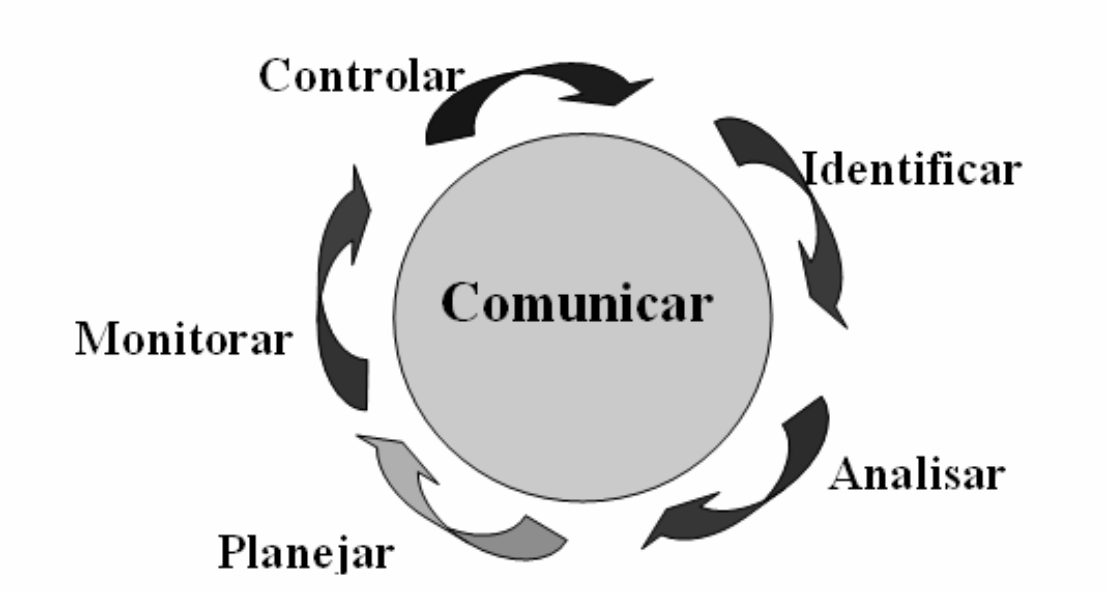


**Ilustração 4.2 - Atividades da Gerência de Riscos**

Todas estas fases estão presentes na ferramenta, e a *Onto*PRIME é usada principalmente na fase de identificação de Riscos. Durante a fase de levantamento de riscos o uso da *Onto*PRIME permite ao programa sugerir riscos para o projeto que podem ser aceitos pelo gerente ou não, estas sugestões podem ser feitas através da análise das tarefas do projeto, da resposta de questionários aderentes ao proposto pelo SEI ou da análise dos riscos já presentes no projeto.

Como dito anteriormente, a atuação da *Onto*PRIME dentro da *m*PRIME tem sua grande importância na área da identificação de riscos, sendo responsável por seis funcionalidades diferentes, apoiadas por três componentes dentro do sistema. Como podemos ver representado no Modelo de Uso da *Onto*PRIME apresentado na Figura 4.

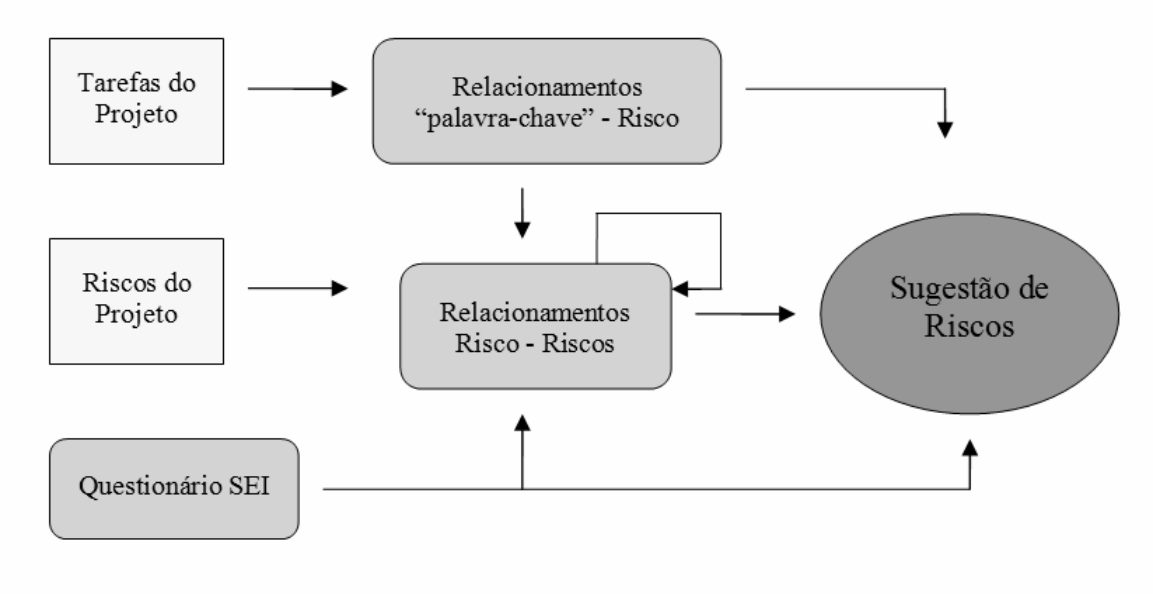


**Ilustração 4.3 - Modelo de uso da OntoPRIME dentro do mPRIME**

#### 4.4.2.1 Componentes

- **Questionário SEI** - Questionário com perguntas relacionadas aos riscos presentes na taxonomia do SEI, as questões são divididas de acordo com as superclasses da taxonomia, e suas repostas geram riscos.
- **Relacionamentos Risco / Riscos** - São ligações, definidas na OntoPRIME, que relacionam um risco, da árvore do SEI, a um conjunto de outros riscos. Ou seja, caso um projeto tenha um risco X, haverá a possibilidade do mesmo também ter um risco Y, sempre lembrando que um risco pode gerar mais de um outro risco ou também nenhum.
- **Relacionamentos "palavra-chave" / Riscos** - São ligações, também definidas na OntoPRIME, que relacionam certas palavras-chaves, que podem estar contidas nas tarefas do projeto, com um conjunto de riscos. Ou seja, caso esse projeto tenha a palavra X entre suas tarefas, haverá a possibilidade do mesmo também ter um risco Y, sempre lembrando que uma palavra pode gerar mais de um outro risco ou também nenhum.

Com o uso desses três componentes foi possível gerar seis formas diferentes de se identificar riscos no projeto, devemos lembrar que esta forma de identificação de riscos

na realidade sugere os riscos ao usuário, ou seja, ao se escolher qualquer uma das opções o resultado será apresentado em forma de uma lista de riscos sugeridos, onde o usuário poderá selecionar aqueles que ele acha convenientes, e descartar os demais.

#### 4.4.2.2 Funcionalidades Geradas

- **Sugestão de riscos pelo questionário SEI** - O usuário pode responder o questionário fornecido pelo SEI, tendo a opção de responder apenas uma parte do questionário, e a partir dessas respostas o sistema irá sugerir os ricos relacionados.
- **Sugestão de riscos pelo questionário SEI, com recursão -** Com esta opção, ao se responder o questionário SEI, os riscos gerados irão passar pelo componente de Relacionamentos Risco – Riscos, podendo gerar uma quantidade maior de riscos.
- **Sugestão de riscos pelas tarefas do projeto** - O usuário pode selecionar esta opção, e o sistema fará uma varredura em todas as palavras presentes na lista de tarefas do projeto, procurando relacioná-las através do componente de Relacionamento "palavra-chave" – Riscos, e sugerir os riscos encontrados.
- **Sugestão de riscos pelas tarefas do projeto, com recursão** - Com esta opção selecionada ao pedir a sugestão de riscos pelas tarefas do projeto, os riscos gerados passarão pelo componente de Relacionamentos Risco – Riscos, podendo gerar uma quantidade maior de riscos.
- **Sugestão de riscos pela lista de riscos do projeto** - Quando o usuário seleciona esta opção, o sistema buscará a lista atual de riscos do projeto, e as passará pelo componente de Relacionamentos Risco – Riscos, gerando assim uma nova lista de risco a serem sugeridos.
- **Sugestão de riscos pela lista de riscos do projeto com recursão -** Com esta opção selecionada, após gerar riscos pela lista de riscos, o sistema irá passar a lista de riscos resultante novamente pelo componente de Relacionamentos Risco – Riscos, gerando um número maior de riscos.

# 5. mPRIME Ontology

Durante o estudo inicial realizado sobre a OntoPRIME, percebeu-se que a maioria dos axiomas gerados na construção da mesma, usavam predicados que não haviam sido definidos, apenas faziam uso de uma nomenclatura intuitiva. Para definição da ontologia no Protégé [PROTEGE], seria necessária a definição desses predicados, o que fez com que o escopo do trabalho aumentasse consideravelmente.

Devido ao grande escopo do trabalho e o curto tempo para realização do mesmo, este trabalho limitou-se a realizar apenas uma primeira fase da redefinição da OntoPRIME, que seria a construção de uma das três sub-ontologias (Engenharia de produto, Ambiente de Desenvolvimento e Restrições de Projeto), neste caso foi escolhida a maior delas, Engenharia de Produto.

Analisando a sub-ontologia de engenharia de produto, percebeu-se que a definição dos predicados formava uma ontologia de projetos de software, levando em conta sua implementação.

Para um melhor entendimento da ontologia, e por fins de padronização, estudou-se a possibilidade do uso de alguma ontologia de projetos de software que já estivesse pronta, mas após pesquisa e busca exaustiva, nenhuma ontologia que satisfizesse as necessidades do trabalho foi encontrada.

Por fim decidiu-se que seria criada uma ontologia própria, para uso no trabalho. Lembrando que por não ser o foco principal do trabalho, a ontologia foi definida sem um grande esmero, e por isso um dos trabalhos futuros proposto é o de redefinição da mesma.

Esta Ontologia está descrita no Apêndice I deste trabalho, e será usada neste capítulo para a construção dos axiomas da mPRIME Ontology.

