Universidade Federal do Estado do Rio de Janeiro

Escola de Informática Aplicada

Amannda Lúcia Misael Alves e André Luis Caglianone Gouvêa

Técnica de inspeção de softwares baseada em técnicas de leitura e verificação de modelos UML de alto nível

Orientador: Prof. Dr. Alexandre Luis Corrêa

Rio de Janeiro

2013.1

Amannda Misael Alves e André Luis Caglianone Gouvêa

Técnica de inspeção de softwares baseada em técnicas de leitura e verificação de modelos UML de alto nível

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Alexandre Luis Corrêa

Rio de Janeiro – RJ

2013.1

Amannda Lucia Misael Alves e André Luis Caglianone Gouvêa

Técnica de inspeção de softwares baseada em técnicas de leitura e verificação de modelos UML de alto nível

Aprovado em 28 de Agosto de 2013

Banca Examinadora:

_____

Orientador: Alexandre Luis Corrêa, D.Sc., UNIRIO

_____

Leonardo Guerreiro Azevedo, D.Sc., PPGI-UNIRIO e IBM Research – Brasil

_____

Gleison dos Santos Souza, D.Sc., UNIRIO

Rio de Janeiro – RJ

2013.1

Os autores deste Projeto autorizam a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, 28 de Agosto de 2013.

Amannda Lúcia Misael Alves

André Luis Caglianone Gouvêa

DEDICATÓRIA

Dedicamos este trabalho a nossa família e aos nossos inspiradores por todo o incentivo e apoio para que esse sonho fosse realizado.

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família pelo amor incondicional, por ser meu porto seguro e razão de todo meu desenvolvimento. Obrigada por compreenderem todas as vezes que não pude estar fisicamente ao lado de vocês e pela confiança depositada em todas as minhas decisões ao longo deste caminho.

Agradeço em especial a minha irmã Brunna por todo o companheirismo, por dividir comigo esta aventura que é sair da casa dos pais e por tornar simples até as tarefas mais difíceis. Obrigada por sempre cuidar de mim e da casa nos momentos mais complicados.

Agradeço ao corpo docente e administrativo da UNIRIO pelo esforço dedicado todos esses anos, por nos proporcionar uma educação de qualidade e nos tornar profissionais completos. Em especial agradeço ao meu orientador Alexandre Correa por sua dedicação, paciência e conhecimento que sempre nos ajudaram não só nesta monografia como ao longo da faculdade.

Agradeço aos meus amigos de graduação pelo companheirismo em todos esses anos, pelo apoio nas etapas mais complicadas e pelas comemorações a cada vitória: Juliana Carvalho, Bruna Christina, Marina Vinhaes e Raphael Rodrigues. Incluo um agradecimento especial ao meu amigo André Caglianone pela parceria ao longo da faculdade, inclusive neste projeto final.

Agradeço também aos amigos de trabalho, por me ensinarem a cada dia, por proporcionarem meu crescimento profissional e também pessoal, pela confiança e respeito mútuo e pela torcida pelo sucesso deste projeto.

Amannda Lucia Misael Alves

Primeiramente, agradeço ao meu orientador Alexandre Corrêa por me ajudar a construir o conhecimento necessário para a realização deste trabalho e aos demais professores do curso de Sistemas de Informação pela formação de excelência a mim concedida.

Agradeço a toda minha família por todo apoio e compreensão: aos meus pais Lauro e Maria e meu irmão Marcelo por ajudarem a me manter sempre no caminho certo e não desistir deste sonho.

Em especial, gostaria de agradecer a minha noiva Natalia pelo grande incentivo nos momentos difíceis, me apoiando incondicionalmente, e pelo seu positivismo (sempre presente) que me fez acreditar mais na minha capacidade.

Gostaria de agradecer também a todos aqueles que de alguma forma me ajudaram a alcançar este objetivo. A todos vocês, meu muito obrigado!

André Luis Caglianone Gouvêa

RESUMO

A inspeção de software é uma etapa de grande importância para garantir a qualidade, reduzir custos de manutenção e auxiliar na adequação do produto às necessidades reais do cliente. Dos três macroprocessos de qualidade de software (verificação, validação e teste) a verificação foi escolhida como objeto de estudo por abranger as etapas iniciais do ciclo vida de desenvolvimento de softwares. Entre as principais linhas de estudo correspondentes a este tema encontram-se as Técnicas de Leitura de Software, que têm por objetivo auxiliar na análise individual dos artefatos, aumentando a eficácia do inspetor. A verificação de artefatos também ganha destaque como método empregado nas inspeções por utilizar produtos que servem como insumo para as atividades seguintes do ciclo de desenvolvimento, evitando a transmissão dos defeitos provenientes do levantamento de requisitos e análise do sistema. Este trabalho se propõe a agregar técnicas de leitura e de verificação de modelos a fim de desenvolver uma nova técnica, aplicável tanto em um ambiente acadêmico quanto empresarial e aos principais modelos atualmente utilizados. Com o intuito de demonstrar os benefícios e identificar pontos de melhoria em sua utilização apresenta-se um exemplo de aplicação da técnica com documentos de visão, casos de uso e um fragmento do modelo de classes de um projeto real finalizado. Como resultado foi demonstrado que a técnica desenvolvida proporcionou a identificação dos principais tipos de defeitos encontrados no levantamento inicial de requisitos e permitiu sua customização para atender aos objetivos de cada etapa ou cliente.

ABSTRACT

Software inspection is a very important step to ensure quality, reduce maintenance costs and assist in adapting the product to the real needs of the customer. Of the three macroprocess of software quality (verification, validation and testing) the verification was chosen as the goal of this work because it covers the initial stages of software development life cycle. Among the main lines of study related to this topic are the Reading Techniques, which aim to assist in the individual analysis of artifacts, increasing the effectiveness of the inspector. The verification of artifacts is also emphasized as a method used in inspections by using products that serves as input for the next activities of the development cycle, preventing the transmission of defects from the requirements analysis and systems analysis. This study aims to investigate reading techniques and model checking in order to develop a new technique, applicable both in an academic environment as business and in main models currently used. In order to demonstrate the benefits and identify areas for improvement in their use presents an example of application of the technique using view documents, use cases, and a fragment of the class model of a real project already concluded. As a result, it was shown that the developed technique provides the identification of the main types of defects found in the initial survey requirements and allowed its customization to meet the goals of each stage or customer.

LISTA DE FIGURAS

Figura 1 – Relação entre os estágios do processo de desenvolvimento de software e o custo de realizar uma alteração em cada um deles 15

Figura 2 – Processo de inspeção definido em [Sauer, 2000] 19

Figura 3 – Representação das técnicas de leitura horizontal e vertical 23

Figura 4 – Famílias de técnicas de leitura 35

LISTA DE TABELAS

Tabela 1 – Técnicas de Leitura Horizontais e Verticais 24
Tabela 2 – Tipos de defeitos utilizados na técnica desenvolvida 37
Tabela 3 – Matriz CRUD das entidades em relação às funcionalidades do exemplo da aplicação 71
Tabela 4 – Quantidade de defeitos encontrados na aplicação da técnica 75
Tabela 5 – Tempo gasto na aplicação da técnica no cenário proposto 77

LISTA DE ABREVIATURAS

SDLC – Systems Development Life Cycle (Ciclo de Vida do Desenvolvimento de Sistemas)

UML – Unified Modeling Language (Linguagem de Modelagem Unificada)

RUP – Rational Unified Process (Processo Unificado Rational)

CRUD – Create, Retrieve, Update, Delete (Criação, Consulta, Alteração, Exclusão)

CBR – Checklist-Based Reading (Leitura Baseada em Checklist)

UBR –Usability-Based Reading (Leitura Baseada em Usabilidade)

DBR –Defect-Based Reading (Leitura Baseada em Defeitos)

OORT –Object-Oriented Reading Technique (Técnica de Leitura Orientada a Objeto)

TBR – Tracebility-Based Reading (Leitura Baseada em Rastreabilidade)

SBR – Scenario-Based Reading (Leitura Baseada em Cenários)

PBR – Perspective-Based Reading (Leitura Baseada em Perspectivas)

OO-PBR – Object-Oriented Perspective-Based Reading (Leitura Baseada em Perspectivas Orientadas e Objeto)

SUMÁRIO

INTRODUÇÃO **14**
1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO 14
1.2 OBJETIVO DO TRABALHO 15
1.3 ESTRUTURA DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA **17**
2.1 QUALIDADE DE SOFTWARE 17
2.2 INSPEÇÃO DE SOFTWARE 18
2.3 TÉCNICAS DE LEITURA DE SOFTWARE 20
2.3.1 *Ad hoc* 21
2.3.2 Leitura baseada em *checklist* (CBR) 21
2.3.3 Leitura baseada em usabilidade (UBR) 22
2.3.4 Leitura baseada em defeitos (DBR) 22
2.3.5 Leitura orientada a objetos (OORTs) 22
2.3.6 Leitura baseada em cenário (SBR) 25
2.3.6.1Leitura baseada em perspectiva (PBR) 25
2.3.7OO-PBR 28
2.4 TÉCNICAS DE VERIFICAÇÃO DE MODELOS 28
2.4.1 Modelo de Classes e Casos de Uso (Cenários) - Martin Glinz 29
2.4.2 Modelo de Classes e Casos de Uso - Georg Kösters 32
2.5 Estudos Comparativos entre Técnicas apresentadas 34
2.6 Conceitos Complementares 37
3 TÉCNICA DESENVOLVIDA **40**
3.1 Definição da Técnica 40
3.1.1 Verificação do Documento de Visão do negócio e construção de artefatos iniciais da Análise 41
3.1.2 Verificação dos Casos de Uso em relação ao Modelo de Classes 43
3.2 Resumo da Técnica 44
3.3 Cenários Desenvolvidos 45
3.4 Guia de Verificação entre Casos de Uso e Modelos de Classes 52
4 EXEMPLO DE APLICAÇÃO DA TÉCNICA **55**

4.1 Etapa 1 – Verificação do Documento de visão do negócio e Construção de artefatos iniciais da análise 55

4.1.1 PBR – Projestista 55

4.1.2 PBR – Usuário 63

4.1.3 Verificação das Interações via Matriz CRUD 69

4.2 Etapa 2 – Verificação dos Casos de Uso em relação ao Modelo de Classes 72

4.2.1 Inclusão de Referências 72

4.2.2 Análise dos Casos de Uso utilizados 74

4.3 Resultados Obtidos na Aplicação 75

5 ANÁLISE DOS RESULTADOS 76

REFERÊNCIAS 78

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

Ao longo dos anos o processo de desenvolvimento de sistemas vem sendo aprimorado visando aumento na qualidade, diminuição no tempo de produção e, consequentemente, menores custos. Este processo, também chamado Ciclo de Vida do Desenvolvimento de Sistemas, SDLC inclui diversas fases com metas específicas dentro do contexto da construção do sistema, buscando a divisão de responsabilidades.

Existem várias abordagens para o SDLC, mas a maioria possui cinco fases em comum: levantamento de requisitos, análise, projeto, implementação e manutenção. A interlocução entre estas atividades é mantida por fluxos de informação, que são as saídas das atividades servindo de insumo para as demais. Contudo, normalmente ocorrem desvios e perdas de informação nestes fluxos, causando diferenças entre os objetivos do patrocinador[1] e a solução final do sistema.

A prevenção de erros deve ser realizada em todas as etapas do SDLC para evitar a necessidade de ajustes e correções posteriores, já que toda alteração realizada causa impacto nos custos e no cronograma do projeto. Segundo pesquisa realizada por Boelm e Papaccio, de 25% a 40% do orçamento dos projetos de software são gastos com correções nos diversos estágios do ciclo de vida. A Figura 1 ilustra a proporção entre as alterações realizadas e os custos relacionados, para cada estágio do SLDC.

Para auxiliar na diminuição do retrabalho e dos custos gerados pelos defeitos de um sistema, devem ser promovidas revisões nos artefatos gerados por cada fase de seu ciclo de vida. Uma das abordagens de revisão é a inspeção de software, que possui um processo de detecção de defeitos bem definido. Existem diversas técnicas de inspeção de software específicas para a minimização das inconsistências destes artefatos.

Contudo, a quantidade de empresas onde a inspeção de software faz parte da cultura e do processo organizacional ainda é pequena face os benefícios de sua utilização. Por isso, o desenvolvimento de técnicas que sejam aplicáveis nos cenários normalmente encontrados nas empresas de software é de extrema importância e pode tornar a utilização de inspeção efetiva.

[1] Pessoa ou grupo de pessoas responsáveis pela iniciativa de construção do sistema.

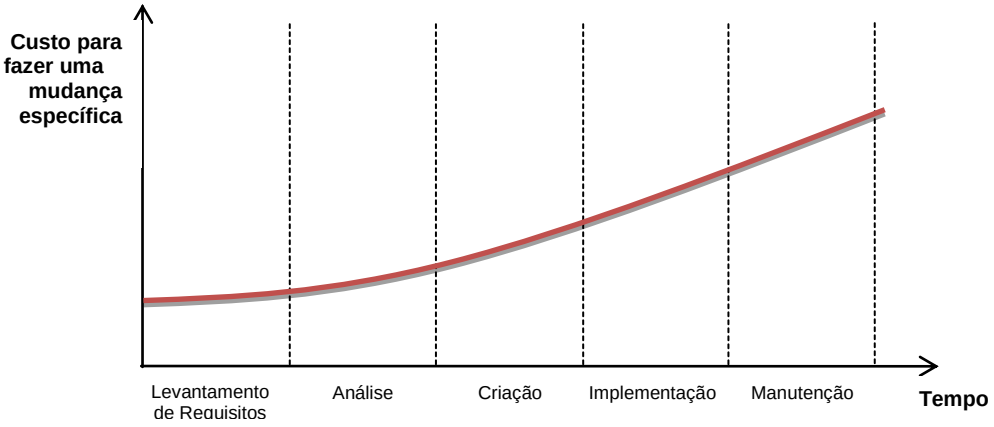


Figura 1 – Relação entre os estágios do processo de desenvolvimento de software e o custo de realizar uma alteração em cada um deles.