## 5.1 Sub-Ontologia de Engenharia de Produto.

Quando da construção inicial da OntoPRIME, a ontologia, como visto em 4.3, foi dividida em três sub-ontologias, como podemos ver na figura abaixo.

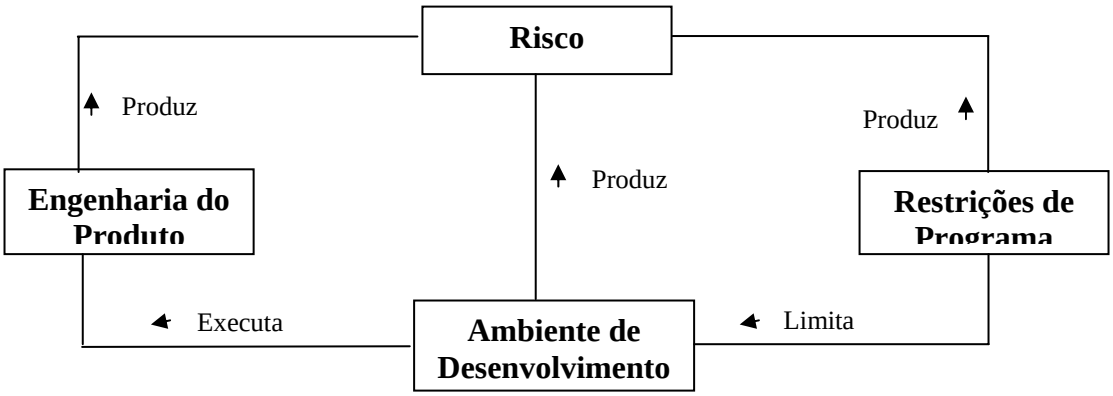


**Ilustração 5.1 - Sub-Ontologias da OntoPRIME [GUSMÂO et al, 2004]**

Nesse trabalho, como, dito acima, vamos focar na redefinição da sub-ontologia de engenharia de produto. Esta classe é composta pelas atividades da Engenharia de Sistema e da Engenharia de Software envolvidas na criação de um sistema que satisfaça requisitos específicos e expectativas do cliente. As atividades incluem:

- Análise e especificação dos requisitos de sistema e de software;
- Modelagem de software e implementação;
- Integração de componentes de hardware e de software; e
- Testes de software e sistema

Inicialmente foram hierarquizadas a classe de Engenharia de Produto e suas subclasses, gerando a classificação de riscos como podemos ver na figura abaixo.


**Ilustração 5.2 - Hierarquia de classes da sub-ontologia de engenharia de produto**

A partir desta classificação, e com o uso da ontologia de projetos descrita no Apêndice I, foram redefinidos os axiomas da OntoPRIME, passando-os para OWL-DL, além de corrigir alguns erros encontrados na mesma.

### 5.1.1 Axiomas

Durante a definição dos axiomas da OntoPRIME, em OWL-DL, todos eles foram revisados e quando necessário corrigidos para prover uma maior concisão da ontologia. Vamos agora mostrar abaixo como foi feita a definição de alguns dos axiomas da OntoPRIME, apesar de todos estarem definidos, para evitar que o trabalho ficasse muito extenso, serão exemplificados apenas alguns, a definição completa dos axiomas se encontra no Apêndice II.

#### 5.1.1.1 Axioma da sub-classe Estabilidade da classe de Risco de Requisitos

**Antiga definição do axioma**

- Se existe algum projeto P no qual o requisito muda durante o processo de desenvolvimento, então existe risco de estabilidade para P.

◦ $\exists p, y.\text{Project}(p) \wedge \text{Requirement}(y, p) \wedge \text{Change}(y) \rightarrow \text{RequirementStabilityRisk}$

Inicialmente foi revisado e corrigido o texto do axioma, que ficou da forma:

- Para todo projeto que possui um requisito que muda durante o processo de desenvolvimento, então existe risco de estabilidade de requisitos para este projeto.

Logo em seguida iniciou-se a definição do axioma em OWL-DL, para isto foi necessária a criação de duas classes dentro da Ontologia de Projetos de Software, essas classes foram: *Project* e *Requirement.*

E também a criação de uma propriedade que mostrasse que os requisitos do projeto estavam mudando, essa propriedade foi: c*hange.* Onde a classe *Project* ficou

sendo o seu domínio, e os artefatos do projeto, incluindo a classe *Requirement*, sendo o seu range.

Agora ficou possível realmente definir o axioma, inicialmente criando uma subclasse da classe *Project*, chamada *ProjectRequirementChanged*, que representava os projetos onde os requisitos mudavam dessa forma:

◦ $ProjectRequirementChanged \equiv \exists \text{change}.Requirement$

E por fim podendo definir a classe *RequirementsStabilityRisk* como contendo, os riscos que fazem parte do projeto que muda, para isso foi necessário criar uma propriedade chamada *isPartOf*, que tem como domínio a classe *Risk* e como range a classe *Project*, logo:

◦ $RequirementsStabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementChanged$

**Definição Final do axioma:**

- Para todo projeto que possui um requisito que muda durante o processo de desenvolvimento, então existe risco de estabilidade de requisitos para este projeto.

◦ $RequirementsStabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementChanged$

◦ $ProjectRequirementChanged \equiv \exists \text{change}.Requirement$

#### 5.1.1.2 Axioma da subclasse Interface da classe de Risco de Design

**Antiga definição do axioma:**

- Se existe algum Projeto P para o qual há um processo para definição de interfaces internas mas não há um processo para controle de mudanças para interfaces internas, então existe risco de interface para P.

◦ $\exists p,x.\,\text{Project}(p) \wedge \exists x.\,\text{ProcessForDefiningInternalInterfaces}(x, p) \wedge \neg\exists y.\,\text{ChangeControlProcessForInternalInterfaces}(y, p) \rightarrow \text{InterfaceRisk}(p)$

Inicialmente foi revisado e corrigido o texto do axioma que ficou da forma:

- Para todo projeto que possui um processo para definição de interfaces internas, mas não há um processo para controle de mudanças para interfaces internas, então existe risco de interface para este projeto.

Logo em seguida iniciou-se a definição do axioma em OWL-DL, para isto foi necessária a criação de duas classes dentro da Ontologia de Projetos de Software, essas classes foram: *ProcessForDefiningInternalInterfaces e ChangeControlProcessFor InternalInterfaces* que são subclasse de *ProjectWorkFlow* definida, também, na Ontologia de Projetos de Software.

E também a criação de uma propriedade que mostrasse se o projeto tinha certo processo, essa propriedade foi: *performs.* Onde a classe *Project* ficou sendo o seu domínio, e os processos do projeto (*ProjectWorkFlows*), incluindo as duas classes definidas acima, sendo o seu range.

Agora ficou possível definir o axioma, inicialmente criando duas subclasses da classe *Project*, chamadas *ProjectPerformsProcessForDefiningInternalInterface*, que representava os projetos que possuem um processo para definir interfaces internas, e *ProjectNotPerformsChangeControlProcessForInternalInterfaces,* que representava o projeto que não possuía processo para controle de mudanças para interfaces internas, dessa forma:

◦ $\textit{ProjectPerformsProcessForDefiningInternalInterfaces} \equiv \exists \text{performs}.\textit{ProcessForDefiningInternalInterfaces}$

◦ $\textit{ProjectNotPerformsChangeControlProcessForInternalInterfaces} \equiv \neg(\exists \text{performs}.\textit{ChangeControlProcessForInternalInterfaces})$

E por fim podendo definir a classe *InterfaceRisk* como contendo, os riscos que fazem parte do projeto que possui processo para definição de interfaces internas, mas não possui processo para controle de mudanças para interfaces internas, usando novamente a propriedade *isPartOf* mostrada no axioma passado, logo:

◦ $\mathit{InterfaceRisk} \sqsupseteq (\forall \text{isPartOf}.\mathit{ProjectPerformsProcessForDefiningInternalInterfaces}) \sqcap (\forall \text{isPartOf}.\mathit{ProjectNotPerformsChangeControlProcessForInternalInterfaces})$

**Definição Final do Axioma:**

- Para todo projeto que possui um processo para definição de interfaces internas, mas não há um processo para controle de mudanças para interfaces internas, então existe risco de interface para este projeto.

◦ $\mathit{InterfaceRisk} \sqsupseteq (\forall \text{isPartOf}.\mathit{ProjectPerformsProcessForDefiningInternalInterfaces}) \sqcap (\forall \text{isPartOf}.\mathit{ProjectNotPerformsChangeControlProcessForInternalInterfaces})$

◦ $\mathit{ProjectPerformsProcessForDefiningInternalInterfaces} \equiv \exists \text{performs}.\mathit{ProcessForDefiningInternalInterfaces}$

◦ $\mathit{ProjectNotPerformsChangeControlProcessForInternalInterfaces} \equiv \neg(\exists \text{performs}.\mathit{ChangeControlProcessForInternalInterfaces})$

### 5.1.1.3 Axioma da subclasse Testes Unitários da classe de Risco de Código e Teste Unitário:

**Antiga Definição do Axioma:**

- Se existe um projeto P no qual existe um módulo implementável para o qual não foram especificados testes unitários, então existe risco de testes unitários.

◦ $\exists p, x.\, \text{Project}(p) \wedge \text{Module}(x, p) \wedge \neg\exists.\, \text{UnitTest}(y, x)) \rightarrow \text{TestingRisk}(p)$

Inicialmente foi revisado e corrigido o texto do axioma que ficou da forma:

- Para todo projeto em que existe um módulo implementável para o qual não foram especificados testes unitários, então existe risco de testes unitários para este projeto.

Logo em sequida iniciou-se a definição do axioma em OWL-DL, para isto foi necessária a criação de duas classes dentro da Ontologia de Projetos de Software, essas classes foram: *Module e UnitTest* que são subclasse de *ProjectWorkFlowArtifacts* definida, também, na Ontologia de Projetos de Software, sendo *module* artefato de implementação e *UnitTet* artefato de teste.

E também a criação de duas propriedades: holds que mostra que certo projeto possui um artefato, sendo *Project* seu domínio e *ProjectWorkFlowArtifacts* o seu range, e *specify,* que mostra se um módulo specifica testes unitários, sendo *Module* seu domínio e *UnitTest* seu range.

Agora ficou possível definir o axioma, inicialmente criando uma subclasse da classe *Module* chamada *ModuleNotSpecifyUnitTest* que representa os módulos que não especificam testes unitários, dessa forma:

$\circ\ ModuleNotSpecifyUnitTest \equiv \neg(\exists \text{specify}.UnitTest)$

E em seguida criando uma subclasse da classe *Project*, chamada *ProjectHoldsModuleNotSpecifyUnitTest,* que representava os projetos que possuem um módulo que não especifica testes unitários, dessa forma:

$\circ\ ProjectHoldsModuleNotSpecifyUnitTest \equiv \exists \text{holds}.ModuleNotSpecifyUnitTest$

E por fim podendo definir a classe *TestingRisk* como contendo, os riscos que fazem parte do projeto que possui um módulo que não especifica testes unitários, usando novamente a propriedade *isPartOf* mostrada anteriormente, logo:

$\circ\ TestingRisk \sqsupseteq \forall \text{isPartOf}.ProjectHoldsModuleNotSpecifyUnitTest$

**Definição Final do Axioma:**

- Para todo projeto em que existe um módulo implementável para o qual não foram especificados testes unitários, então existe risco de testes unitários para este projeto.

◦ $TestingRisk \sqsupseteq \forall \text{isPartOf}.ProjectHoldsModuleNotSpecifyUnitTest$

◦ $ProjectHoldsModuleNotSpecifyUnitTest \equiv \exists \text{holds}.ModuleNotSpecifyUnitTest$

◦ $ModuleNotSpecifyUnitTest \equiv \neg(\exists \text{specify}.UnitTest)$

### 5.1.1.4 Axioma da subclasse Produto da classe de Risco Integração e Testes.

**Antiga Definição do Axioma:**

- Se existe um projeto P no qual a integração de produto não foi especificada suficientemente, então existe risco de integração e teste de produto para P.

◦ $\exists p.\text{Project}(p) \wedge \neg(\exists x.\text{ProductIntegration}(x, p) \wedge \text{Sufficient}(x, p)) \rightarrow \text{ProductIntegrationAndTestRisk}(p)$

Inicialmente foi revisado e corrigido o texto do axioma que ficou da forma:

- Para todo projeto em que a integração de produto não foi especificada suficientemente, então existe risco de integração e teste de produto para este projeto.

Logo em seguida iniciou-se a definição do axioma em OWL-DL, para isto foi necessária a criação de uma classe dentro da Ontologia de Projetos de Software, essa classe foi: *ProductIntegration* que é subclasse de *ProjectWorkFlows* definida, também, na Ontologia de Projetos de Software, mais especificamente é subclasse do *workflow* de integração que faz parte do *workflow* de implementação.

E também o uso da propriedade *specify* que indica se um projeto especifica algum *workflow,* sendo seu domínio a classe *Project* e o seu range a classe *ProjectWorkFlows.*

Agora ficou possível definir o axioma, criando uma subclasse da classe *Project* chamada *ProjectProductIntegrationNotSpecified,* que representa os projetos que não especificam a integração de produto, dessa forma:

◦ $ProjectProductIntegrationNotSpecified \equiv \neg(\exists \text{specify}.ProductIntegrationWorkFlow)$

E por fim podendo definir a classe *ProductIntegrationAndTestRisk* como contendo, os riscos que fazem parte do projeto não especifica integração de produto, usando novamente a propriedade *isPartOf* mostrada anteriormente, logo:

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectProductIntegrationNotSpecified$

**Definição Final do Axioma:**

- Para todo projeto em que a integração de produto não foi especificada suficientemente, então existe risco de integração e teste de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectProductIntegrationNotSpecified$

◦ $ProjectProductIntegrationNotSpecified \equiv \neg(\exists \text{specify}.\text{Pr}oductIntegrationWorkFlow)$

### 5.1.1.5 Axioma da subclasse Proteção da classe de Risco de Especialidades de Engenharia.

**Antiga Definição do Axioma:**

- Se existe algum projeto P no qual requisito de proteção não foi alocado, então existe risco de proteção para P.

◦ $\exists p, y.\text{Project}(p) \wedge \text{SafetyRequirement}(y, p) \wedge \neg \text{Allocated}(y) \rightarrow \text{SafetyRisk}(p)$

Inicialmente foi revisado e corrigido o texto do axioma que ficou da forma:

- Para todo projeto em que requisito de proteção não foi alocado, então existe risco de proteção para este projeto.

Logo em seguida iniciou-se a definição do axioma em OWL-DL, para isto foi necessária a criação de uma classe dentro da Ontologia de Projetos de Software, essa

classe foi: *SafetyRequirement* que é subclasse de *ProjectWorkFlowArtifacts* também definida na Ontologia de Projetos de Software, mais especificamente é subclasse de *Requirement* .

E também o uso da propriedade *allocated* que indica se um projeto aloca algum requisito, sendo *Project* seu domínio e *Requirement* seu range.

Agora ficou possível definir o axioma, criando uma subclasse da classe *Project* chamada *ProjectSafetyRequirementNotAllocated*, que representa os projetos que não alocam requisito de proteção, dessa forma:

◦ $ProjectSafetyRequirementNotAllocated \equiv \neg(\exists \text{allocated}.SafetyRequirement)$

E por fim podendo definir a classe *SafetyRisk* como contendo, os riscos que fazem parte do projeto não aloca requisitos de proteção, usando novamente a propriedade *isPartOf* mostrada anteriormente, logo:

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectSafetyRequirementNotAllocated$

**Definição Final do Axioma:**

- Para todo projeto em que requisito de proteção não foi alocado, então existe risco de proteção para este projeto.

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectSafetyRequirementNotAllocated$

◦ $ProjectSafetyRequirementNotAllocated \equiv \neg(\exists \text{allocated}.SafetyRequirement)$

# 6. Conclusões

Dentro dos objetivos desse trabalho, que eram de redefinir a OntoPRIME, criando a mPRIME Ontology, sendo esta construída em Linguagem Descritiva enquanto aquela usava Lógica de Primeira Ordem, e em seguida a implementação da ontologia na ferramenta Protégé, como uma forma de validação da ontologia e também para facilitar futuras modificações, podemos tirar algumas conclusões:

O trabalho encontrou dificuldades durante seu desenvolvimento que não eram esperadas, principalmente o fato de os predicados usados nos axiomas da ontoprime não estarem definidos, este problema acabou gerando a necessidade da criação de uma ontologia auxiliar de projetos de software, vide Apêndice I, o que gerou um grande trabalho extra, acarretando uma diminuição do escopo, o trabalho que antes englobava toda a OntoPRIME passou a contemplar apenas a sub-ontologia de Engenharia de Produto.

Por fim o trabalho que pretendia fazer uma nova modelagem dos antigos axiomas, precisou redefinir todos os axiomas, fazendo correções quando necessário, e adaptando à nova linguagem. Embora esses imprevistos tenham causado muitas dificuldades, com certeza eles foram fundamentais para um entendimento melhor de todo o escopo do projeto, desde um estudo mais profundo de Lógica Descritiva como foi necessário um estudo bem mais profundo de ontologias e da área de engenharia de software.

## 6.1 Trabalhos Futuros

Trabalhos como esse não podem ser feitos com o intuito de se completarem por eles mesmos, e sim de serem continuados e evoluídos por outras pessoas, ou até pelo próprio autor, no futuro. Dessa forma deixamos aqui algumas propostas de trabalhos futuros, que se realizadas serão de grande proveito para a melhoria da ontologia.

Completar a construção da OntoPRIME em Lógica Descritiva, através da redefinição das sub-ontologias restantes, de Ambiente de Desenvolvimento e de Restrições de Programa.

Realização de uma análise minuciosa na Ontologia de Engenharia de Software descrita no Apêndice I, com o intuito de defini-la de forma mais consistente e completa.

# 7. Referências Bibliográficas

[ALMEIDA & BAX, 2003] ALMEIDA, M e BAX, M. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção, 2003.

[BAADER & NUTT, 2002] Franz Baader and Werner Nutt, "Basic Description Logics" in Baader (Ed.) et al "Description Logics Handbook", Cambridge, 2002.

[BOEHM 1991] Boehm, B. W. (1991) Software Risk Management: Principles and Practices, IEEE Software, Volume 8. No1. pp 32-40.

[BRACHMAN et al, 1983] Brachman, R.; Fikes, R.; and Levesque, H. 1983. KRYPTON: A functional approach to knowledge representation. *IEEE Computer* 16(10):67{73.

[BRACHMAN & SCHMOLZE, 1985] R. J. Brachman and J. G. Schmolze, An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2): 171-216, 1985.

[CHARETTE, 1990] CHARETTE, R. *Application Strategies for Risk Analysis*. New York: MultiScience Press. pp 17-21. 1990.

[CHAPMAN & WARD, 1997] Chapman, C. e Ward, S. (1997) Project Risk Management: Processes, Techniques and Insights. John Wiley & Sons. p 30-41.

[CSEP 1995] Computational Science Education Project (1995) Introduction to Monte Carlo Methods. USA.

[DEMARCO, 2003] De MARCO, T e LISTER, T. *Waltzing with Bears*. New York: Dorset House Publishing. 2003.

[FAIRLEY, 1994] Fairley, R., (1994) Risk Management for Software Projects. IEEE Software. pp 57-67.

[FENSEL, 2001] Fensel, D. (2001) "Ontologies: a silver bullet for knowledge management and electronic commerce". Springer, 2001.

[GOMEZ-PEREZ, 1999] Gómez-Pérez, A. and V.R. Benjamins, Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods, in International Joint Conference on Artificial Intelligence(IJCAI-99), Workshop on Ontologies and Problem-Solving Methods (KRR5), V.R. Benjamins, et al., Editors. 1999: Stockolm, Sweden.