Fonte: Adaptação da figura 12.5, página 468, do livro Princípios de Sistemas de Informação [Stair, 2010].

1.2 OBJETIVO DO TRABALHO

O objetivo deste trabalho é se basear nas técnicas de inspeção de software comumente utilizadas para desenvolver uma nova técnica aplicável tanto no ambiente acadêmico quanto empresarial, que proporcione simplicidade e rapidez na identificação de defeitos em artefatos produzidos nas fases de levantamento de requisitos e análise do sistema e, sobretudo, melhora na qualidade dos softwares produzidos.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido em mais quatro partes.

O capítulo dois contém os principais conceitos estudados para o desenvolvimento do projeto, concentrados em dois grupos principais: técnicas de leitura de softwares e técnicas de revisão de casos de uso e modelos de classe.

O terceiro capítulo apresenta em detalhes a técnica de inspeção de softwares desenvolvida, organizada em duas etapas: verificação do documento de visão do negócio e construção de artefatos iniciais da análise, e verificação dos casos de uso em relação ao modelo de classes. O material desenvolvido para guiar o inspetor ao longo da técnica de inspeção de softwares também é apresentado.

Um exemplo de aplicação da técnica é apresentado no Capítulo 4 para elucidar as principais atividades envolvidas na inspeção proposta e identificar a sua conformidade as características previstas.

Por fim, o Capítulo 5 apresenta a análise dos resultados obtidos, com vantagens e desvantagens, e as propostas de trabalhos futuros para este projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os principais conceitos e técnicas de inspeção de software, utilizados como objetos de estudo. As seções do capítulo especificam os fundamentos que alicerçam a técnica desenvolvida neste trabalho.

2.1 QUALIDADE DE SOFTWARE

A complexidade e o escopo dos sistemas vêm crescendo, acompanhando a evolução dos processos organizacionais. Por esses motivos, a garantia de qualidade de software[2], extremamente importante para a satisfação das necessidades dos clientes, tem se tornado mais difícil e custosa.

Além disso, a construção de um sistema é realizada por diversas pessoas que interagem em momentos distintos do processo de produção e, ao evoluir a solução, podem também inserir problemas.

Os problemas encontrados em um sistema podem ser classificados em três tipos:

- **Defeito:** Problema de mais baixo nível do sistema, relacionado à implementação;
- **Erro:** Resposta inesperada do sistema, desencadeada por um defeito;
- **Falha:** Comportamento da aplicação que não corresponde ao comportamento esperado, em decorrência de um problema na especificação.

Existem alguns tipos de defeitos que podem ser inseridos nas etapas do ciclo de desenvolvimento da aplicação. Para um melhor entendimento, suas definições estão listadas a seguir:

- **Omissão:** Falta de representação ou definição de informação relevante para o projeto;
- **Fato Incorreto:** Algo que está no sistema e que vai de encontro a algo especificado;
- **Inconsistência:** A mesma informação definida de formas diferentes no projeto;

[2] **“**Conformidade a requisitos funcionais e de desempenho que foram explicitamente declarados, a padrões de desenvolvimento claramente documentados, e a características implícitas que são esperadas de todo software desenvolvido por profissionais”. (Pressman, 2002)

- **Ambiguidade:** Informação com falta de clareza na especificação, causando problemas na sua interpretação;
- **Informação Estranha**: Informação especificada não é utilizada em nenhum artefato de desenvolvimento do software.

Para encontrar e tratar estes defeitos, é preciso primeiramente conhecer os conceitos inseridos no contexto de qualidade de software. A verificação, a validação e o teste são os três macroprocessos que contém as atividades relacionadas à investigação e constatação de erros atuando em todas as fases do desenvolvimento, e estão definidas a seguir:

- **Verificação:** Processo que tem por objetivo conferir se os artefatos gerados ao longo da etapa de desenvolvimento estão de acordo com a especificação do software;
- **Validação:** Processo que tem por objetivo conferir se o produto está concordante com os requisitos do sistema;
- **Teste:** Processo que tem por objetivo conferir se o produto possui o comportamento esperado através de sua execução.

Com o intuito de viabilizar a garantia da qualidade de software, diversas técnicas de verificação, validação e teste foram elaboradas e estão sendo utilizadas por empresas do ramo de desenvolvimento de sistemas.

Apesar disso, problemas são frequentemente encontrados, fato que gera grandes impactos nos custos e nos prazos para a conclusão dos projetos. Como visto na figura 1, o impacto pode ser reduzido com a descoberta dos erros nas fases iniciais do SDLC do sistema. Uma abordagem de verificação que tem revelado bons resultados é a inspeção de software.

2.2 INSPEÇÃO DE SOFTWARE

A inspeção foi introduzida na área computacional como proposta para análise das etapas de projeto e codificação do ciclo de vida de desenvolvimento de software, a fim de auxiliar na garantia da qualidade de software [Fagan, 1976].

Diversos estudos foram realizados comprovando a eficiência deste método. Verificou-se que a inspeção pode ser realizada também nas demais fases do processo de desenvolvimento, com ganhos expressivos na qualidade final do produto e com redução dos custos de reparo dos defeitos. Outra vantagem relevante abordada é sua eficiência que usualmente atinge taxas entre 30 a 75% [Bernard e Price, 1994].

Outra vantagem em destaque é o baixo custo e esforço empregados na correção dos defeitos identificados na inspeção em detrimento dos defeitos encontrados em fases posteriores como na etapa de testes. Pesquisas indicam que em média 1,75 horas são utilizadas na correção de um defeito encontrado na inspeção de software, 1,46 horas em um defeito encontrado na inspeção de código e 17 horas em um defeito encontrado nos testes [Kelly, 1992].

O processo de inspeção de software, definido em [Fagan, 1986], é algo bastante discutido e já passou por várias modificações desde sua primeira versão. Nos estudos realizados por Votta [1993], Johnson [1996] e Basili [1999] o processo original foi analisado com a identificação da necessidade de melhoria. Em [Sauer, 2000] uma reestruturação do processo foi proposta, sendo considerada como a versão atualizada do processo de inspeção (Figura 2).

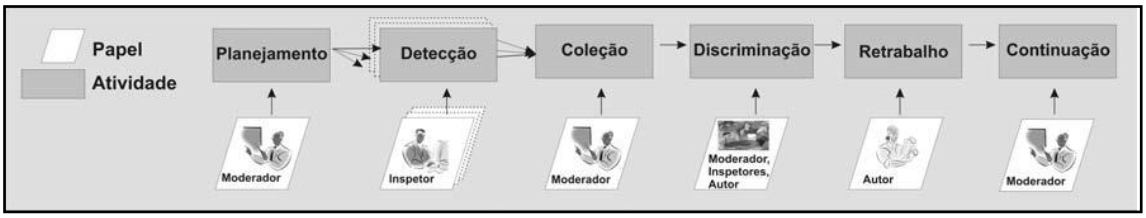


Figura 2 – Processo de inspeção definido em [Sauer, 2000].

Fonte: Kalinowski *et al*., 2004.

As atividades deste processo podem são definidas abaixo [Kalinowski *et al*., 2004]:

- **Planejamento:** Pessoa que atua como moderador da inspeção define as técnicas a serem utilizadas, determina os artefatos a serem inspecionados, seleciona os inspetores e distribui os artefatos entre eles;
- **Detecção de Defeitos:** Os inspetores realizam a inspeção a procura de defeitos e produzem um documento contendo os defeitos encontrados (lista de discrepâncias);

- **Coleção de Defeitos:** Moderador agrupa as listas de discrepâncias, eliminando os defeitos repetidos;
- **Discriminação de Defeitos:** Moderador, o autor do artefato inspecionado e os inspetores discutem sobre as discrepâncias encontradas em cada artefato, com o intuito de verificar se a discrepância é, de fato, defeito;
- **Retrabalho:** Autor do documento realiza as alterações necessárias para corrigir os defeitos, produzindo um relatório explicando o trabalho realizado;
- **Continuação:** Com base no relatório produzido na correção dos defeitos o Moderador decide se realizará uma nova inspeção no artefato.

Podemos separar as técnicas de inspeção em dois grandes grupos: técnicas de leitura, que procuram orientar o inspetor[3] na leitura do artefato para determinados aspectos; técnicas de verificação de modelos, que propõem um conjunto de procedimentos de verificação e validação.

2.3 TÉCNICAS DE LEITURA DE SOFTWARE

As técnicas de leitura de software têm por objetivo auxiliar na análise individual dos artefatos, buscando aumentar a eficácia do inspetor na verificação e validação do software [Travassos *et al.*, 1999].

Estudos na área de inspeção de software apontam para a relevância da preparação individual dos revisores para a eficiência do processo de revisão [Christenson, 1990; Laitenberger *et al*., 2002].

As técnicas de leitura tornam-se importantes neste caso porque indicam ao revisor uma série de etapas que devem ser seguidas para a análise individual de um artefato, de forma a proporcionar o entendimento da tarefa ou problema em questão e, consequentemente, a detecção de suas falhas [Basili, 1996].

Neste documento as técnicas mais relevantes, segundo os diversos trabalhos estudados, para o mercado e para o projeto proposto serão abordadas individualmente, de

[3]Pessoa encarregada de inspecionar determinado artefato ou conjunto de artefatos de software.

forma a construir uma opinião abrangente e formalizada sobre as técnicas de leitura de maior importância.

2.3.1 *Ad hoc*

Ad hoc é o método mais empregado atualmente no mercado para as inspeções. Ela não possui nenhuma orientação ou método aos revisores, ou seja, não segue nenhuma técnica de leitura, simplesmente propõe a detecção do maior número de defeitos possíveis através da análise dos artefatos em mãos [Lahtinen, 2012].

Esta abordagem é altamente dependente do conhecimento e das habilidades dos revisores, tornando-se uma opção não recomendada para usuários inexperientes. Além disso, não há garantias de que todo o documento foi analisado, já que o único produto é uma lista dos defeitos encontrados.

2.3.2 Leitura Baseada em *Checklist* (CBR)

Esta técnica, também chamada de CBR (em inglês Checklist-Based Reading), consiste na utilização de um formulário com uma lista de questões (*checklist*) que deverão ser respondidas pelo inspetor. Não existe um padrão para a configuração do questionário, o que torna a inspeção mais flexível. Entretanto, uma boa prática é utilizar esta flexibilidade para direcionar a leitura do inspetor para os aspectos que mais interessam ser verificados dentro do contexto em questão.

Sabendo-se que o *checklist* guia a leitura do inspetor, um ponto relevante a ser observado é que a construção do questionário utilizado na CBR deve ser feita com atenção para que não sejam ignorados aspectos importantes na inspeção dos artefatos[4]. Por isso, esses aspectos devem ser definidos antes da confecção do questionário e, após estar pronto, espera-se que todas as características de qualidade definidas sejam atingidas pelas perguntas do *checklist*.

A cobertura do documento inspecionado depende diretamente do questionário elaborado e do inspetor que irá realizar a leitura e, quanto maior a cobertura, maior será o custo na execução da inspeção [Travassos, 2007].

[4] Todo documento gerado no ciclo de vida de desenvolvimento do software

2.3.3 Leitura baseada em usabilidade (UBR)

UBR propõe uma abordagem de verificação de softwares baseada em casos de uso. Estes seriam empregados nas inspeções ao longo de toda etapa de desenvolvimento (requisitos, projeto e código) e o objetivo é que o revisor leia os documentos enquanto simula uma execução manual do caso de uso e identifica as falhas existentes [Mafra, 2005].

Outra diferença relevante é que tal técnica se baseia na priorização dos casos de uso e, consequentemente, das funcionalidades mais relevantes para o sistema. Apenas esses casos de uso serão executados para evitar as falhas mais críticas e que comprometam mais gravemente a qualidade do sistema.

2.3.4 Leitura Baseada em Defeitos (DBR)

Representa uma família de técnicas de leitura para a detecção de defeitos em documentos de requisitos descritos em SCR, uma notação formal baseada em máquinas de estado [Mafra, 2005].

Cada revisor é responsável por analisar os artefatos procurando por um tipo específico de falhas, utilizando para isto cenários derivados de questões de *checklist* [Lahtinen, 2011].

2.3.5 Leitura Orientada a Objetos (OORTs)

Esta técnica propõe revisões individuais de diversos diagramas e documentos de projetos orientados a objetos.

Para isto duas técnicas devem ser empregadas: leitura horizontal e vertical. A leitura horizontal tende a encontrar mais defeitos de ambiguidade e inconsistência entre documentos de especificação gerados na mesma atividade do ciclo de vida de desenvolvimento de softwares. A leitura vertical tende a encontrar mais defeitos de omissão e funcionalidade incorreta e, ao contrário da leitura vertical, verifica artefatos de diferentes atividades do ciclo de vida de desenvolvimento.

A Figura 3 apresenta os diferentes tipos de revisão.

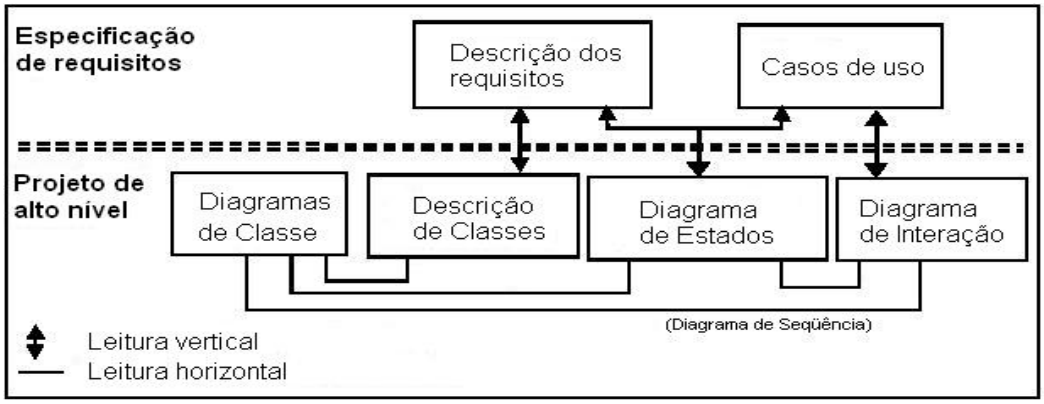


Figura 3 – Representação das técnicas de leitura horizontal e vertical.

Fonte: Mafra, 2005.

- Leitura Baseada em Rastreabilidade (TBR)

A técnica TBR (em inglês Traceability-Based reading), apresentada em [Travassos *et al.*, 1999], é uma combinação de leituras verticais e horizontais que possui enfoque nos artefatos de alto nível gerados em projetos que utilizam o paradigma da orientação a objetos e UML[5].

Esse processo permite o aprimoramento dos artefatos produzidos pela equipe de análise, garantindo que sejam capazes de refletir corretamente as necessidades dos *stakeholders*[6], evitando os defeitos de software decorrentes de uma falha nesta identificação [Travassos *et al.*, 1999]. O produto da inspeção deve proporcionar ao desenvolvedor um conjunto de informações completas, corretas, consistentes e sem ambiguidade.

Os artefatos de alto nível são formas de representação que auxiliam os desenvolvedores a entender o problema. Baseado nisso, Travassos *et al.* [1999] criaram esta técnica por considerar que a sua utilização aumenta as chances do entendimento correto sobre

[5]Unified Model Language , principal forma de visualização, especificação, construção e documentação de artefatos de software orientados à objetos [OMG Site].

[6]O termo inglês stakeholder (que poder ser traduzido como 'parte interessada') designa uma pessoa, grupo ou entidade com legítimos interesses no desempenho de uma organização, como um todo ou em uma área de negócios específica, e cujas decisões e atuações possam afetar, direta ou indiretamente, essa mesma organização.

o problema por parte do desenvolvedor, o que diminui os riscos da solução definida ser inadequada.

A leitura é dita "baseada em rastreabilidade" pelo fato de ocorrer um rastreamento entre as informações dos artefatos do projeto e a parte funcional do sistema. O objetivo é verificar a consistência dos artefatos e se eles estão corretos e completos em relação à parte funcional. Para isso, o TBR fornece ao inspetor uma série de instruções a serem seguidas para destacar diversos conceitos dos diagramas, diferenciados normalmente pela utilização de cores. O passo seguinte é analisar estas marcações e identificar as discrepâncias entre os modelos.

As formas de aplicação sugeridas para as técnicas de leitura apresentadas por [Travassos *et al*., 1999] estão listadas na tabela 1.

Tabela 1 – Técnicas de Leitura Horizontais e Verticais

Técnicas de Leitura Horizontais
Comparação entre os diagramas de classes e as respectivas descrições das classes.
Comparação entre os diagramas de classes e os respectivos diagramas de transição de estados.
Comparação entre os diagramas de sequência e os respectivos diagramas de transição de estados.
Comparação entre os diagramas de sequência e os respectivos diagramas de classes.
Técnicas de Leitura Verticais
Comparação entre as descrições das classes e os respectivos requisitos.
Comparação entre os diagramas de sequência e os respectivos casos de uso.
Comparação entre os diagramas de estados e os respectivos requisitos e casos de uso.

Fonte: Travassos *et al*., 1999.

Um problema frequentemente encontrado nas técnicas de leitura é a dificuldade em aplicá-la para projetos em larga escala. Na TBR esse problema é bem contundente devido às verificações entre diversos artefatos de fases diferentes do ciclo de desenvolvimento. Algumas estratégias de decomposição foram definidas por Travassos *et al*. [1999] e sugerem a decomposição da inspeção por funcionalidades do sistema ou por entidades do sistema.

2.3.6 Leitura Baseada em Cenário (SBR)

A SBR (em inglês "Scenario-Based Reading"), introduzida por Porter e Votta [1994], apoia a inspeção através da utilização de um documento, denominado cenário, que contém instruções e diretrizes que orientam o inspetor a como realizar a inspeção e a encontrar defeitos [Shull *et al.*, 2003].

Os estudos da utilização da SBR em inspeção de documentos de requisitos em linguagem natural [Basili *et al.*, 1996], [Ciolkowski *et al.*, 1997], [Shull, 1998], documentos formais de requisitos [Porter, 1998], interfaces com o usuário [Zhang, 1999], projetos orientados a objetos [Laitenberger *et al.*, 2000], [Conradi, 2003] e códigos [Laitenberger, 2001] comprovaram os benefícios da técnica aplicada em várias fases do desenvolvimento de software [Shull *et al.*, 2003].

A experiência adquirida com sua utilização tornou possível o surgimento de outras técnicas mais específicas baseadas na SBR. Por isso, atualmente existe um conjunto de técnicas de leitura baseadas em cenários.

2.3.6.1 Leitura Baseada em Perspectiva (PBR)

A técnica de leitura PBR (em inglês "Perspective-Based Reading"), possui enfoque na inspeção de documentos através de cenários projetados para reproduzir pontos de vista (perspectivas) de *stakeholders* envolvidos diretamente com os artefatos inspecionados [Basili *et al.* 1996].

O principal objetivo em reproduzir as diferentes perspectivas presentes no contexto do desenvolvimento do sistema é garantir uma inspeção com ampla cobertura, sem que para isso tenham que ser realizadas apenas leituras superficiais. Cada cenário aprofunda a inspeção nos aspectos relevantes para uma perspectiva, o que aumenta as chances de encontrar defeitos que não seriam percebidos em uma leitura mais geral.

- Cenário PBR

Um cenário utilizado na PBR consiste em: uma introdução sobre os interesses do *stakeholder* no artefato inspecionado; um conjunto de instruções que indicam como obter as

informações relevantes para a inspeção; e um conjunto de perguntas que devem ser respondidas durante a execução das instruções.

Após a identificação do artefato a ser inspecionado, a construção do cenário é necessária para a realização da PBR. Contudo, alguns estudos já realizados possuem exemplos de cenários para diversos artefatos de software, e podem ser utilizados como base na construção do novo cenário. O processo para a construção de um cenário, descrito em [Laitenberger e Atkinson, 1999], possui as seguintes atividades:

- **Identificação dos documentos de revisão:** Os documentos que possuam informações importantes sobre o sistema devem ser identificados, sejam estes textuais, diagramas ou modelos;
- **Identificação dos *stakeholders*:** Os *stakeholders* envolvidos no processo de desenvolvimento do software devem ser identificados. Os de maior importância devem ser selecionados para que seus pontos de vista sirvam de perspectivas para os cenários;
- **Identificação das informações importantes:** As informações de maior relevância para cada perspectiva devem ser identificadas. Esta tarefa pode ser auxiliada por entrevistas realizadas com os *stakeholders* envolvidos;
- **Criação da introdução e das instruções do cenário:** Após as três primeiras atividades, é possível começar a escrever o cenário. A introdução deve indicar os interesses do *stakeholder* no documento inspecionado. As instruções devem ser escritas de forma a orientar o inspetor em como identificar e extrair informações importantes do artefato para um modelo, utilizando como base o que foi realizado no passo anterior;
- **Criação das perguntas:** O último passo é construir uma lista de perguntas. As perguntas devem ser escritas levando em consideração os tipos de defeitos que se deseja eliminar e os problemas normalmente encontrados no tipo de artefato inspecionado.

Um cenário poderá ser utilizado em qualquer outro artefato do mesmo tipo. Com sua utilização contínua em inspeções, podem ser identificados pontos de melhoria, que devem ser incorporados para aumentar a cobertura das inspeções futuras.

O nível de detalhamento dos cenários deve variar de acordo com a proficiência do inspetor. Para inspetores experientes não são necessários cenários com muitos detalhes. Inspetores com pouca experiência devem utilizar cenários com nível de detalhes maior, visando execução correta da leitura.

Na utilização do cenário, primeiramente o inspetor deve ler a introdução para entender o contexto e os interesses do *stakeholder* no artefato. Em seguida, a partir das instruções, deve abstrair as principais informações para um modelo. Por último, responder as perguntas sobre o modelo construído.

- Perspectivas PBR

As perspectivas são importantes, pois os *stakeholders* do artefato inspecionado possuem interesses diferentes no que tange a qualidade do documento [Ciolkowski, 1999]. Ao garantir que os principais pontos de vista do documento foram cobertos pela leitura, garante-se também uma inspeção com boa cobertura.

Para o contexto deste trabalho, as perspectivas identificadas por Laitenberger *et al.* [2000] são mais relevantes por se tratarem da visão dos *stakeholders* envolvidos com os documentos de projeto do sistema. Foram identificadas três perspectivas principais:

- **Projetista:** O cenário que representar o ponto de vista do projetista deve validar se não há desvios entre as informações correlatas dos modelos de projeto e dos documentos da análise, garantindo que estão completos e corretos entre si;
- **Testador:** Um cenário baseado na perspectiva do testador deve indicar como identificar as diversas funcionalidades do sistema nos modelos de projeto e como criar casos de teste para cada uma delas, para que assim o inspetor possa garantir que o comportamento esperado está sendo realizado;
- **Desenvolvedor:** Na perspectiva do desenvolvedor, o mais importante é garantir que toda informação contida nos documentos construídos na fase de projeto estão representadas nos modelos de projeto e são suficientes para a implementação do sistema.

Perspectivas diferentes podem ser encontradas em outros estudos e até mesmo as mesmas perspectivas, porém com cenários utilizados para validações diferentes. Além disso,

dependendo do contexto do software e dos requisitos de qualidade impostos outras perspectivas podem ser criadas.

2.3.7 OO-PBR

Segundo Mafra [2006] o desenvolvimento orientado a objetos está sendo amplamente utilizado na produção de sistemas, porém ainda não contribuiu de forma expressiva na qualidade e na produtividade do desenvolvimento de software. Ainda segundo Mafra [2006] um dos fatores para que isso ocorra é a falta de total conhecimento do engenheiro de software sobre o problema descrito no documento de requisitos.

Por isso, Mafra [2006] propõe a aplicação da PBR focada na visão do engenheiro de software em relação ao documento de requisitos, procurando aumentar o conhecimento do projetista em torno do problema a ser tratado pelo sistema e a construção de um modelo de projeto orientado a objetos preliminar de alto nível.

O engenheiro de software pode utilizar esta abordagem para tornar a sua leitura mais técnica e focada nos conceitos da orientação a objetos, porém, segundo Mafra [2006], o objetivo central do inspetor deve ser identificar discrepâncias nos requisitos.

A inspeção é divida em quatro passos:

1. Identificar classes e atributos;
2. Identificar relacionamentos e suas multiplicidades;
3. Identificar funcionalidades do sistema e identificar classes candidatas a satisfazerem-na;
4. Identificar como as classes candidatas irão satisfazer as funcionalidades do sistema através de responsabilidades.

2.4 TÉCNICAS DE VERIFICAÇÃO DE MODELOS

A produção de softwares envolve ao longo de suas etapas a construção de diversos modelos para cada nível de abstração: levantamento de requisitos, análise, desenvolvimento, implementação e documentação.

Atualmente a UML encontra-se consolidada entre os estudiosos e no mercado de trabalho como a principal linguagem destes artefatos. Seus principais diagramas são: Diagrama de Caso de Uso, Diagrama de Classes, Diagrama de Interação, Diagramas de Sequência, Diagrama de Colaboração, Diagrama de Estado e Diagrama de Atividade.

Tais artefatos são usualmente gerados por pessoas diferentes, com ferramentas de apoio e em momentos distintos, o que aumenta a possibilidade de inconsistências e incoerências entre os modelos ou parte deles.

Apesar do impacto de um problema deste tipo ser propagado por todas as etapas subsequentes do projeto, as técnicas de desenvolvimento atuais não incluem uma abordagem que permita uma verificação sistemática entre eles.

Outra linha de pesquisa amplamente difundida prevê a verificação e a validação de pares de artefatos UML, comparados de forma bidirecional a procura de diversos tipos de falhas. Normalmente são utilizados nessa comparação um artefato de visão funcional e um artefato de visão estrutural, checando a consistência das principais visões do projeto. As técnicas mais relevantes para este projeto serão discutidas a seguir.

2.4.1 Modelo de Classes e Casos de Uso (Cenários) - Martin Glinz

Martin Glinz propõe em seu artigo [Glinz, 2000] uma validação entre o modelo de classe e o modelo de caso de uso, referenciado como cenário. A adoção de tais produtos é adequada ao padrão de artefatos utilizados nas validações já que o modelo de classes representa uma visão estrutural e comportamental, enquanto os casos de uso modelam a interação existente no projeto.

O modelo de classes representa uma combinação entre diagrama de classe e de estado. Por outro lado, o cenário representa um conjunto ordenado de interações entre o sistema e atores externos. A principal vantagem dos casos de uso consiste em representar em linguagem natural, mas não puramente narrativa, como os usuários trabalharão com o sistema.