[GUSMÃO et al, 2004] Campello, A. ; Gusmão, C. ; Amorim, L. ; Guedes, M. ; Monteiro, M. ;OntoPRIME: Ontologia de Riscos para Ambientes de Desenvolvimento de Software Multiprojetos, Universidade Federal de Pernambuco, Recife, Brasil.(2004)

[GUSMÃO, 2005] Gusmão, C.M.G. et al. (2005) "Ontologia de Domínio de Riscos", In Suppera Solutions Relatório Técnico, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brasil.

[GUSMÃO, 2007] Gusmão, C (2007) Um Modelo de Processo de Gestão de Riscos para Ambientes deMúltiplos Projetos de Desenvolvimento de Software. Tese de Doutorado. Universidade Federal de Pernambuco. Recife – PE, Brasil.

[HAAV & LUBI, 2001] HAAV, H. M.; LUBI, T. L. A survey of concept-based information retrieval tools on the web. *In*: PROCEEDINGS OF EAST-EUROPEAN CONFERENCE ADBIS. 5. 2001.

[HALL, 1998] HALL, E. M. *Managing Risk – Methods for Software Systems Development*. Addison-Wesley. pp 88-103. 1998.

[HELDMAN 2005] Heldman, K. (2005) Project Manager's Spotlight on Risk Managament. Harbor Light Press. Sybex Inc. San Francisco. USA.

[HIGUERA, 1994] Higuera, P.R. (1994) An Introduction to Team Risk Management, Technical Report. Software Engineering Institute, Carnegie Mellon University. USA.

[JASPER & USCHOLD, 1999] JASPER, R.; USCHOLD, M. A framework for understanding and classifying ontology applications. IJCAI-99, *ONTOLOGY WORKSHOP,1999*, Stockholm, [S. l. : s. n.], 1999.

[KONTIO, 2001] Kontio, J. (2001) Software Engineering Risk Management: A Method, Improvement Framework and Empirical Evaluation. Tese de Doutorado. Universidade de Tecnologia de Helsinki. Finlândia.

[MACHADO, 2002] Machado, C. A. F. (2002) A-Risk: Um método para identificar e quantificar risco de prazo em projetos de desenvolvimento de software. Dissertação de Mestrado. Pontifícia Universidade Católica do Paraná. Curitiba - PR, Brasil.

[MAEDCHE, 2001] Maedche, A. And Staab, S. Onology learning for the semantic web, IEEE Inteligent Systems, march-april 2001.

[MARTINO, 2003] MARTINO, Joseph P. Technological forecasting for decision making. 3. ed. New York: Mc Graw-Hill Inc., 1993.

[MIZOGUCHI et al, 1995] MIZOGUCHI, R.; VANWELKENHUYSEN, J.; IKEDA, M. Task ontology for reuse of problem solving knowledge. *In*: PROCEEDINGS OF ECAI'94 TOWARDS VERY LARGE KNOWLEDGE BASES, 1994, Amsterdam. [S. l.] : IOS Press, 1995, p. 46-59.

[MOYNIHAM, 1997] Moynihan, T. (1997) How experienced Project Managers Access Risk. IEEE Software. Volume 14. Nº 3. 35-41.

[PMI, 2004] PMI - Project Management Institute. (2004) A Guide to the Project Management Body of Knowledge. – ANSI/PMI 99-01-2004. Project Management Institute. Four Campus Boulevard. Newtown Square. USA.

[PMI, 2006] PMI - Project Management Institute. (2006) Best approach to identify risks Results. Disponível na URL: <http://www.pmi.org>. Acesso em: 28.08.2006.

[PRESSMAN, 2006] Pressman, R. S. (2006) Engenharia de Software. 6ª edição. São Paulo:McGraw-Hill. Pp 577-595.

[PROTEGE] http://protege.stanford.edu.

[ROBIN et al, 2002] Robin, A.; Preedy, D.; Campbell, D.; Paschino, E. and Hargrave, L. (2002) Microsoft Solutions Framwork. MSF Risk Management Discipline v.1.1. White Paper. Microsoft Solutions Corporation. USA.

[SEI, 2001] SEI - Software Engineering Institute. (2001) CMMI - Capability Maturity Model Integration version 1.1 Pittsburgh, PA. Software Engineering Institute, Carnegie Mellon University. USA.

[USCHOLD & GRUNINGER, 1996] USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods an applications. *Knowledge Engineering Review*, v. 11, n. 2, 1996.

[VAN-HEIJIST et al, 1997] VAN HEIJIST, G.; SCHREIBER, A. T.; WIELINGA, B. J. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, v. 46, n. 2-3, p. 183-192, feb./march 1997.

[VICTORIA et al, 2000] Victoria, C. G. et al. (2000) Pesquisa Qualitativa em Saúde: uma introdução ao tema. Porto Alegre: Tomo Editorial. pp. 33 – 78.

Carr, M. et al (1993) “Taxonomy Based Risk Identification. Technical report CMU/SEI-93-TR-6”, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

Apêndice I

# Ontologia Auxiliar de Projetos de Software

A ontologia foi definida baseando-se nos predicados usados nos axiomas da sub-ontologia de Engenharia de Produto da OntoPRIME, e classificados também de acordo com os axiomas, mas também levando em conta uma análise dos fluxos de trabalho básicos (*Analysis and Design, Business Modeling, Deployment, Implementation, Requirements* e *Test*) do RUP (Rational Unified Process) assim como dos artefatos gerados em cada um desses fluxos.

Na figura abaixo podemos ver uma classificação superficial dos relacionamento entre as classes.

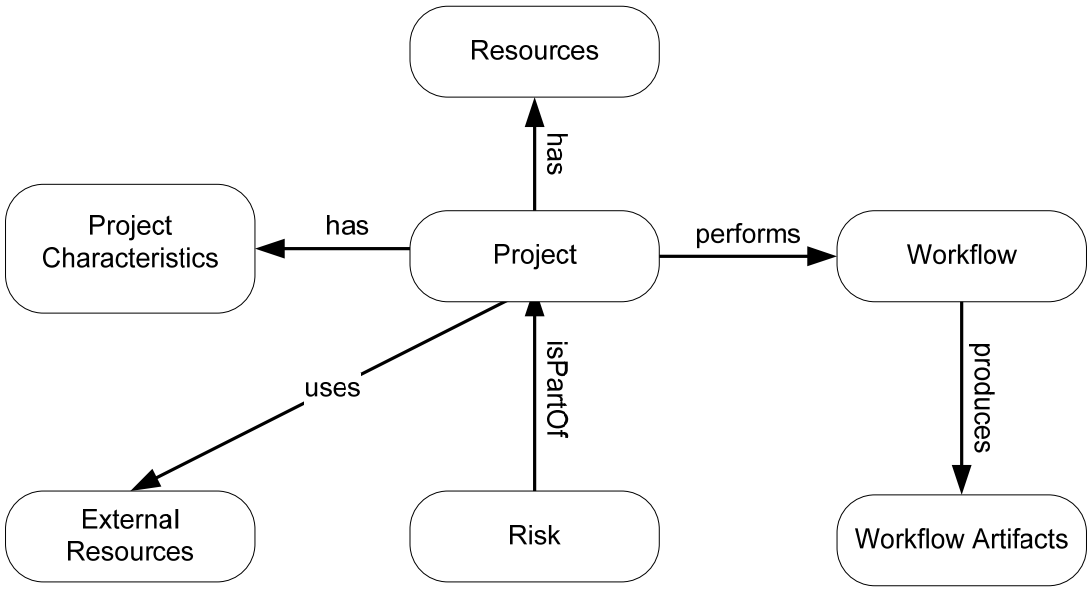


**Ilustração I.1 - Modelo Geral da Ontologia de Projetos de Software**

Já nessa figura podemos ver a classificação das subclasses de *Worflow*, de acordo com o RUP.

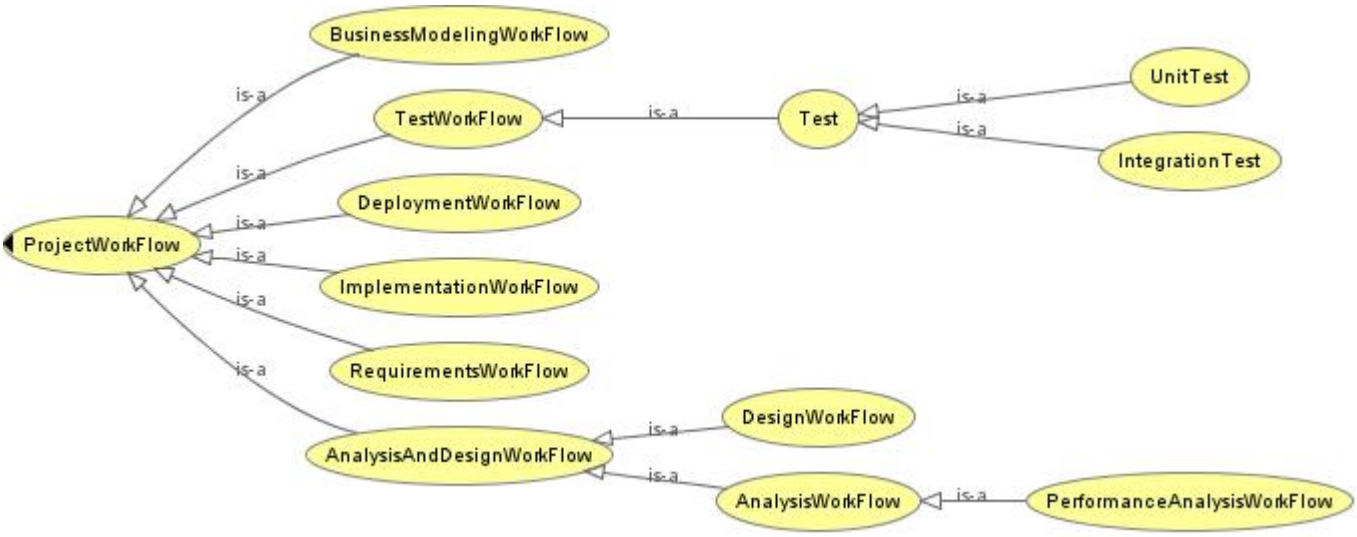


**Ilustração I.2 - Classificação da classe Workflow**

Nessa figura podemos ver a classificação das subclasses de *WorkFlowArtifacts*, que são os artefatos gerados durante cada *WorkFlow* do processo.