Como tais modelos baseiam-se em diferentes níveis de abstração e utilizam técnicas de modelagem distintas, sérios problemas de consistência e completude podem ser incluídos no sistema e por isso precisam ser identificados e tratados.

A validação proposta baseia-se na redução da sobreposição e na dupla referência entre as informações presentes nos dois modelos. Para isto um conjunto de regras pode ser utilizado para auxiliar no desenvolvimento de uma especificação consistente ou checar a consistência de uma especificação completa [Glinz, 2000].

Por definição, os modelos são considerados consistentes quando nenhum dos dois modelos encontra-se incompleto em relação ao outro e quando não há contradições entre eles.

A metodologia elaborada sugere a adoção de três etapas, analisadas a seguir.

- Sobreposições

As sobreposições entre os cenários e diagramas de classe ocorrem quando determinado dado ou operação é representado nos dois modelos. Esta dupla definição aumenta as chances de redundância e, consequentemente, de contradições entre os modelos. Por outro lado, a retirada sem critérios das informações de um dos diagramas onde se supõe existir uma sobreposição pode gerar lacunas na especificação.

Para orientar esta definição [Glinz, 2000] reuniu um conjunto de regras que orientam o analista a localizar corretamente a informação que deseja inserir. Suas principais orientações são: dados devem ser modelados apenas no modelo de classes; interação dos usuários e mudança de estado deve estar apenas nos cenários; detalhes de uma operação que envolva dados de uma única classe devem estar modelados no modelo de classes, caso contrário deve-se utilizar um cenário.

- Referências cruzadas

Os diagramas adotados nesta técnica não são formais, portanto, uma validação automática não se torna uma possibilidade. Para lidar com este empecilho Glinz [2000] propõe um modelo de referências cruzadas identificando as informações existentes nos dois diagramas, o que representa um dos principais diferenciais desta proposta.

Podem ser referenciados nos diagramas de classe os nomes das classes, dos seus atributos e operações. Nos diagramas de estado podem ser referenciados os nomes de estados e de ações ou eventos envolvidos nas transições de estado. Os diagramas de classe permitem algumas referências mais formais, como domínios de atributos e parâmetros, além do tipo das operações. Nos cenários, as referências são relativas aos nomes dos cenários e a ordem das interações entre sistema e usuário.

O caractere escolhido para representar uma referência nas descrições textuais é a seta para cima (↑) seguida do nome da classe, um ponto (.) e o nome do atributo ou operação que se deseja representar. Nos diagramas também é utilizada uma linha pontilhada entre os dois itens para indicar a referência.

Dois níveis de esquemas estão previstos para utilização nas referências cruzadas:

- **Esquema simples de referências cruzadas:** Sempre que uma resposta em um cenário necessitar de uma ação em um objeto de alguma classe uma referência deve ser acrescentada no cenário para indicar tal elemento no diagrama de classes. O diagrama de classes não é alterado.

- **Esquema elaborado de referências cruzadas:** Esta técnica implica em uma especificação mais formal das informações. Nos cenários são incluídas informações como os insumos e produtos de uma ação e especificações técnicas de ordem no fluxo de dados, como *check step X*, *go to step X*, *delivers* e *terminate*. Para isto um pseudocódigo de cada operação torna-se necessário. Os diagramas de classe também são alterados com o acréscimo de um item para representar o cenário e o seu relacionamento com as demais classes do modelo através da utilização de estereótipos. Os principais estereótipos definidos são *<<uses>>*, *<<sends>>*, *<<creates>>* e *<<deletes>>*. Todos eles qualificam a referência existente entre os modelos.

- Checagem de consistência

Para garantir a consistência entre os cenários e os diagramas de classe dois grupos de regras foram compilados:

- **Regras formais verificáveis:** considera uma série de regras formais, visa garantir a completude e conformidade de nomes e sintaxe entre os itens referenciados e as referências. Como um importante diferencial, pode ter o processo de verificação apoiado por uma ferramenta.

- **Regras para inspeção:** necessária para garantir a consistência de partes mais subjetivas do sistema. Seu objetivo é verificar separa toda ação não local em relação a um cenário ou diagrama de classe uma referencia está incluída e é suficiente para prover a resposta adequada, e garantir se o fluxo de dados necessário para a referência entre os dois modelos está completo tanto no envio da informação para o segundo modelo quanto na volta deste.

2.4.1 Modelo de Classes e Casos de Uso - Georg Kösters, Hans-Werner Six e Mario Winter

Körsters *et al.* [2001] propõe um método para tornar o modelo de classes e os casos de uso[7] (notação UML) rastreáveis entre si, de forma a criar um acoplamento entre os aspectos estruturais e as funcionalidades do sistema, com o objetivo de verificar a consistência e a completude entre eles.

Para que seja possível realizar tal acoplamento, os casos de uso devem ser refinados de forma a aumentar o detalhamento de suas interações. Isso é realizado através da construção de diagramas de atividades[8] customizados que proporcionam uma modelagem adequada ao comportamento do caso de uso. Dessa forma, é possível rastrear neste diagrama as transições dos casos de uso no modelo de classes. Além disso, Körsters *et al.* [2001] propõe a construção de escopos de classe que são os grupos de todas as classes presentes no caso de uso e que de alguma forma são referenciadas na execução de uma ação (criação, consulta, alteração ou remoção), para complementar o proposto acoplamento entre os dois diagramas.

- Customização do Diagrama de Atividades

[7]Descreve uma unidade de trabalho ou tarefa coerente, realizada por um ou mais atores pela interação com o sistema.

[8]Variação da máquina de estados onde cada um deles representa a aplicação de uma ação ou de subatividades e as transições são disparadas ao completar ações ou subatividades [OMG UnifiedModelingLanguageSpecification].

Para modelar um diagrama de atividades que represente o comportamento de um caso de uso, deve-se utilizar alguns estereótipos de extensão definidos no artigo. A seguir estão listados os principais estereótipos e quando devem ser utilizados.

- **Ação Contextual:** Utilizar quando houver a execução de uma ação por um ator com a ajuda do sistema.
- **Interação:** Utilizar para representar ações suportadas ou realizadas pelo sistema.
- **Ator na Ação:** Utilizar para representar os atores que realizam as ações.
- **Macro Ação:** Utilizar para abstrair um conjunto de ações ou outros diagramas de atividades.

- Modelagem de Domínio e Validação

A modelagem de domínio tem por objetivo entender e descrever as principais classes dentro do contexto de seu domínio. Ela é fundamental para a definição dos casos de uso e análise dos modelos de classes [Jacobson, 1999].

Körster *et al.*(2001) propõe um método chamado *SCORES*, para a modelagem de domínio e sua validação. Ele se baseia na divisão de Pohl e Haumer dos três tipos de informação que devem estar presentes em um caso de uso, interna, externa e estrutural (referentes a informações internas do sistema, representações do ambiente externo no sistema e as informações de diálogo entre estes meios, respectivamente); e propõe validações de diferentes itens de especificação para cada uma delas.

A sequência sugerida para a especificação dos casos de uso coloca a descrição do fluxo normal das atividades em um diagrama de sequência como o primeiro passo; o segundo é a identificação dos fluxos alternativos; seguido pela criação do diagrama de atividades com as novas informações obtidas; o quarto passo propõe a verificação do nível de cobertura até que ele seja suficiente, caso contrário a ação deve ser repetida desde o início.

A validação feita nos casos de uso selecionados segue o modelo de inspeções e *walkthroughs*, onde dois níveis distintos são sugeridos:

1. Analistas e testadores fazem uma pré-inspeção no modelo de casos de uso, modelo de classes e seu domínio para detectar violações a UML e ao *SCORES*. Cenários de negócio são construídos para cada caso de uso de acordo com suas pré-condições e o diagrama de atividades correspondente. O próximo passo é construir constelações de objetos, baseado no modelo de classes, e combinar os dois nos chamados cenários do mundo real.

2. No segundo nível *walkthroughs* são feitos cenários de negócio por especialistas de domínio e usuários, utilizando as constelações de objetos e pré-condições dos casos de uso. O objetivo é garantir que todas as informações necessárias estão presentes e as descrições das atividades estão corretas e completas. A próxima etapa prevê a validação dos diagramas de classe através da análise das descrições das ações comparada as constelações de objetos. Por fim, um pequeno *checklist* de verificação é aplicado para auxiliar na garantia do nível de completude, corretude e consistência dos modelos.

- Análise, Refinamento e Verificação

Na etapa de análise do *SCORES*, duas atividades importantes para a modelagem do domínio e validação são executadas: refinamento e verificação, tanto do modelo de caso de uso quanto do modelo de classe.

Na modelagem de domínio, um modelo de classe preliminar é construído, contendo apenas as classes mais importantes, as operações raiz e alguns atributos e relacionamentos. O processo de refinamento deve ser realizado a fim de encontrar atualizações a serem realizadas no modelo de classe e deve ser repetido até que se perceba que foi atingida a estabilidade do modelo. O resultado é uma primeira versão do modelo de classe passível de utilização no desenvolvimento.

Após o refinamento, analistas e testadores realizam uma análise mais formal e detalhada no modelo de classes e no modelo de caso de uso, verificando a consistência entre eles. Ao final, o modelo de caso de uso pode ser considerado verificado e o modelo de classes consistente e sem ambiguidades.

2.5 Estudos comparativos entre as técnicas apresentadas

Com base na diversidade de técnicas apresentadas uma análise torna-se necessária a fim de identificar aquelas que poderão contribuir de maneira mais eficaz à proposta sugerida neste trabalho.

Um exemplo deste tipo de análise encontrado na literatura é o estudo comparativo entre algumas técnicas de leitura desenvolvidas por Shull *et. al.* [2001]. Eles propuseram uma árvore que representa as diversas famílias de técnicas desenvolvidas, conforme pode ser verificado na figura 4.

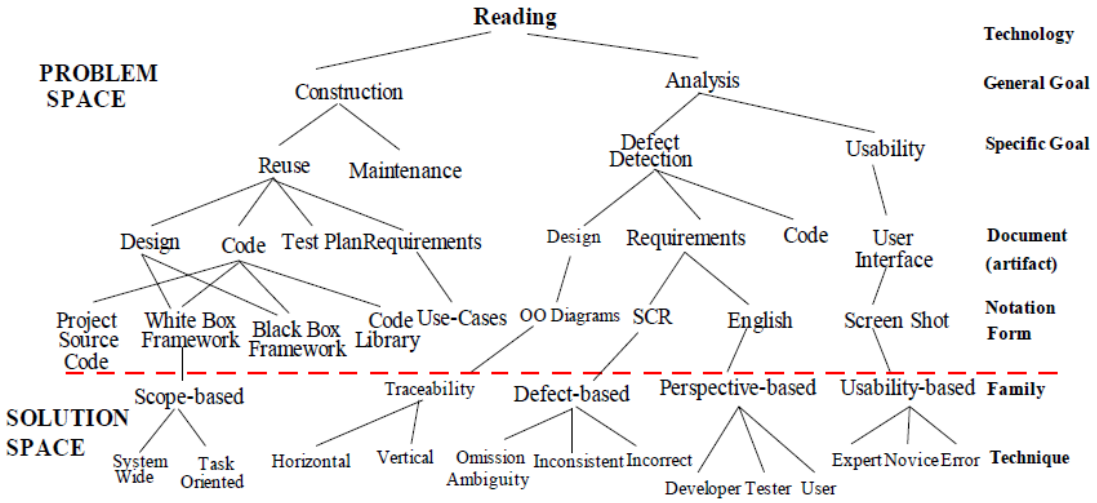


Figura 4 – Famílias de técnicas de leitura.

Fonte: Shull *et al*., 2001.

A parte superior da árvore (acima da linha pontilhada) modela os problemas abordados por cada técnica. Cada nível define a característica de desenvolvimento de software descrita na coluna mais à direita. A parte inferior da árvore indica possíveis soluções para os problemas indicados nos respectivos caminhos da árvore.

Essa árvore indica a diversidade de técnicas de leitura existentes e evidencia a importância da escolha da técnica mais adequada para cada caso, que deve ser feita em função da avaliação dos artefatos, objetivos e características de cada técnica.

Esta seção apresenta alguns estudos referentes às práticas relevantes para a técnica de inspeção e seus resultados. No entanto, é preciso salientar que ainda não existe uma resposta

conclusiva sobre este assunto. Os resultados apresentados apenas indicam a técnica mais eficiente para um determinado cenário empírico utilizado nos experimentos [Berling, 2003].

Nos próximos itens serão expostos os principais resultados de estudos referentes a técnicas que apresentaram maior relevância no contexto da proposta deste projeto:

- PBR

Segundo estudos realizados por Mafra, a técnica PBR é mais efetiva na detecção de defeitos em documentos de requisitos do que a técnica Checklist. Além disso, a PBR proporciona um ganho no conhecimento do problema, permitindo a identificação de defeitos de maior severidade para o sistema. É aconselhada para revisões curtas (2h), realizadas por equipes pequenas (máximo 3 revisores) [Mafra, 2006].

- OORT

Uma série de estudos sobre OORT foi desenvolvida por estudantes e profissionais de diversas universidades e empresas de diferentes países, por exemplo, os artigos já citados de Travassos e Shull e o artigo de Melo [Melo, 2001]. Um consenso entre eles é a efetividade da técnica. Destacam-se também o alto nível de detecção de defeitos, sua aplicabilidade em vários ambientes e a inexistência de diferença relevante nos resultados em função do nível de experiência do inspetor no domínio do problema.

Como possível forma de aplicação da verificação vertical proposta pela OORT, foram analisadas duas técnicas desenvolvidas por Georg Kösters e Martin Glinz que serão comparadas de forma simplificada a seguir.