**Ilustração I.3 - Classificação da classe WorkFlowArtifacts**

Nessa figura podemos ver a classificação das características que um projeto pode ter, essas características são as que foram usadas nos axiomas da OntoPRIME.

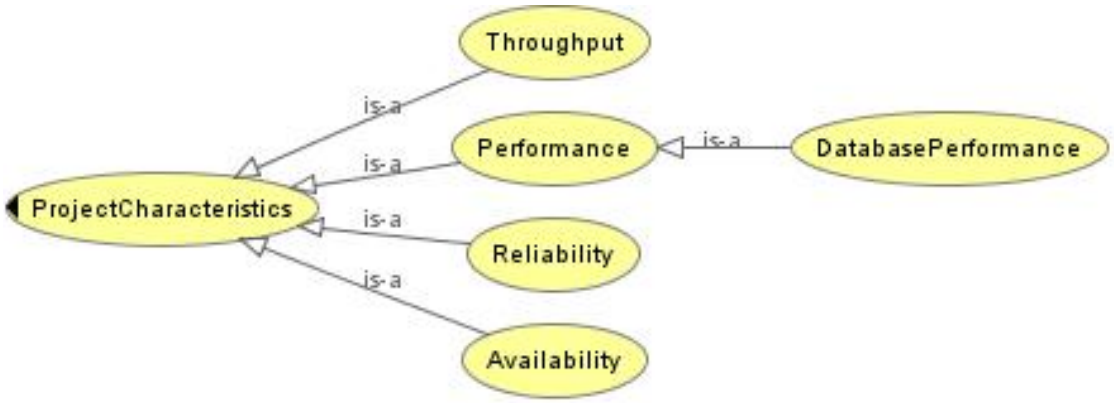


**Ilustração I.4 - Classificação da classe ProjectCharacteristcs**

Nessa figura podemos ver a classificação dos resources do projeto, também de acordo com os axiomas da OntoPRIME.

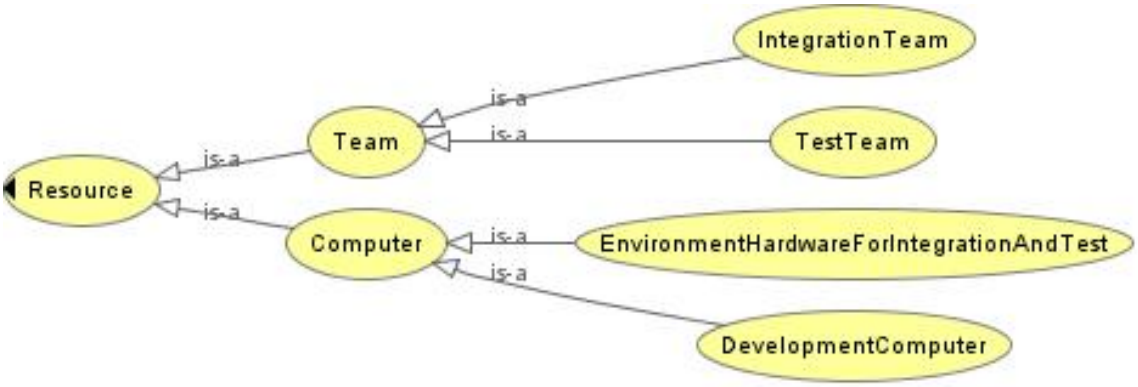


**Ilustração I.5 - Classificação da classe Resource**

Nessa figura podemos ver a classificação dos resource externos, também usados na OntoPRIME.

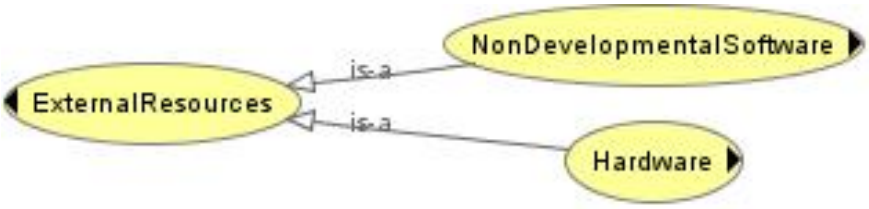


**Ilustração I.6 - Classificação da classe External Resources**

Nessa tabela podemos ver todas as propriedades geradas, assim como seu *range* e *domain.* Essas propriedades foram criadas de acordo com a necessidade durante a definição da mPRIME Ontology.

**Tabela I.1- Lista de propriedades da Ontologia de Projeto de Software**

| Property | Domain | Range |
|---|---|---|
| adequate | Project | ProjectWorkflowArtifacts |
| allocated | Project | Requirement |
| beingDeveloped | Project | Hardware |
| change | Project | ProjectWorkflowArtifacts |
| clear | Project | Requirement |
| companyExperience | Project | Product |
| complex | Project | Product |
| define | Project | Requirement<br>Interface |
| demonstrable | Project | Requirement |
| difficultMaintain | Project | ProjectWorkflowArtifacts |
| difficultUnderstand | Project | ProjectWorkFlow |
| easyDesign | Project | Requirement |
| easyImplementation | Project | ImplementationArtifacts |
| feasible | Project | Requirement |
| increases | ExternalResources | ResponseTime<br>RecoveryTime |
| isPartOf | Risk | Project |
| large | Project | Product |
| limits | ExternalResources | ProjectWorkFlowArtifacts<br>ProjectCharacteristcs |
| needsInterpretation | Project | Requirement |
| neverDone | Project | Requirement |
| performs | Project | ProjectWorkFlow |
| precedent | Project | Requirement |
| representCostumersNeed | Project | Requirement |
| scheduled | Project | AsynchronousRealTimeEvent |
| solutionExsists | Project | Requirement |
| specifiedNeverDone | Project | Requirement |
| specify | Project | Requirement<br>Module |
| technicalDifficult | Project | Requirement |
| understood | Project | Requirement |
| uses | Project | ExternalResources |

Apêndice II

# Definição dos Axiomas da mPRIME Ontology

## II.1 Axiomas da classe Risco de Requisitos

Para a classe Risco de Requisitos e suas subclasses temos os seguintes axiomas e relacionamentos:

**Estabilidade:**

- Para todo projeto que possui um requisito que muda durante o processo de desenvolvimento, então existe risco de estabilidade de requisitos para este projeto.

◦ *RequirementsStabilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementChanged*
◦ *ProjectRequirementChanged* $\equiv \exists$ change.*Requirement*

- Para todo projeto que possui um interface externa e esta interface muda então existe risco de estabilidade de requisitos para este projeto.

◦ *RequirementsStabilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectExternalInterfaceChanged*
◦ *ProjectExternalInterfaceChanged* $\equiv \exists$ change.*ExternalInterface*

**Completude:**

- Para todo projeto que possui um requisito e esse requisito não está completamente definido, então existe risco de completude de requisitos para este projeto.

◦ *RequirementsCompletenessRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNotDefined*
◦ *ProjectRequirementNotDefined* $\equiv \neg(\exists$ define.*Requirement*$)$

- Para todo projeto que possui um requisito que não está completamente especificado, então existe risco de completude de requisitos para este projeto.

◦ *RequirementsCompletenessRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNotSpecified*
◦ *ProjectRequirementNotSpecified* $\equiv \neg(\exists$ specify .*Requirement*$)$

- Para todo projeto que possui uma interface externa que não está completamente definida, então existe risco de completude de requisitos para este projeto.

◦ *RequirementsCompletenessRisk* $\sqsupseteq \forall$ isPartOf .*ProjectExternalInterfaceNotDefined*
◦ *ProjectExternalInterfaceNotDefined* $\equiv \neg(\exists$ define .*ExternalInterface*$)$

**Clareza:**

- Para todo projeto que possui requisito e este requisito não está claro, então existe risco de clareza para este projeto.

◦ *RequirementsClarityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNotClearly*
◦ *ProjectRequirementNotClearly* $\equiv \neg(\exists$ clear .*Requirement*$)$

- Para todo projeto que possui um requisito que precisa de interpretação, então existe risco de clareza de requisitos para este projeto.

◦ *RequirementsClarityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNeedsInterpretation*
◦ *ProjectRequirementNeedsInterpretation* $\equiv \exists$ needsInterpretation .*Requirement*

**Validade:**

- Para todo projeto que possui um requisito que não representa a necessidade do usuário, então existe risco de validade de requisitos para este projeto.

◦ *RequirementsValidityRisk* $\sqsupseteq$
$\forall$ isPartOf .*ProjectRequirementNotRepresentCostumersNeed*
◦ *ProjectRequirementNotRepresentCostumersNeed* $\equiv$
$\neg(\exists$ representCostumersNeed.*Requirement*$)$

- Para todo projeto que possui um requisito que não tem o mesmo entendimento para o cliente e desenvolvedor, então existe risco de validade para este projeto.

◦ *RequirementsValidityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNotUnderstood*
◦ *ProjectRequirementNotUnderstood* $\equiv \neg(\exists$ understood.*Requirement*$)$

**Viabilidade:**

- Para todo projeto que possui um requisito inviável sob o ponto de vista da análise, então existe risco de viabilidade de requisitos para este projeto.

◦ *RequirementsFeasibilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNotFeasible*
◦ *ProjectRequirementNotFeasible* $\equiv \neg(\exists$ feasible.*Requirement*$)$

- Para todo projeto que possui um requisito tecnicamente dificil de implementar, então existe risco de viabilidade de requisitos para este projeto.

◦ *RequirementsFeasibilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementTechnicalDifficult*
◦ *ProjectRequirementTechnicalDifficult* $\equiv \exists$ techinicalDifficult.*Requirement*

**Precedente:**

- Para todo projeto que possui um requisito que nunca foi desenvolvido anteriormente, então existe risco de precedente para este projeto.

◦ *RequirementsPrecedentRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementNeverDone*
◦ *ProjectRequirementNeverDone* $\equiv \exists$ neverDone .*Requirement*

- Para todo projeto que possui um requisito que especifica algo que nunca foi realizado anteriormente, então existe risco de precedente para este projeto.

◦ *RequirementsPrecedentRisk* $\sqsupseteq \forall$ isPartOf .*ProjectRequirementSpecifiedNeverDone*
◦ *ProjectRequirementSpecifiedNeverDone* $\equiv \exists$ specifiedNeverDone .*Requirement*

**Escala:**

- Para todo projeto para o qual será desenvolvido um produto complexo, então existe risco de escala para este projeto.

◦ *RequirementsScaleRisk* $\sqsupseteq \forall$ isPartOf .*ProjectProductComplex*
◦ *ProjectProductComplex* $\equiv \exists$ complex .*Product*

- Para todo projeto para o qual será desenvolvido um produto grande, então existe risco de escala para este projeto.

◦ *RequirementsScaleRisk* $\sqsupseteq \forall$ isPartOf .*ProjectProductLarge*
◦ *ProjectProductLarge* $\equiv \exists$ large .*Product*

- Para todo projeto para o qual será desenvolvido um produto que a organização não tem experiência, então existe risco de escala para este projeto.

◦ *RequirementsScaleRisk* $\sqsupseteq \forall$ isPartOf .*ProjectProductNotCompanyExperience*
◦ *ProjectProductNotCompanyExperience* $\equiv \neg(\exists$ companyExperience .*Product*$)$

**II.2 Axiomas da classe Risco de Design**

Para a classe Risco de Design, e suas subclasses, foram definidos os seguintes relacionamentos e axiomas:

**Funcionalidade:**

- Para todo projeto em que existe um algoritmo que não satisfaz o requisito para o qual foi projetado, então existe risco de funcionalidade para este projeto.

◦ $\mathit{FunctionalityRisk} \sqsupseteq \forall \text{isPartOf}.\mathit{ProjectHoldsAlgorithmNotSatisfyRequirement}$
◦ $\mathit{ProjectHoldsAlgorithmNotSatisfyRequirement} \equiv \exists \text{holds}.\mathit{AlgorithmNotSatisfyRequirement}$
◦ $\mathit{AlgorithmNotSatisfyRequirement} \equiv \neg(\exists \text{satisfy}.\mathit{Requirement})$

**Dificuldade:**

- Para todo projeto no qual exista um requisito cujo design não é fácil, então existe risco de dificuldade para este projeto.

◦ $\mathit{DificultyRisk} \sqsupseteq \forall \text{isPartOf}.\mathit{ProjectRequirementNotEasyDesign}$
◦ $\mathit{ProjectRequirementNotEasyDesign} \equiv \neg(\exists \text{easyDesign}.\mathit{Requirement})$

- Para todo projetono qual existe um requisito que não tem solução, então existe risco de dificuldade para este projeto.

◦ $\mathit{DificultyRisk} \sqsupseteq \forall \text{isPartOf}.\mathit{ProjectRequirementNotSolutionExsists}$
◦ $\mathit{ProjectRequirementNotSolutionExsists} \equiv \neg(\exists \text{solutionExsists}.\mathit{Requirement})$

**Interfaces:**

- Para todo projeto que possui uma interface interna que não está completamente definida, então existe risco de interface para este projeto.

◦ $InterfaceRisk \sqsupseteq \forall \text{isPartOf}.ProjectInternalInterfaceNotDefined$

◦ $ProjectInternalInterfaceNotDefined \equiv \neg(\exists \text{define}.InternalInterface)$

- Para todo projeto que não possui um processo para definição de interfaces internas, então existe risco de interface para este projeto.

◦ $InterfaceRisk \sqsupseteq \forall \text{isPartOf}.ProjectNotPerformsProcessForDefiningInternalInterfaces$

◦ $ProjectNotPerformsProcessForDefiningInternalInterfaces \equiv \neg(\exists \text{performs}.ProcessForDefiningInternalInterfaces)$

- Para todo projeto que possui um processo para definição de interfaces internas, mas não há um processo para controle de mudanças para interfaces internas, então existe risco de interface para este projeto.

◦ $InterfaceRisk \sqsupseteq (\forall \text{isPartOf}.ProjectPerformsProcessForDefiningInternalInterfaces) \sqcap (\forall \text{isPartOf}.ProjectNotPerformsChangeControlProcessForInternalInterfaces)$

◦ $ProjectPerformsProcessForDefiningInternalInterfaces \equiv \exists \text{performs}.ProcessForDefiningInternalInterfaces$

◦ $ProjectNotPerformsChangeControlProcessForInternalInterfaces \equiv \neg(\exists \text{performs}.ChangeControlProcessForInternalInterfaces)$

- Para todo projeto que possui hardware sendo desenvolvido em paralelo e as especificações desse hardware estão mudando, então existe risco de interface para este projeto.

◦ $InterfaceRisk \sqsupseteq (\forall \text{isPartOf}.ProjectHardwareBeingDeveloped) \sqcap (\forall \text{isPartOf}.ProjectHardwareSpecificationChanged)$

◦ $ProjectHardwareBeingDeveloped \equiv \exists \text{beingDeveloped}.Hardware$

◦ $ProjectHardwareSpecificationChanged \equiv \exists \text{change}.HardwareSpecification$

- Para todo projeto que possui um hardware sendo desenvolvido em paralelo e a interface com sioftware desse hardware não está completamente definida, então existe risco de interface para este projeto.

◦ $\mathit{InterfaceRisk} \sqsupseteq (\forall \text{isPartOf}.\mathit{ProjectHardwareBeingDeveloped}) \sqcap (\text{isPartOf}.\mathit{ProjectHardwareInterfaceNotDefined})$

◦ $\mathit{ProjectHardwareBeingDeveloped} \equiv \exists \text{beingDeveloped}.\mathit{Hardware}$

◦ $\mathit{ProjectHardwareInterfaceNotDefined} \equiv \neg(\exists \text{define}.\mathit{HardwareInterface})$

**Performance:**

- Para todo projeto que possui um evento de tempo real assíncrono que não está sendo escalonado, então existe risco de performance para este projeto.

◦ $\mathit{PerformanceRisk} \sqsupseteq \forall \text{isPartOf}.\mathit{ProjectAssynchronousRealTimeEventNotScheduled}$

◦ $\mathit{ProjectAssynchronousRealTimeEventNotScheduled} \equiv \neg(\exists \text{scheduled}.\mathit{AssynchronousRealTimeEvent})$

- Para todo projeto em que não foi realizada uma análise de performance, então existe risco de performance para este projeto.

◦ $\mathit{PerformanceRisk} \sqsupseteq \forall \text{isPartOf}.\mathit{ProjectPerformanceAnalysisNotPerformed}$

◦ $\mathit{ProjectPerformanceAnalysisNotPerformed} \equiv \neg(\exists \text{performs}.\mathit{PerformanceAnalysisWorkFlow})$

- Para todo projeto para o qual foi realizada uma análise de performance , porém não existe um modelo de acompanhamento de performance, então existe risco de performance para este projeto.

◦ $PerformanceRisk \sqsupseteq (\forall \text{isPartOf}.ProjectPerformanceAnalysisNotPerformed) \sqcap (\forall \text{isPartOf}.ProjectNotHasModelToTrackPerformance)$

◦ $ProjectPerformanceAnalysisNotPerformed \equiv \neg(\exists \text{performs}.PerformanceAnalysis)$

◦ $ProjectNotHasModelToTrackPerformance \equiv \neg(\exists \text{has}.ModelToTrackPerformance)$

**Testabilidade:**

- Para todo projeto que não possui um processo de testes fácil de executar, então existe risco de testabilidade para este processo.

◦ $TestabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectNotEasyPerformsTestWorkFlow$

◦ $ProjectNotEasyPerformsTestWorkFlow \equiv \neg(\exists \text{easyPerforms}.TestWorkFlow)$

- Para todo projeto que não possui um elemento que auxilie os testes, então existe risco de testabilidade para esse projeto.

◦ $TestabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectNotAidsTestWorkFlow$

◦ $ProjectNotAidsTestWorkFlow \equiv \neg(\exists \text{aids}.TestWorkFlow)$

**Limitação de Hardware:**

- Para todo projeto que utiliza um hardware que limita a arquitetura, então existe risco de limitação de hardware para este projeto.

◦ $HardwareConstraintsRisk \sqsupseteq \forall \text{isPartOf}.ProjectUsesHardwareLimitsArchitecture$

◦ $ProjectUsesHardwareLimitsArchitecture \equiv \exists \text{uses}.HardwareLimitsArchitecture$

◦ $HardwareLimitsArchitecture \equiv \exists \text{limits}.Architecture$

- Para todo projeto que utiliza um hardware que diminui a capacidade de memória desejada, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* ⊒
∀ isPartOf .*ProjectUsesHardwareLimitsDesiredMemoryCapacity*
◦ *ProjectUsesHardwareLimitsDesiredMemoryCapacity* ≡
∃ limits .*DesiredMemoryCapacity*

- Para todo projeto que utiliza um hardware que diminui a vazão do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* ⊒ ∀ isPartOf .*ProjectUsesHardwareLimitsThroughput*
◦ *ProjectUsesHardwareLimitsThroughput* ≡ ∃ uses .*HardwareLimitsThroughput*
◦ *HardwareLimitsThroughput* ≡ ∃ limits .*Throughput*

- Para todo projeto que utiliza um hardware que aumenta o tempo médio de reposta do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* ⊒ ∀ isPartOf .*ProjectUsesHardwareIncreasesResponseTime*
◦ *ProjectUsesHardwareIncreasesResponseTime* ≡
∃ uses .*HardwareIncreasesResponseTime*
◦ *HardwareIncreasesResponseTime* ≡ ∃ increases .*ResponseTime*

- Para todo projeto que utiliza um hardware que aumenta o tempo de recuperação após falha do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* ⊒ ∀ isPartOf .*ProjectUsesHardwareIncreasesRecoveryTime*
◦ *ProjectUsesHardwareIncreasesRecoveryTime* ≡
∃ uses .*HardwareIncreasesRecoveryTime*
◦ *HardwareIncreasesRecoveryTime* ≡ ∃ increases .*RecoveryTime*

- Para todo projeto que utiliza um hardware que diminui a performance do banco de dados do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* $\sqsupseteq$ $\forall$ isPartOf .*ProjectUsesHardwareLimitsDatabasePerformance*
◦ *ProjectUsesHardwareLimitsDatabasePerformance* $\equiv$ $\exists$ uses.*HardwareLimitsDatabasePerformance*
◦ *HardwareLimitsDatabasePerformance* $\equiv \exists$ limits.*DatabasePerformance*

- Para todo projeto que utiliza um hardware que impede a implementação de uma funcionalidade do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* $\sqsupseteq \forall$ isPartOf .*ProjectUsesHardwareLimitsFunctionality*
◦ *ProjectUsesHardwareLimitsFunctionality* $\equiv \exists$ uses.*HardwareLimitsFunctionality*
◦ *HardwareLimitsFunctionality* $\equiv \exists$ limits.*Functionality*

- Para todo projeto que utiliza um hardware que diminui a confiabilidade do mesmo, então existe risco de limitação de hardware para este projeto.