- Kösters x Glinz

As duas propostas de verificação de modelos de classes e casos de uso descritos nos artigos de Kösters *et al.* (2001) e Glinz (2000) propõem a verificação em duas diferentes perspectivas de informação do sistema: estrutural e funcional. Entretanto, os autores utilizam formas bem distintas para atingir seus objetivos.

Glinz concentra sua proposta na verificação da consistência dos modelos, enquanto Kösters adota a verificação e validação completa dos itens como seu objetivo final. Além disso, a proposta de Glinz considera as informações dos dois modelos como complementares

para a especificação de requisitos do sistema, ou seja, encontram-se divididas nos dois modelos, enquanto Kösters foca na necessidade de informações completas em cada um dos modelos.

Conforme sugerido, o modelo de Kösters é o mais completo, porém mais complexo e custoso que o modelo de Glinz. O modelo de Kösters prevê a criação de diversos artefatos intermediários como cenários de negócios, diagramas de atividades, escopos de classe, constelações de objetos, além de envolver usuários e especialistas de domínio, além dos analistas e testadores, na aplicação desta técnica. A proposta de Glinz, por outro lado, preza por uma verificação mais eficiente, contando com o apoio ferramental de alguns programas.

2.6 Conceitos complementares

Alguns dos principais conceitos da análise de sistemas necessários para o desenvolvimento da técnica serão brevemente descritos nesta subseção a fim de assegurar uma correta correlação entre estes e sua forma adequada de aplicação.

- Taxonomia dos Defeitos

A taxonomia de defeitos adotada foi apresentada no Capítulo 2 e será mantida por representar os principais defeitos em um nível de abstração relevante para a técnica de inspeção de softwares desenvolvida.

A única alteração no modelo original é a retirada do tipo “Fato Incorreto”, já que sua definição (algo que está no sistema e que vai de encontro a algo especificado) implica em um alto conhecimento do domínio, o que não é um pré-requisito desta técnica. Os tipos de defeitos considerados são apresentados na Tabela 2.

Tabela 2 – Tipos de defeitos utilizados na técnica desenvolvida.

Tipo de Defeito	Definição
Omissão	Falta de representação ou definição de informação relevante para o projeto.
Ambiguidade	Informação com falta de clareza na especificação, causando problemas na sua interpretação.
Inconsistência	Informação definida de formas diferentes ao longo do projeto.

Informação Estranha	Informação especificada não utilizada em nenhum artefato de desenvolvimento do software.

[Fonte própria]

- Matriz de Interações (Matriz CRUD)

A Matriz de Interações é utilizada para representar e analisar as relações entre funcionalidades (atividades) e entidades (tipos de objetos de dados) de um sistema de informação.

Sua construção baseia-se em dois eixos, um com as funcionalidades e outro com as entidades. As células de interseção denotam o tipo de interação existente, ou seja: mostram que entidade será afetada pela execução de uma determinada funcionalidade, e explicita as propriedades CRUD para tal interseção. Estas podem ser *Create* (inclusão), *Read* (leitura), *Update* (atualização) e *Delete* (exclusão) [Pressman, 2002].

Analistas podem usar este artefato para verificar a consistência entre as definições funcionais e de dados inerentes à solução que estão desenvolvendo. No contexto da técnica desenvolvida, o principal objetivo é a análise de rastreabilidade. Ela é feita através de reflexões acerca dos modelos de dados e do negócio e podem levar a uma reestruturação das informações no projeto. Para isto algumas questões podem ser utilizadas:

- **Todas as entidades estão associadas à no mínimo uma atividade do tipo Create?** Caso contrário deve-se refletir sobre a forma de absorção destes dados pelo sistema.

- **Todas as entidades estão associadas à no mínimo uma atividade dos tipos Delete e Update?** Caso contrário deve-se refletir se estas funcionalidades são realmente irrelevantes para o negócio.

- **Todas as entidades associadas a atividades dos tipos Create e Delete possuem no mínimo uma atividade do tipo Update?** Caso contrário, a usabilidade do sistema estará comprometida, mas a funcionalidade de certa forma já está oferecida ao usuário. Deve-se refletir sobre a disponibilização direta desta funcionalidade no sistema.

- **As atividades interagem com no mínimo uma entidade?** Caso contrário, isso pode indicar que a decomposição funcional está em um nível de granularidade desbalanceado, a análise não foi concluída ou o sistema apenas realiza cálculos.

- **As entidades interagem com no mínimo uma atividade?** Caso contrário, isso pode indicar que a lista de funcionalidades não está completa ou que existem dados cuja armazenagem não é feita de forma informatizada.

- **As atividades relacionadas à entidade atendem todas as necessidades do negócio referentes a ela?** Caso contrário considere a necessidade da representação de uma atividade mais de uma vez na mesma entidade, relacionando-se a funcionalidades diferentes.

- Casos de Uso [Ivar Jacobson, 1992]

Para formalizar a definição de casos de uso e orientar inspetores quanto às informações esperadas, adotou-se como documento esperado aquele de acordo com a definição formal proposta por Ivar Jacobson, um dos criadores deste conceito.

Segundo sua primeira definição, um caso de uso é um documento narrativo que descreve a sequência de eventos de um ator (um agente externo), que usa um sistema para completar um processo [Jacobson, 1992].

Tais sequências de ações são definidas como transações. Em outras palavras, em uma transação o ator executa alguma ação que representa uma entrada para o sistema, que então reage, processando essa entrada de dados e retornando um resultado para o ator.

Com isto, uma transação de um caso de uso deve ser capaz de reportar três tipos de dados a seu leitor:

- **Ação:** dado ou ação executada pelo ator e que compõe a entrada do sistema.
- **Verificação:** condições ou restrições capazes de definir se o sistema se manterá no fluxo ou seguirá por um de seus caminhos alternativos.
- **Resposta:** resultado esperado pelo ator em função da ação executada. Uma resposta pode ser interna (atualização da memória do sistema) ou externa (saída de informações do sistema para um elemento externo - ator).

3 TÉCNICA DESENVOLVIDA

No capítulo anterior, foram apresentadas diversas técnicas desenvolvidas para auxiliar no processo de inspeção de software. Cada uma possui pontos positivos e negativos, ficando a cargo do responsável pela inspeção decidir qual(is) técnica(s) utilizar, de acordo com o contexto do sistema e com a qualidade final do produto que se deseja atingir.

O objetivo central deste trabalho é construir uma técnica de inspeção a partir de um subconjunto das técnicas apresentadas no capítulo 2, às quais foram adicionadas algumas melhorias.

3.1 Definição da Técnica

O objetivo deste projeto é propor uma técnica de inspeção de software que contemple os principais pontos de algumas propostas existentes, baseando-se nas informações obtidas nas seções anteriores.

Pretende-se com esta técnica verificar os artefatos mais importantes das atividades referentes ao levantamento de requisitos e análise do sistema de forma prática e eficaz, tendo em vista que inspeção não faz parte da cultura e dos processos organizacionais da maioria das empresas do ramo de tecnologia da informação.

O desenvolvimento da técnica foi baseado em fases distintas do ciclo de desenvolvimento do software utilizando como modelo o processo RUP[9] e seus artefatos, embora isso possa ser transportado para outros processos que gerem documentos similares considerando apenas a necessidade de algumas adaptações. A sua aplicação foi divida em duas etapas, para que fosse possível contemplar as fases de levantamento e especificação de requisitos:

- Verificação do documento de visão do negócio e construção de artefatos iniciais da análise.
- Verificação dos casos de uso em relação ao modelo de classes.

As etapas serão detalhadas nas próximas seções.

[9] Rational Unified Process (Processo Unificado Rational) é um processo proprietário de engenharia de software criado pela Rational Software Corporation. Fornece técnicas a serem seguidas pelos membros da equipe de desenvolvimento de software com o objetivo de aumentar a sua produtividade no processo de desenvolvimento.O RUP usa a abordagem da orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (*Unified Modeling Language*) para ilustrar os processos em ação.

3.1.1 Verificação do documento de visão do negócio e construção de artefatos iniciais da análise

Na atividade de levantamento de requisitos do ciclo de desenvolvimento de softwares, os objetivos principais são entender o contexto do negócio do cliente e identificar suas principais necessidades e expectativas em relação ao produto que será desenvolvido. Normalmente, documentos são construídos pela equipe de levantamento de requisitos para registrar as informações obtidas, sendo posteriormente utilizados como insumo para análise, *design* e construção do sistema.

Muitos estudos de caso foram realizados utilizando técnicas de inspeção em documentos dessa natureza. Contudo, a maioria desses estudos pressupõe que o contexto do negócio e os requisitos do sistema estão altamente detalhados, o que não costuma ser encontrado nos projetos reais em função de fatores como limitação de tempo e falta de clareza nas necessidades do cliente. Com o intuito de manter o viés prático da técnica, ou seja, tornar sua aplicação viável em cenários usualmente encontrados no mercado, foi realizada uma análise para identificar os artefatos mais importantes produzidos pela atividade de levantamento de requisitos e o nível de detalhamento normalmente encontrado nesses artefatos.

O documento de visão é um dos artefatos gerados na atividade de levantamento de requisitos e possui grande importância no entendimento inicial do contexto em que o sistema está inserido e das necessidades fundamentais que levaram à sua concepção. Por ser redigido em linguagem natural e não possuir um padrão seja para o nível de detalhamento da informação, seja para o formato de sua apresentação, a ocorrência de defeitos nesse documento é frequente, causando dificuldades no entendimento inicial do projeto. Por esses motivos, o documento de visão foi escolhido como um dos alvos de aplicação da técnica proposta, na expectativa de melhorar a qualidade da informação inicial do sistema.

Existem diversas técnicas de inspeção aplicáveis ao documento de visão, sendo *Ad hoc*, *Checklist* e PBR as mais utilizadas pelas empresas do ramo. No comparativo entre as técnicas (ver seção 3.2.1) foram identificadas algumas vantagens da técnica PBR em relação às demais: (a) maior efetividade na descoberta de defeitos; (b) proporciona um melhor entendimento do contexto da aplicação a ser desenvolvida; (c) redução do tempo de inspeção

e garantia de uma boa cobertura do documento inspecionado em cenários bem elaborados e com a escolha certa das perspectivas. Em função dessas vantagens, a técnica PBR foi escolhida para a realização da parte de verificação do documento de visão produzido na atividade de levantamento de requisitos.

Na atividade de análise do sistema, normalmente é construído um modelo de classes inicial baseado no documento de visão, contendo as entidades e atributos mais importantes do sistema e uma lista de casos de uso preliminar com as funcionalidades extraídas do documento. Por isso, a ideia inicial foi desenvolver uma técnica onde a primeira etapa permitisse a verificação da consistência entre o documento de visão e o modelo inicial de classes e casos de uso. Como a técnica de leitura selecionada foi a PBR, e esta já inclui necessariamente a construção de um modelo que abstraia o documento inspecionado, levando em consideração aspectos relevantes para cada perspectiva definida, a atividade de geração do modelo inicial de classes e a lista de casos de uso preliminar foram incluídas nas instruções dos cenários definidos. Ao indicar como os artefatos devem ser gerados, garante-se a consistência deles em relação ao documento inspecionado.

Definiram-se, então, dois cenários PBR de verificação: (a) um cenário PBR na perspectiva do projetista, com o intuito de construir o modelo inicial de classes e verificar a consistência do documento de visão em relação às classes, atributos e relacionamentos; (b) um cenário PBR na perspectiva do usuário, com o objetivo de listar os casos de usos iniciais do projeto e verificar a consistência do documento de visão em relação às funcionalidades e participantes (atores, sistemas externos e *hardwares*). As inconsistências encontradas em ambas as verificações são listadas e classificadas de acordo com a taxonomia descrita no próximo item.

Os cenários desenvolvidos para auxiliar os inspetores na aplicação da técnica podem ser encontrados ao final desta seção no item 3.3, Cenário da OO-PBR para a perspectiva do Projetista e Cenário PBR para a perspectiva do Usuário.

Ao longo da aplicação dos cenários, reflexões são propostas para analisar o documento de visão utilizado na verificação. Deseja-se com isso prover ao inspetor uma visão técnica sobre o documento utilizado, já que de suas informações depende o nível de qualidade dos modelos gerados nesta etapa.

O último ponto desta etapa propõe uma análise com base na Matriz CRUD. Todas as entidades e atividades identificadas nos cenários PBR devem ser dispostas nesta matriz, que tem por objetivo identificar discrepâncias no fluxo de dados da aplicação. Esta verificação torna-se relevante para a técnica de inspeção desenvolvida por sua baixa complexidade, já que pode ser aplicada de forma rápida por inspetores independentemente de seu grau de experiência. A Matriz CRUD possui também alto teor de detecção de defeitos considerados graves para o desenvolvimento de uma aplicação, como a falta de uma entidade ou de atividade relevante para o contexto do sistema.

Para auxiliar na aplicação da técnica, é indicada a utilização de um documento onde serão mantidas as informações da inspeção. Este documento foi chamado de **Documento de Análise Inicial** (ver *template* no Anexo I) e ao final da execução desta etapa de inspeção deve conter dois importantes artefatos da fase de análise do sistema: o modelo inicial de classes e a lista de casos de uso iniciais do sistema, além da identificação dos defeitos encontrados no documento de visão.

3.1.2 Verificação dos casos de uso em relação ao modelo de classes

Com a maior qualidade do documento de visão e dos artefatos iniciais do sistema, produto da etapa anterior, garante-se o material necessário para a criação de artefatos mais detalhados, como os casos de uso e um refinamento do modelo de classes, adotados como a principal forma de representação dos modelos de software. Estes modelos por sua vez serão analisados em uma verificação de artefatos UML de alto nível.