◦ *HardwareConstraintsRisk* $\sqsupseteq \forall$ isPartOf .*ProjectUsesHardwareLimitsReliability*
◦ *ProjectUsesHardwareLimitsReliability* $\equiv \exists$ uses.*HardwareLimitsReliability*
◦ *HardwareLimitsReliability* $\equiv \exists$ limits.*Reliability*

- Para todo Projeto que utiliza um hardware que diminui a disponibilidade do mesmo, então existe risco de restrições de hardware para este projeto.

◦ *HardwareConstraintsRisk* $\sqsupseteq \forall$ isPartOf .*ProjectUsesHardwareLimitsAvailability*
◦ *ProjectUsesHardwareLimitsAvailability* $\equiv \exists$ uses.*HardwareLimitsAvailability*
◦ *HardwareLimitsAvailability* $\equiv \exists$ limits.*Availability*

**Software Externo:**

- Para todo projeto que utiliza um software externo que diminui a performance do mesmo, então existe risco de software externo para este projeto.

◦ $NonDevelopmentalSoftwareRisk \sqsupseteq \forall \text{isPartOf}.ProjectUsesNonDevelopmentalSoftwareLimitsPerformance$

◦ $ProjectUsesNonDevelopmentalSoftwareLimitsPerformance \equiv \exists \text{uses}.NonDevelopmentalSoftwareLimitsPerformance$

◦ $NonDevelopmentalSoftwareLimitsPerformance \equiv \exists \text{limits}.Performance$

- Para todo projeto que utiliza um software não desenvolvido pelo programa que compromete uma funcionalidade do mesmo, então existe risco de software externo para este projeto.

◦ $NonDevelopmentalSoftwareRisk \sqsupseteq \forall \text{isPartOf}.ProjectUsesNonDevelopmentalSoftwareLimitsFunctionality$

◦ $ProjectUsesNonDevelopmentalSoftwareLimitsFunctionality \equiv \exists \text{uses}.NonDevelopmentalSoftwareLimitsFunctionality$

◦ $NonDevelopmentalSoftwareLimitsFunctionality \equiv \exists \text{limits}.Functionality$

### II.3 Axiomas da classe Risco de Código e Teste Unitário

Para a classe de Risco de código e teste unitário e suas subclasses, foram definidos os seguintes relacionamentos e axiomas.

**Viabilidade:**

- Para todo projeto que possui um modulo implementável não especificado durante a fase de design, então existe risco de viabilidade de implementação para este projeto.

◦ $ImplementationFeasibilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectModuleNotSpecified$

◦ $ProjectModuleNotSpecified \equiv \neg(\exists \text{specify}.Module)$

- Para todo projeto que possui um algoritmo cuja implementação não é facil, então existe risco de viabilidade de implementação para este projeto.

◦ $ImplementationFeasibilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectAlgorithmNotEasyImplementation$

◦ $ProjectAlgorithmNotEasyImplementation \equiv \neg(\exists \text{easyImplementation}.Algorithm)$

**Testes Unitários:**

- Para todo projeto em que existe um módulo implementável para o qual não foram especificados testes unitários, então existe risco de testes unitários para este projeto.

◦ $TestingRisk \sqsupseteq \forall \text{isPartOf}.ProjectHoldsModuleNotSpecifyUnitTest$

◦ $ProjectHoldsModuleNotSpecifyUnitTest \equiv \exists \text{holds}.ModuleNotSpecifyUnitTest$

◦ $ModuleNotSpecifyUnitTest \equiv \neg(\exists \text{specify}.UnitTest)$

- Para todo projeto em que existe um módulo implementável para o qual não houve tempo suficiente para a execução de testes unitários, então existe risco de testes unitários para este projeto.

◦ $TestingRisk \sqsupseteq \forall \text{isPartOf}.ProjectHoldsModuleNotPerformsUnitTest$

◦ $ProjectHoldsModuleNotPerformsUnitTest \equiv \exists \text{holds}.ModuleNotPerformsUnitTest$

◦ $ModuleNotPerformsUnitTest \equiv \neg(\exists \text{performs}.UnitTest)$

**Codificação / Implementação:**

- Para todo projeto em que existe um módulo implementável cuja especificação de design não é suficiente para escrever o código, então existe risco de implementação para este projeto.

◦ *ImplementationRisk* $\sqsupseteq$
$\forall$ isPartOf .*ProjectHoldsModuleNotSufficientDesignSpecification*
◦ *ProjectHoldsModuleNotSufficientDesignSpecification* $\equiv$
$\exists$ holds .*ModuleNotSufficientDesignSpecification*
◦ *ModuleNotSufficientDesignSpecification* $\equiv \neg(\exists \text{sufficient}.\textit{DesignSpecification})$

- Para todo projeto em que existe uma restrição de sistema que dificulta a implementação do mesmo, então existe risco de implementação para este projeto.

◦ *ImplementationRisk* $\sqsupseteq$
$\forall$ isPartOf .*ProjectExistsSystemConstraintDifficultsImplementation*
◦ *ProjectExistsSystemConstraintDifficultsImplementation* $\equiv$
$\exists$ exists .*SystemConstraintDifficultsImplementation*
◦ *SystemConstraintDifficultsImplementation* $\equiv \exists$ difficults .*ImplementationWorkFlow*

- Para todo projeto em que a linguagem de programação não é adequadapara sua finalidade, então existe risco de implementação para este projeto.

◦ *ImplementationRisk* $\sqsupseteq \forall$ isPartOf .*ProjectProgrammingLanguageNotAdequate*
◦ *ProjectProgrammingLanguageNotAdequate* $\equiv \neg(\exists \text{adequate}.\textit{ProgrammingLanguage})$

**II.4 Axiomas da classe Risco de Integração e Testes**

Para a classe Risco de Integração e testes e suas subclasses foram definidos os seguintes relacionamentos e axiomas.

**Ambiente:**

- Para todo projeto em que não há hardware suficiente para integração e testes, então existe risco de ambiente de integração para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectNotHasSufficientHardwareForIntegrationAndTest$
◦ $ProjectNotHasSufficientHardwareForIntegrationAndTest \equiv$
$\neg(\exists \text{sufficient}.HardwareForIntegrationAndTest)$

- Para todo projeto em que não é possível desenvolver um cenário realista de tráfego de dados, então existe risco de ambiente de integração e testes para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectNotHoldsDataTrafficScenario$
◦ $ProjectNotHoldsDataTrafficScenario \equiv \neg(\exists \text{holds}.DataTrafficScenario)$

- Para todo projeto em que não é possível desenvolver um cenário realista de respostas a eventos de tempo real, então existe risco de ambiente de integração e testes para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectNotHoldsRealTimeResponseScenario$
◦ $ProjectNotHoldsRealTimeResponseScenario \equiv \neg(\exists \text{holds}.RealTimeResponseScenario)$

- Para todo projeto em que não é possível desenvolver um cenário realista de tratamento de eventos assíncronos, então existe risco de ambiente de integração e testes para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectNotHoldsAsynchronousEventHandlingScenario$
◦ $ProjectNotHoldsAsynchronousEventHandlingScenario \equiv$
$\neg(\exists \text{holds}.AsynchronousEventHandlingScenario)$

- Para todo projeto em que não é possível desenvolver em cenário realista de interação multi-usuário, então existe risco de ambiente de integração e testes para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq \forall \text{isPartOf}.ProjectNotHoldsMultiUserInteractionScenario$

◦ $ProjectNotHoldsMultiUserInteractionScenario \equiv \neg(\exists \text{holds}.MultiUserInteractionScenario)$

- Para todo projeto em que não existe software e hardware que facilite os testes de integração, então existe risco de ambiente de integração e testes para este projeto.

◦ $IntegrationAndTestEnvironmentRisk \sqsupseteq (\forall \text{isPartOf}.ProjectNotUsesHardwareFacilitatesIntegrationTest) \sqcap (\forall \text{isPartOf}.ProjectNotUsesNonDevelopmentalSoftwareFacilitatesIntegrationTest)$

◦ $ProjectNotUsesHardwareFacilitatesIntegrationTest \equiv \neg(\exists \text{uses}.HardwareFacilitatesIntegrationTest)$

◦ $HardwareFacilitatesIntegrationTest \equiv \exists \text{facilitates}.IntegrationTest$

◦ $ProjectNotUsesNonDevelopmentalSoftwareFacilitatesIntegrationTest \equiv \neg(\exists \text{uses}.NonDevelopmentalSoftwareFacilitatesIntegrationTest)$

◦ $NonDevelopmentalSoftwareFacilitatesIntegrationTest \equiv \exists \text{facilitates}.IntegrationTest$

**Produto:**

- Para todo projeto em que o hardware alvo não está sempre disponível quando necessário, então existe risco de integração e teste de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectTargetHardwareNotAvailable$

◦ $ProjectTargetHardwareNotAvailable \equiv \neg(\exists \text{available}.TargetHardware)$

- Para todo projeto em que existe um requisito para o qual não foram definidos critérios formais de aceitação, então existe risco de integração e testes de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectHoldsRequirementNotDefineFormalAcceptanceCriteria$

◦ $ProjectHoldsRequirementNotDefineFormalAcceptanceCriteria \equiv \exists \text{holds}.RequirementNotDefineFormalAcceptanceCriteria$

◦ $RequirementNotDefineFormalAccepatanceCriteria \equiv \neg(\exists \text{define}.FormalAcceptanceCriteria)$

- Para todo projeto em que existe uma interface externa que não é bem definida ou bem documentada, então existe risco de integração e teste de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq (\forall \text{isPartOnly}.ProjectExternalInterfaceNotDefined) \sqcup (\forall \text{isPartOnly}.ProjectExternalInterfaceNotDocumented)$

◦ $ProjectExternalInterfaceNotDefined \equiv \neg(\exists \text{define}.ExternalInterface)$

◦ $ProjectExternalInterfaceNotDocumented \equiv \neg(\exists \text{documented}.ExternalInterface)$

- Para todo projeto em que existe um requisito para o qual não é fácil executar teste de integração de produto, então existe risco de integração e teste de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementNotEasyToTest$

◦ $ProjectRequirementNotEasyToTest \equiv \neg(\exists \text{easyTest}.Requirement)$

- Para todo projeto em que a integração de produto não foi especificada suficientemente, então existe risco de integração e teste de produto para este projeto.

◦ $ProductIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectProductIntegrationNotSpecified$

◦ $ProjectProductIntegrationNotSpecified \equiv \neg(\exists \text{specify}.ProductIntegration)$

**Sistema:**

- Para todo projeto em que a integração de sistema não foi especificada suficientemente, então existe risco de integração e teste de sistema para este projeto.

◦ $SystemIntegrationAndTestRisk \sqsupseteq \forall \text{isPartOf}.ProjectSystemIntegrationNotSpecified$
◦ $ProjectSystemIntegrationNotSpecified \equiv \neg(\exists \text{specify}.SystemIntegration)$

- Para todo projeto em que existe um contratante que não faz parte do time de integração de sistema, então existe risco de integração e teste de sistema para este projeto.

◦ $SystemIntegrationAndTestRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectContractorNotPartOfIntegrationTeam$
◦ $ProjectContractorNotPartOfIntegrationTeam \equiv$
$\exists \text{participate}.ContractorNotPartOfIntegrationTeam$
◦ $ContractorNotPartOfIntegrationTeam \equiv \neg(\exists \text{isPartOf}.IntegrationTeam)$

- Para todo projeto em que a integração do sistema não será executada no local do cliente, então existe risco de integração e teste de sistema para este projeto.

◦ $SystemIntegrationAndTestRisk \sqsupseteq$
$\forall \text{isPartOf}.ProjectSystemIntegrationNotOnCostumersSite$
◦ $ProjectSystemIntegrationNotOnCostumersSite \equiv$
$\exists \text{performs}.SystemIntegrationNotOnCostumersSite$
◦ $SystemIntegrationNotOnCostumersSite \equiv \neg(\exists \text{takePlace}.CostumersSite)$

### II.5 Axiomas da classe Risco de Especialidades de Engenharia

Para a classe Risco de Especialidades de Engenharia e suas subclasses, foram definidos os seguintes relacionamentos e axiomas.

**Manutenabilidade:**

- Para todo projeto que seu produto será dificil de manter, então existe risco de manutenabilidade para este projeto.

◦ *MaintainbilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectProductDificultMaintain*
◦ *ProjectProductDificultMaintain* $\equiv \exists$ difficultMaintain .*Product*

- Para todo projeto em que a fase de implementação será dificil de entender, então existe risco de manutenabilidade para este projeto.

◦ *MaintainbilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectImplementationDificultUnderstand*
◦ *ProjectImplementationDificultUnderstand* $\equiv$
$\exists$ difficultUnderstand .*ImplementationsWorkflow*

- Para todo projeto em que a arquitetura pode trazer dificuldade para manutenção, então existe risco de manutenabilidade para este projeto.

◦ *MaintainbilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectArchitectureDifficultMaintain*
◦ *ProjectArchitectureDifficultMaintain* $\equiv \exists$ difficultMaintain .*Architecture*

- Para todo projeto que a construção do código pode trazer dificuldades para manutenção, então existe risco de manutenabilidade para este projeto.

◦ *MaintainbilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectCodeDifficultMaintain*
◦ *ProjectCodeDifficultMaintain* $\equiv \exists$ difficultMaintain .*Code*

- Para todo projeto em que a modelagem pode trazer dificuldade para manutenção, então existe risco de manutenabilidade para este projeto.

◦ *MaintainbilityRisk* $\sqsupseteq \forall$ isPartOf .*ProjectDesignDifficultMaintain*
◦ *ProjectDesignDifficultMaintain* $\equiv \exists$ difficultMaintain .*Design*

**Confiabilidade:**

- Para todo projeto em que requisito de confiança não foi alocado, então existe risco de confiabilidade para este projeto.

◦ $RealiabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectReliabilityRequirementNotAllocated$
◦ $ProjectReliabilityRequirementNotAllocated \equiv \neg(\exists \text{allocated}.ReliabilityRequirement)$

- Para todo projeto em que requsito de avaliação não foi alocado, então existe risco de confiabilidade para este projeto.

◦ $ReliabilityRisk \sqsupseteq \forall \text{isPartOf}.ProjectEvaluationRequirementNotAllocated$
◦ $ProjectEvaluationRequirementNotAllocated \equiv \neg(\exists \text{allocated}.EvaluationRequirement)$

**Proteção:**

- Para todo projeto em que requisito de proteção não foi alocado, então existe risco de proteção para este projeto.

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectSafetyRequirementNotAllocated$
◦ $ProjectSafetyRequirementNotAllocated \equiv \neg(\exists \text{allocated}.SafetyRequirement)$

- Para todo projeto que possui um requisito inviável, então existe risco de proteção para este projeto.

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementNotFeasible$
◦ $ProjectRequirementNotFeasible \equiv \neg(\exists \text{feasible}.Requirement)$

- Para todo projeto que possui um requisito não demonstrável, então existe risco de proteção para este projeto.

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementNotDemonstrable$
◦ $ProjectRequirementNotDemonstrable \equiv \neg(\exists \text{demonstrable}.Requirement)$

- Para todo projeto que possui um requisito de proteção que é dificil de verificar sua satisfação, então existe risco de proteção para este projeto.

◦ $SafetyRisk \sqsupseteq \forall \text{isPartOf}.ProjectSafetyRequirementHardEvaluateSatisfaction$
◦ $ProjectSafetyRequirementHardEvaluateSatisfaction \equiv$
$\exists \text{hardEvaluateSatisfaction}.SafetyRequirement$

**Segurança:**

- Para todo projeto que possui um requisito com nível de segurança nunca antes realizado, então existe risco de segurança para este projeto.

◦ $SecurityRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementSecurityLevelNeverDone$
◦ $\text{ProjectRequirementSecurityLevelNeverDone} \equiv \exists \text{securityLevelNeverDone}.Requirement$

- Para todo projeto que possui um requisito que não é precedente, então existe risco de segurança para este projeto.

◦ $SecurityRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementNotPrecedent$
◦ $ProjectRequirementNotPrecedent \equiv \neg(\exists \text{precedent}.Requirement)$

**Fatores Humanos:**

- Para todo projeto que possui um requisito de fatores humanos não satisfeito, então existe risco de fatores humanos para este projeto.

◦ $HumanFactorsRisk \sqsupseteq \forall \text{isPartOf}.ProjectHumanFactorsRequirementNotSatisfied$

◦ $ProjectHumanFactorsRequirementNotSatisfied \equiv \neg(\exists \text{satisfy}.HumanFactorsRequirement)$

**Especificações:**

- Para todo projeto em que a documentação de design não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf}.ProjectDesignDocumentationNotAdequate$

◦ $ProjectDesignDocumentationNotAdequate \equiv \neg(\exists \text{adequate}.DesignDocumentation)$

- Para todo projeto em que a documentação de implementação não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf}.ProjectImplementationDocumentationNotAdequate$

◦ $ProjectImplementationDocumentationNotAdequate \equiv \neg(\exists \text{adequate}.ImplementationDocumentation)$

- Para todo projeto em que a documentação de teste do sistema não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf}.ProjectTestDocumentationNotAdequate$

◦ $ProjectTestDocumentationNotAdequate \equiv \neg(\exists \text{adequate}.TestDocumentation)$

- Para todo projeto em que a documentação de especificação de requisitos não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf}.ProjectRequirementSpecificationNotAdequate$

◦ $ProjectRequirementSpecificationNotAdequate \equiv \neg(\exists \text{adequate}.RequirementSpecification)$

- Para todo projeto em que a especificação de hardware não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf} .ProjectHardwareSpecificationNotAdequate$
◦ $ProjectHardwareSpecificationNotAdequate \equiv \neg(\exists \text{adequate} .HardwareSpecification)$

- Para todo projeto que possui uma interface externa que não está bem especificada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf} .ProjectExternalInterfaceNotSpecified$
◦ $ProjectExternalInterfaceNotSpecified \equiv \neg(\exists \text{specify} .ExternalInterface)$

- Para todo projeto em que a especificação de testes não é adequada, então existe risco de especificações para este projeto.

◦ $SpecificationRisk \sqsupseteq \forall \text{isPartOf} .ProjectTestSpecificationNotAdequate$
◦ $ProjectTestSpecificationNotAdequate \equiv \neg(\exists \text{adequate} .TestSpecification)$

# Assinaturas

**Fernando Valeriano de Almeida Lins**

**Aluno**

_____

**Frederico Luiz Gonçalves de Freitas**

**Orientador**

_____

**Cristine Martins Gomes de Gusmão**

**Co-Orientadora**

_____