Esta etapa possui o objetivo de garantir que os artefatos de diferentes níveis de abstração (estático e dinâmico) são consistentes entre si e estão corretos. A técnica utilizada para esta etapa foi baseada em [Glinz, 2000], sobre a validação dos modelos de classe e de caso de uso, chamado de cenário.

Para se adequar as características de praticidade e agilidade previstas para a proposta desenvolvida e ao objetivo de verificação de modelos desta etapa, apenas o esquema de referências cruzadas proposto por Glinz será utilizado. Destacam-se como principais razões para sua utilização a aplicabilidade em diversos padrões de casos de uso, a experiência no domínio e na aplicação resultante do método e a integração entre os dois principais modelos

utilizados atualmente na análise de softwares, tanto no ambiente acadêmico quanto empresarial: casos de uso e classes.

O esquema adotado pode ser definido como uma derivação do esquema simples definido por Glinz por utilizar a inclusão de referências no modelo de caso de uso para as operações definidas no modelo de classes, porém sem a inclusão de parâmetros ou a formação de pseudocódigos. Outro diferencial é a extensão das referências a entidades, atributos e relacionamentos definidos no modelo de classes e estados definidos no diagrama de estados.

Casos onde a existência de uma referência seja identificada como necessária, porém impraticável por um defeito no modelo verificado, devem ser classificados, de acordo com a taxonomia descrita anteriormente, e listados no **Documento de Análise Final** (ver *template* no anexo II), planejado para armazenar todos os defeitos identificados nesta verificação.

Um guia detalhado da verificação entre os modelos encontra-se listado no item 3.4, Verificação de Consistência entre Casos de Uso e Modelos de Classe.

Ao longo de toda esta etapa é sugerido ao inspetor uma análise dos modelos de caso de uso utilizados na verificação em relação à definição de casos de uso proposto por Ivar Jacobson, definido anteriormente. O objetivo desta análise não é um produto físico, e sim proporcionar ao inspetor uma visão a respeito da abrangência dos casos de uso utilizados por sua relação direta com a eficácia desta etapa.

Todos os defeitos observados ao longo da aplicação desta técnica de inspeção deverão ser compilados em suas respectivas listas de discrepâncias. Não faz parte do escopo deste projeto resolver as inconsistências identificadas, apenas destacá-las para que os responsáveis possam empregar a forma mais adequada ao contexto do sistema e orçamento e metodologia da empresa para resolvê-los.

Com isso, o resultado da aplicação desta etapa visa garantir que os modelos verificados contenham, de forma clara, todas as informações necessárias para a representação do ambiente do sistema a ser desenvolvido, a fim de gerar para a próxima etapa do ciclo de vida de desenvolvimento de softwares (Projeto ou Criação) modelos completos, corretos e consistentes, minimizando a transposição de erros de uma etapa para a outra do projeto.

3.2 Resumo da Técnica

A partir de todas as informações detalhadas anteriormente podemos esquematizar a técnica de inspeção de softwares proposta no seguinte esquema:

PARTE 1) Verificação do documento de visão do negócio e construção de artefatos iniciais da análise:

a. Aplicação de cenário PBR – Projetista
b. Aplicação de cenário PBR – Usuário
c. Validação CRUD

PARTE 2) Verificação da consistência dos casos de uso em relação ao modelo de classes:

a. Inclusão de referências nos casos de uso
b. Análise da consistência dos casos de uso

3.3 Cenários desenvolvidos

Os cenários desenvolvidos tiveram como base o cenário de [Mafra, 2006], perspectiva projetista, e [Shull, 1998], perspectiva usuário. Ambos sofreram a remodelação de algumas etapas para garantir a praticidade e agilidade requeridas para a técnica proposta e absorver apropriadamente qualquer nível de conhecimento utilizado para descrever o documento de visão, que serve como insumo desta etapa. Novos questionamentos também foram inseridos para permitir a identificação de um maior número de inconsistências neste documento.

- Cenário da OO-PBR para a perspectiva do Projetista

Introdução
Assuma que você é um engenheiro de software. O objetivo principal do engenheiro de software é construir os artefatos relacionados à fase de projeto do sistema. Ao executar as instruções deste cenário, você será orientado a construir um modelo de classes de alto nível orientado a objetos, ou seja, abstrair classes, seus relacionamentos e suas responsabilidades em um modelo de acordo com o documento de visão.
Instruções
As considerações abaixo devem ser lembradas durante toda a inspeção:

- O seu objetivo principal é gerar um modelo de classes de alto nível e uma lista inicial dos casos de uso do sistema.
- É provável que algumas dúvidas e discrepâncias surjam no processo de construção do modelo e de identificação dos casos de uso. Não tente solucioná-las neste momento. Sugerimos anotá-las no **Formulário de Discrepâncias** presente no **Documento de Análise Inicial**.

Passo 1 – Classes e Atributos

a) Faça uma leitura de todo o documento de visão sem se preocupar em identificar defeitos, apenas para se contextualizar sobre os assuntos abordados.

b) Leia o documento novamente, agora com a intenção de extrair potenciais nomes de **classes** e **atributos**. Candidatos a atributos e classes encontram-se como substantivos e seus complementos ao longo do texto. Identifique aqueles que representam conceitos que você considere relevantes para o contexto do negócio tratado no sistema.

c) Preencha o **Formulário de Classes** do **Documento de Análise Inicial**. Verifique, para cada nome identificado no item *a*, se este representa uma classe ou um atributo. Caso o nome represente um atributo, indique a qual classe ele pertence.

d) Para cada atributo extraído, verifique se foi especificada uma unidade de valor. Ex.: "(...) velocidade (atributo) medida em metros por segundo (unidade de valor) (...)". Em função da unidade de valor especificada, identifique seu provável tipo.

e) Caso algum atributo represente uma enumeração de valores, como por exemplo, os vários estados de um objeto, extraia todos os possíveis valores descritos no documento, e descreva-os no **Formulário de Classes**.

f)Represente graficamente as classes e seus atributos no **Formulário de Classes**.

g)Responda as questões a seguir referentes à verificação do documento de visão e anote as discrepâncias encontradas no **Formulário de Discrepâncias**.

P1.1 As classes extraídas representam conceitos envolvidos no domínio da aplicação? Caso negativo, conceitos irrelevantes podem estar especificados no documento de visão. Reporte uma *informação estranha*.

P1.2 Todas as definições importantes para o domínio da aplicação estão detalhadas adequadamente em uma seção de glossário ou outra seção similar? Caso negativo reporte uma *omissão*.

P1.3 Existem sinônimos no documento, mais de um nome representando um mesmo conceito? Caso positivo, reporte uma *ambiguidade*.

P1.4 A descrição de algum conceito contradiz outro(s)? Existem homônimos, ou seja, um termo representando mais de um conceito? Caso positivo, reporte uma *inconsistência*.

Passo 2 - Relacionamentos e Multiplicidades

a)Leia o documento de visão de forma a extrair **relacionamentos** entre as classes. Possíveis relacionamentos candidatos vêm dos verbos de orações envolvendo os nomes das classes. Ex: "*O cliente possui no máximo um cartão fidelidade*".

b) Verifique a existência de **generalizações**. Termos como "*é um"* ou *"é um tipo de"* normalmente representam relacionamentos de generalização. Identifique classes que possuam algum atributo em comum e verifique se essas classes podem formar uma relação de generalização.

c) Represente os relacionamentos no modelo. Extraia as quantidades descritas para determinar as suas **multiplicidades**. Exemplo: "*O cliente possui no máximo um cartão fidelidade*".

d) Verifique se alguma classe não está relacionada a nenhuma outra. Caso positivo, certifique se esta é de fato uma classe ou se seus atributos e relacionamentos podem ser incluídos em outra.

e)Responda as questões abaixo referentes à verificação do documento de visão e anote as

discrepâncias encontradas no **Formulário de Discrepâncias**.

P2.1 As informações presentes no documento de visão permitem a você estipular a multiplicidade dos relacionamentos envolvidos? Caso contrário, reporte uma *omissão*.

P2.2 Existe algum relacionamento onde a cardinalidade indicada é diferente em momentos distintos do documento? Caso positivo, reporte uma *inconsistência*.

P2.3 Na sua opinião, os relacionamentos entre as classes estão coerentes em relação ao domínio da aplicação? Caso negativo identifique a informação que gerou o defeito e reporte uma *informação estranha*.

- Cenário PBR para a perspectiva do Usuário

Introdução
Assuma que você é um usuário do sistema. Por este ponto de vista, a sua expectativa é encontrar as funcionalidades necessárias para o sistema refletidas no documento de visão. Para mapear as funcionalidades você deverá: o Identificar os participantes (usuários ou sistemas externos que interagem com o sistema, enviando e/ou recebendo informações). o Criar uma lista de casos de uso, que represente todas as possibilidades de utilização do sistema indicadas no documento de visão.
Instruções
As considerações abaixo devem ser lembradas durante toda a inspeção: o O seu objetivo principal é formular uma lista inicial dos casos de uso do sistema. o É provável que algumas dúvidas e discrepâncias surjam no processo de construção do modelo e de identificação dos casos de uso. Não tente solucioná-las neste momento. Sugerimos anotá-las no **Formulário de Discrepâncias**, presente no **Documento de Análise Inicial**.

Passo 1 - Identificação dos Participantes

a)Faça uma leitura de todo o documento de visão sem se preocupar em identificar defeitos, apenas para se contextualizar sobre os assuntos abordados.

b)Leia o documento de visão novamente de forma a identificar os participantes (sistemas externos, classes de usuários, hardwares) envolvidos com o sistema. Liste cada participante no **Formulário de Participante s**e adicione uma breve descrição sobre ele. Utilize as perguntas abaixo para auxiliar na identificação:

- Quais grupos de usuários utilizam o sistema para realizar tarefas?
- Quais grupos de usuários são necessários pelo sistema para executar suas funções? Estas funções podem ser principais ou secundárias.
-Quais são os sistemas externos que utilizam o sistema para realizar tarefas?
- Quais componentes ou hardwares são gerenciados pelo sistema?
- Quais são os grupos de usuários ou sistemas externos que enviam informação para o sistema?
- Quais são os grupos de usuários ou sistemas externos que recebem informação do sistema?

c) Identifique quais dos participantes são atores, ou seja, quais participantes representam uma classe de usuários. Algumas considerações que podem auxiliar na identificação:

- A diferença de um ator para um participante comum é que suas ações são não-determinísticas (ou seja, não possui um comportamento padrão).
- Atores representam classes de usuários, e não usuários individuais do sistema (por exemplo: Professor Fábio não é ator, é instância de Professor).
- Atores representam o conjunto de coisas externas que precisam trocar informação com o sistema.

d) Responda as questões abaixo referentes à verificação do documento de visão e anote as discrepâncias encontradas no Formulário de Discrepâncias.

P1.1 Múltiplos termos são utilizados para descrever o mesmo participante no documento? Caso positivo, identifique os termos e reporte uma *inconsistência.*

P1.2A descrição de como o sistema interage com o participante é inconsistente com a descrição do participante? Caso positivo, reporte uma *inconsistência*.

P1.3 Participantes importantes foram omitidos, isto é, o sistema necessita interagir com outro sistema ou *hardware* ou um tipo de usuário que não está descrito? Caso positivo, reporte uma *omissão*.

P1.4 Existe algum sistema externo ou classe de usuários descrita nos requisitos que não interaja em momento algum com o sistema? Caso positivo, reporte uma *informação estranha*.

Passo 2 - Identificação das Funcionalidades

a)Leia o documento de visão de forma a identificar as funcionalidades (toda ação ou comportamento do sistema que produz alguma saída de valor para algum participante ou para o próprio sistema). As funcionalidades que indicam fluxos de exceção também devem ser identificadas.

b)A lista construída no item anterior pode conter funcionalidades de baixo nível. Estas são fragmentos de uma função maior e devem ser agrupadas para formar casos de uso que representem uma tarefa de alto nível do sistema que será utilizada por um ator. Para realizar o agrupamento, siga as orientações abaixo:

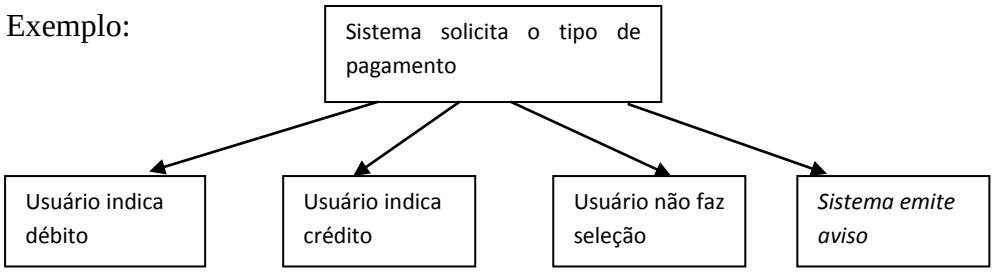
- Agrupe as funcionalidades que estão envolvidas em um mesmo contexto.
- Agrupe funcionalidades que lidam com um objetivo em comum para o sistema ou para um ator.
- Identifique fases de processamento: existe uma sequência ou ordem para a execução da funcionalidade? Agrupe funcionalidades que são consecutivas.

c) Os casos de uso identificados devem ser registrados no **Formulário de Casos de Uso**.

Adicione uma breve descrição sobre cada um deles.

d)Para cada agrupamento realizado, desenhe uma árvore contendo a ordem de execução das funções de baixo nível e registre no **Formulário de Casos de Uso – Fluxo de Execução**. Esta árvore deve ser construída de cima para baixo, onde as funções mais acima são executadas antes em relação às mais abaixo. Esta etapa auxiliará o seu entendimento sobre o fluxo de execução dos casos de uso identificados.

Exemplo:



e)Verifique se o nome de cada caso de uso está coerente com o seu propósito e faça alterações se necessário.

f) Por fim, relacione os atores aos casos de uso identificados preenchendo a coluna **Atores** do **Formulário de Casos de Uso**.

g) Responda as questões abaixo referentes à verificação do documento de visão e anote as discrepâncias encontradas no **Formulário de Discrepâncias**.

P2.1 Com base em seu conhecimento sobre o contexto do sistema e em tudo que foi lido no documento de visão, existe alguma funcionalidade de algum caso de uso que foi omitida? Caso positivo, reporte uma *omissão.*

P2.2 A descrição do fluxo de execução de alguma funcionalidade causa dúvida na interpretação e/ou possui termos vagos? Ex.: "*O sistema deve analisar periodicamente o CPF do cliente".* Caso positivo, reporte uma *ambiguidade.*

P3.3 Caso as funcionalidades possuam restrições, o documento descreve como essas restrições devem ser verificadas? Ex.: "*O sistema deve permitir que o gerente visualize as compras do cliente*" é descrito como o sistema pode verificar se a pessoa executando a

operação é um gerente? Caso negativo reporte uma *omissão.*

P3.4 Existem funções descritas no documento e que não fazem parte do escopo do sistema, ou seja, não devem ser implementadas na solução final? Ex.: "*Após emitir o recibo, o documento deve ser fotocopiado e enviado à diretoria*". Caso positivo, reporte uma *informação estranha*.

3.4 Guia de verificação entre Casos de Uso e Modelos de Classe

Uma série de instruções foi compilada para proporcionar ao inspetor um guia de verificação entre Casos de Uso e Modelos de Classe. Ele se baseia em alguns métodos de identificação de conceitos como classes, atributos e estados, utilizados nos cenários anteriores e no esquema de referências cruzadas proposto por Glinz. Este guia representa também um detalhamento em relação à proposta original de referências cruzadas, já que esta não menciona como identificar os itens que devem ser referenciados.

Introdução

Ao executar as instruções deste guia, você será orientado a realizar diversas marcações nos casos de uso que permitam suas referências às entidades, atributos, operações e restrições do modelo de classes; e estados e operações de guarda dos diagramas de estado.

Instruções

As considerações abaixo devem ser lembradas durante toda a inspeção:

- O seu objetivo principal é garantir a consistência do modelo de caso de uso em relação ao modelo de classes.
- É provável que algumas dúvidas e discrepâncias surjam no processo verificação. Não tente solucioná-las neste momento. Sugerimos anotá-las no **Formulário de Discrepâncias** presente no **Documento de Análise Inicial**.
- Adota-se como premissa desta técnica que uma vez mencionado o item não necessitará ser novamente referenciado ao longo do documento.

Inclusão de Referências no Modelo de Caso de Uso ao Modelo de Classes

1) Leia o modelo de caso de uso para compreender todas as funcionalidades envolvidas, a participação dos atores e os conceitos do ambiente do sistema presentes no documento.

2) Leia novamente o modelo de caso de uso incluindo marcações para os modelos de classe e diagrama de estado. As marcações devem seguir os padrões definidos a seguir para cada conceito do modelo de classes e do diagrama de estados.

- **Classe:** Identifique os substantivos mais relevantes que devem ser representados como entidades no modelo de classes e busque ali por seu correspondente, seja por uma ocorrência exata ou por sinônimos. Caso encontre, inclua uma marcação a respectiva entidade, conforme o modelo **↑Classe**.

- **Atributo:** Identifique os principais substantivos utilizados como característica de uma entidade ao longo do fluxo de dados, estes serão os atributos. Relacione-os também conforme o modelo **↑Classe.atributo**.

- **Operação:** Os principais verbos e seus complementos representarão as operações do sistema, relacione-as conforme o modelo **↑Classe.operacao()** . Não se preocupe com os parâmetros, como não serão desenvolvidos pseudocódigos esta verificação não será feita.

- **Relacionamentos:** Caso uma operação ou uma característica envolva dois objetos de classes diferentes inclua a referencia a um relacionamento, conforme o modelo **Classe1→Classe2**.

- **Enumeração:** Substantivos identificados como atributos e a uma enumeração de valores devem ser associados a uma enumeração no modelo de classes. Faça a referência no caso de uso de forma semelhante a uma entidade, conforme o modelo **↑Enumeração**.

- **Estados:** Inclua referências aos estados de um diagrama de estados caso identifique

trechos semelhantes a: X está “no período Y”, “na fase Y”, “pode ser do tipo X ou Y”. Lembre-se de que um atributo com estados ocorre quando ele assume valores diferentes de acordo com restrições pré-estabelecidas e não pode assumir mais de um desses valores ao mesmo tempo. As referências devem ocorrer no formato **↓DiagramaDeEstados.estado**.

- **Operação de Guarda:** Os principais verbos e seus complementos que desencadeiem mudanças no estado de um atributo devem ser referenciados como **↓DiagramaDeEstados.operacaoDeGuarda()**. Não se preocupe com os parâmetros, como não serão desenvolvidos pseudocódigos esta verificação não será feita.

- **Restrição:** Todas as demais restrições a dados presentes nos casos de uso, como citações a cardinalidades ou tipos de dados, devem ser mencionadas no modelo de classes como uma restrição, que pode estar representada de forma técnica, como em OCL (Object Constraint Language), ou de forma mais simples, como uma nota. Inclua a restrição no formato **↑Classe.atributo.tipoDeRestrição:valor**.

4 EXEMPLO DE APLICAÇÃO DA TÉCNICA

Neste capítulo será apresentado um resumo contendo as partes mais relevantes da aplicação da técnica desenvolvida. A documentação do exemplo da aplicação da técnica (documento de visão, caso de uso, modelo de classe e diagrama de estado) encontra-se na seção de anexos (Anexo III – Documento de Visão, Anexo IV – Caso de Uso, Anexo V – Diagrama de Estado e Anexo VI – Modelo de Classes).

4.1 Etapa 1 - Verificação do Documento de Visão do Negócio e Construção de Artefatos Iniciais da Análise

A primeira etapa da técnica é dividida em aplicação da técnica PBR visão Projetista e da técnica PBR visão Usuário, cujos cenários elaborados foram apresentados na seção 3.3.

Nos itens a seguir serão apresentados fragmentos relacionados a cada instrução do cenário PBR desenvolvido, indicando o que foi selecionado do documento de visão e o resultado obtido. O documento de visão completo encontra-se no Anexo III e o *template* do documento de análise inicial no Anexo I.

Os documentos utilizados para este exemplo de aplicação da técnica compõem um projeto de automatização de ferramentas utilizadas na gestão do ensino à distância em universidades de ensino superior, em especial da fase de automatização do processo de seleção de coordenadores de polo, responsáveis por administrar um espaço físico à disposição dos alunos matriculados neste tipo de graduação.

4.1.1 PBR – Projetista

O cenário PBR – Projetista foi dividido em dois passos, apresentados a seguir.

- Identificação de Classes e Atributos

No passo 1, são identificadas classes e atributos do sistema. Além disso, as unidades e faixas de valores dos atributos também são identificadas.

a) Leitura do documento de visão
   Leitura de todo o documento de visão sem se preocupar em identificar defeitos.

Esta instrução foi inserida no cenário devido a uma dificuldade encontrada durante experiências prévias do cenário. Ao ler o documento de visão com o intuito de identificar conceitos relacionados ao modelo de classes notou-se que a atenção para entender o contexto do sistema já havia se perdido. Nenhum produto é gerado neste momento, a intenção é proporcionar maior entendimento ao inspetor acerca do contexto do sistema, incitando a leitura do documento para auxiliar nas instruções seguintes.

b) Identificação de Classes e Atributos

Ler o documento com a intenção de extrair nomes de classes e atributos.

O fragmento a seguir demonstra a identificação de uma classe candidata e de seus possíveis atributos no documento de visão:

> (...) memorial descritivo contendo uma breve descrição de no máximo 1580 caracteres (...)
>
> (...)memorial (medida para evitar eventuais favorecimentos) e registra a nota que pode variar de 0 a 10 e conter até 2 casas decimais, e uma observação opcional a respeito da nota indicada (...)

c) Preenchimento do Formulário de Classes

Após identificar as classes e atributos candidatos, uma reflexão deve ser realizada com intuito de decidir se o termo identificado é um atributo ou uma classe.

Com a definição das classes e atributos, o formulário de classes deve ser preenchido, conforme demonstrado a seguir:

CLASSE:	Memorial Descritivo				
ATRIBUTOS					
Nome	Descrição	**Tipo**		**Valores**	
Nome	Nota	**Tipo**		**Valores**	
Nome	Observação	**Tipo**		**Valores**	

d) Identificação de Unidades de Valor

É normal que a unidade de valor de alguns atributos seja descrito junto aos próprios atributos, podendo seu tipo ser identificado.

Alguns tipos podem ser identificados de maneira indireta, através do contexto onde o atributo está inserido no texto, como mostra o fragmento a seguir:

> (...) memorial descritivo contendo uma breve descrição de no máximo 1580 caracteres (...)
>
> (...)memorial (medida para evitar eventuais favorecimentos) e registra a nota que pode variar de 0 a 10 e conter até 2 casas decimais, e uma observação opcional a respeito da nota indicada (...)

No fragmento, apesar de não estar explicitado o tipo do atributo 'descrição', a palavra 'caracteres' já indica que o atributo é uma cadeia de caracteres.

CLASSE:	Memorial Descritivo				
ATRIBUTOS					
Nome	Descrição	**Tipo**	**String**	**Valores**	
Nome	Nota	**Tipo**	**Real**	**Valores**	
Nome	Observação	**Tipo**		**Valores**	

e) Identificação de Faixa de Valores

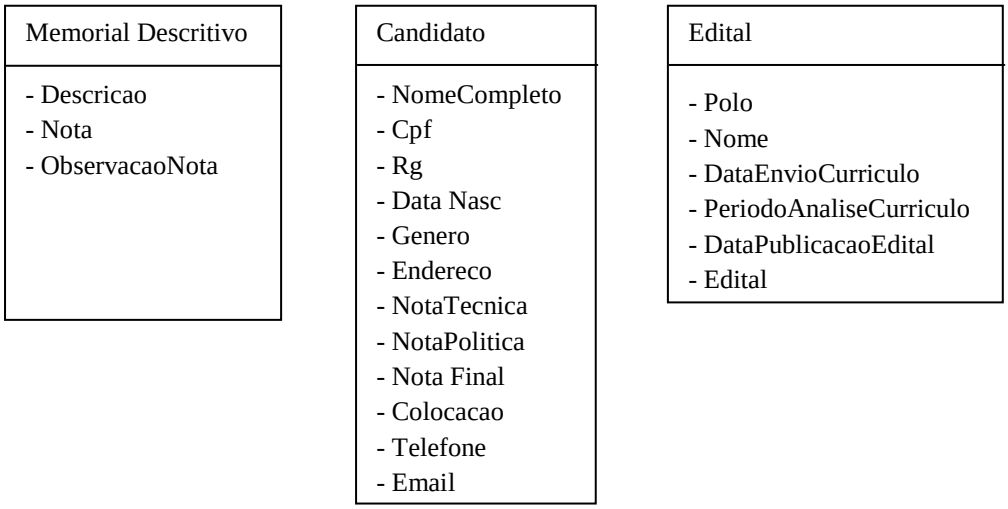
A faixa de valores não faz parte do nível de detalhamento normalmente encontrado no documento de visão. Contudo, caso esteja especificado, é interessante armazenar esta informação. O fragmento a seguir possui um exemplo:

> (...) memorial descritivo contendo uma breve descrição de no máximo 1580 caracteres (...)
>
> (...) memorial (medida para evitar eventuais favorecimentos) e registra a nota que pode variar de 0 a 10 e conter até 2 casas decimais, e uma observação opcional a respeito da nota indicada (...)

Esta informação também deve ser adicionada no formulário de classes:

CLASSE:	Memorial Descritivo				
ATRIBUTOS					
Nome	Descrição	**Tipo**	String	**Valores**	
Nome	Nota	**Tipo**	Float	**Valores**	**0 a 10,00**
Nome	Observação	**Tipo**		**Valores**	

f) Representação gráfica dos Elementos



...

g) Responder as questões de verificação de defeitos.

P1.1 A pergunta faz o inspetor refletir sobre as classes que ele extraiu do documento de visão, pois informações irrelevantes para o contexto do sistema podem estar sendo especificadas, o que atrapalha no entendimento correto do contexto. No exemplo de aplicação proposto, foram identificados alguns atributos da classe "Candidato" que fazem parte do cadastro, mas não são úteis para nenhuma funcionalidade do sistema, e, portanto podem ser removidas.

DESCREPÂNCIAS		
N°	Tipo	Descrição
1	Informação Estranha	As informações “RG”, “Gênero” e “Endereço” não são utilizadas no processo seletivo em momento algum e portanto não fazem parte do contexto do sistema.

P1.2 *É importante que todos os conceitos relevantes estejam definidos no glossário do documento de visão para que este seja uma fonte comum de consulta aos termos oriundos do negócio em que o sistema está inserido. Caso haja conceitos importantes que não estão no glossário, uma omissão deve ser reportada para que o autor do documento possa adicionar uma definição deste conceito no glossário.*

Uma discrepância relacionada a isso foi encontrada no exemplo, como mostra a tabela a seguir.

DISCREPÂNCIAS		
N°	Tipo	Descrição
2	Omissão	Os termos “memorial descritivo”, “documentação comprobatória” e “edital” não estão detalhados suficientemente para seu correto entendimento no contexto da aplicação.

P1.3 *É comum no documento de visão os termos utilizados ainda não estarem bem definidos e um conceito ser retratado através de termos diferentes, o que causa dúvidas em qual termo deve ser utilizado no resto do sistema.*

Um exemplo deste defeito do tipo ambiguidade está descrito a seguir, conforme discrepância encontrada.

DISCREPÂNCIAS		
N°	Tipo	Descrição
3	Ambiguidade	Os termos “IES” e “universidade”, “local de apoio” e “polo”, representam sinônimos e estão sendo utilizados para referenciar os mesmos conceitos.

P1.4 Outros problemas frequentes no documento de visão são a contradição entre conceitos apresentados e termo que representa mais de um conceito (homônimo):

DISCREPÂNCIAS		
N°	Tipo	Descrição
4	Inconsistência	O termo "edital" está sendo utilizado neste contexto para descrever tanto um arquivo físico quanto um conjunto de informações.

- Identificação de Relacionamentos e Multiplicidades

No passo 2, deve ser identificado como as classes encontradas no passo 1 se relacionam entre si e qual a cardinalidade destes relacionamentos.

a) Identificação de Relacionamentos

Os verbos conectam as classes de forma a estabelecer um relacionamento entre elas.

Um exemplo obtido na aplicação é demonstrado no fragmento a seguir:

> (...) os polos são utilizados pelas IES (...)

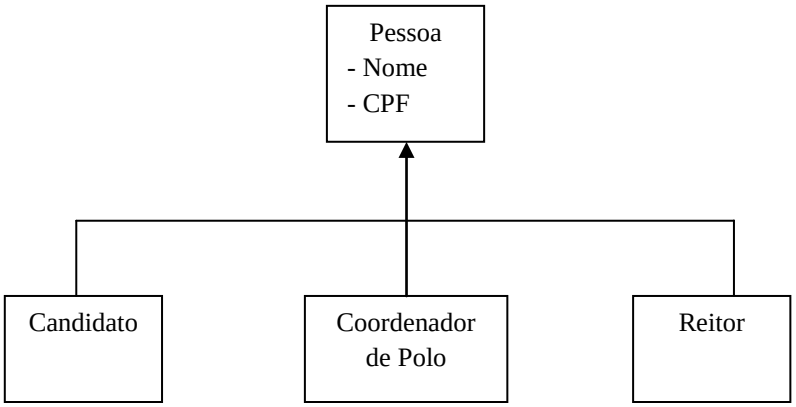
A identificação deste verbo que relaciona as classes "polo" e "IES" originou o seguinte relacionamento:

b) Identificação de Generalização.

No documento de visão é comum encontrar conceitos que possuem características similares e que poderiam ser abstraídos para um conceito mais amplo e menos específico.

É compreensível que essa preocupação não seja levada em consideração neste documento já que seu objetivo é detalhar as informações referentes a aplicação.

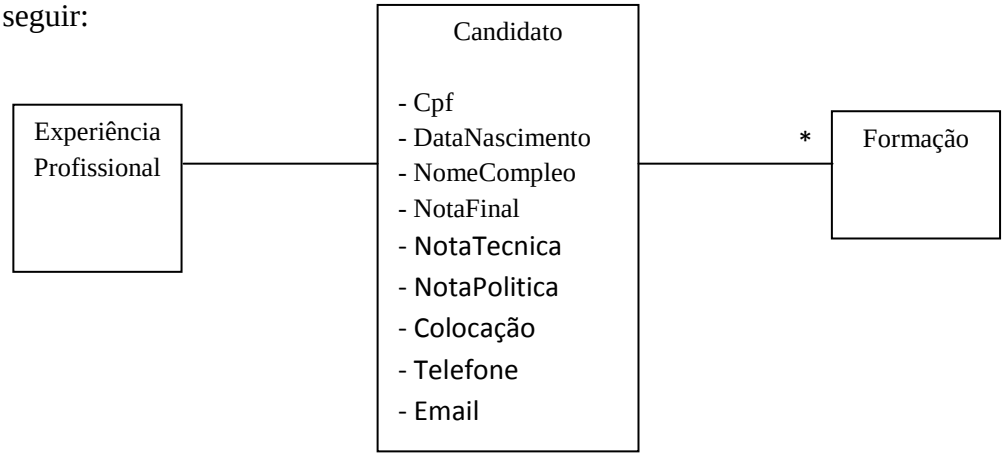
No exemplo utilizado, existem 3 classes que poderiam ser generalizadas, porém com as informações obtidas no documento não é possível afirmar se esta generalização de fato será realizada. A seguir, é demonstrada uma possível generalização das classes "Candidato", "Coordenador de Polo" e "Reitor", considerando que todas estas classes possuem os atributos "Nome" e "CPF":



c) A multiplicidade (ou cardinalidade) dos relacionamentos é normalmente descrita conforme o fragmento a seguir:

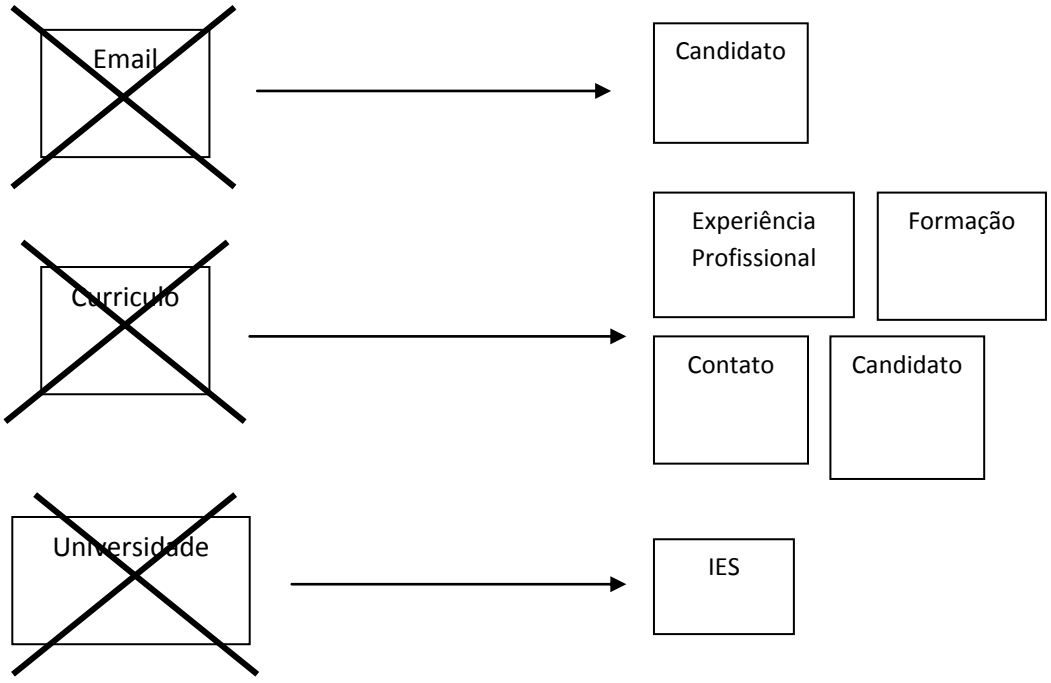
> (...) *O candidato deve poder cadastrar <u>várias</u> experiências profissionais* e <u>várias</u> formações (...)

A representação no modelo de classes deve ser realizada conforme fragmento a seguir:



d) *Classes identificadas que não possuam relacionamento com nenhuma outra pode significar que este conceito foi erroneamente identificado ou já está sendo incluído em outra classe.*

No exemplo utilizado, três classes não possuíam nenhum relacionamento: Email, Currículo e Universidade. Todas já estavam representadas em outras classes ou atributos, como demonstra o esquema a seguir:



e) Responder as questões de verificação de defeitos.

P2.1 *Se as informações presentes no documento de visão não permitem estipular a multiplicidade dos relacionamentos envolvidos isso pode comprometer a qualidade do modelo de classes desenvolvido. Nestes casos reporte uma omissão*.

DISCREPÂNCIAS		
N°	Tipo	Descrição
13	Omissão	Não foi possível determinar quantos e nem quais documentos serão enviados pelo candidato para fins de comprovação.

P2.2 *Caso exista algum relacionamento onde a cardinalidade indicada é diferente em momentos distintos do documento, deve ser reportada uma inconsistência, pois não é possível afirmar qual delas é a correta.*

DISCREPÂNCIAS		
N°	Tipo	Descrição
12	Inconsistência	No glossário, na descrição do termo “Coordenador de Polo”, é indicado que um coordenador de polo pode ser coordenador de mais de um polo. Contudo o resto do documento leva a crer que esta relação é 1 – 1, ou seja, um coordenador para um polo.

P2.2 *Ao final da construção do modelo inicial de classes, é importante refletir sobre as classes, atributos e relacionamentos identificados para certificar que estão coerentes com o domínio da aplicação. Caso seja identificada alguma discrepância, deve ser relatada uma informação estranha.*

Nenhum defeito foi identificado através desta pergunta. Isso ocorreria se houvesse um relacionamento entre as classes “Reitor” e “Experiência Profissional”, por exemplo. Livre de contexto, o relacionamento é coerente, porém no domínio da aplicação do exemplo isto não é parte do escopo, caracterizando uma informação estranha.

4.1.2 PBR – Usuário

O cenário PBR – Usuário também foi dividido em dois passos, apresentados a seguir:

- Identificação dos Participantes

Esta etapa tem o objetivo de identificar os participantes do sistema e destacar entre eles quais serão os atores.

a) Leitura do Documento de Visão
O objetivo desta etapa é ganhar conhecimento do domínio da aplicação, permitindo associações mais corretas, conforme descrito no item anterior.

b) Identificação de participantes e atores.
O fragmento a seguir demonstra a identificação de participantes no documento de visão:

(...)um funcionário alocado no polo para gerenciá-lo, chamado Coordenador de Polo (...)

(...) Pessoas afetadas: Reitor da IES (...)

c) Preenchimento do Formulário de Participantes

O formulário de participantes deve ser preenchido conforme demonstrado a seguir:

PARTICIPANTES		
É ator?	**Nome**	**Descrição**
☐	Coordenador de Polo	Os coordenadores de polo são selecionados pelo sistema e tornam-se responsáveis por gerenciar a infraestrutura e as atividades de um polo
■	Candidatos a Coordenador de Polo	São pessoas candidatas a vaga de coordenador de polo, enviam currículo e documentação comprobatória, além de receber uma notificação do sistema caso sejam selecionados.

d) Identifique os atores entre os participantes selecionados

O cenário propõe uma reflexão para auxiliar os inspetores a detectar quais são os participantes com interação direta com o sistema, serão os atores.

Estes devem ser identificados no campo "É ator?" com um quadrado preenchido, conforme o exemplo acima.

e) Responder as questões de verificação de defeitos

P1.1 *Múltiplos termos para definir um mesmo participante podem provocar falhas na identificação de cada ator e na compreensão de suas responsabilidades.*

Caso identifique um problema semelhante a este no documento o inspetor deve relatar uma inconsistência, conforme o exemplo abaixo.

DISCREPÂNCIAS		
N°	Tipo	Descrição
5	Inconsistência	Os temos "candidatos" e "concorrentes" representam sinônimos e estão sendo utilizados para referenciar os mesmos atores.

P1.2 *Se a descrição de como o sistema interage com o participante é inconsistente com a descrição do participante, reporte uma inconsistência.*

Este tipo de defeito não foi encontrado no modelo utilizado. Em uma simulação isso ocorreria se o sistema exigisse ao reitor que confirmasse cada inscrição de currículo no processo seletivo. Isso porém seria inconsistente com o conceito de reitor IES definido, uma pessoa responsável por manter o edital e avaliar os memoriais descritivos.

P1.3 *Caso um participante, sistema, tipo de usuário ou hardware importante for omitido, reportar uma omissão.*

O exemplo abaixo demonstra um hardware importante para o contexto do sistema e não mencionado no documento de visão.

DISCREPÂNCIAS		
N°	Tipo	Descrição
7	Omissão	Um leitor biométrico, hardware fundamental para a execução de uma etapa no sistema não está descrito no documento.

P1.4*Sistemas externos ou classe de usuários que não interajam em momento algum com o sistema não devem ser incluídos no documento.*

Se isto ocorrer uma informação estranha é encontrada, conforme o exemplo a seguir:

DISCREPÂNCIAS		
N°	Tipo	Descrição
6	Informação Estranha	O participante “aluno” não possui nenhuma interação com o sistema.

- Identificação das Funcionalidades

Esta etapa tem o objetivo de identificar as principais funcionalidades do sistema e através disso gerar a lista inicial dos casos de uso que irão compor a solução do problema.

a) Identificação das funcionalidades do sistema e seus fluxos de exceção.

O inspetor deve atentar-se que apesar de uma mesma funcionalidade estar descrita diversas vezes ao longo do documento de visão ela deve ser incluída uma única vez.

O fragmento a seguir demonstra a identificação das funcionalidades do sistema.

> RF08 – Permitir o envio do resultado do processo seletivo de coordenador de polo aos candidatos
>
> (...)A nota final do candidato deve ser igual a soma das notas técnica e política divididos por dois. Em caso de empate, primeiramente deve ser considerada a nota técnica como critério de desempate, depois(...)
>
> 1 - Manter Edital de Processo Seletivo de Coordenador de Polo

b) Agrupamento das funcionalidades semelhantes, consecutivas ou com um objetivo comum.
Realizar reflexão sobre o agrupamento das funcionalidades, já que estas podem estar descritas em diferentes níveis de granularidade.

O fragmento abaixo demonstra a identificação do agrupamento de funcionalidades reunidas em Manter Edital de Processo Seletivo de Coordenador de Polo:

Necessidades	**Funcionalidades Correspondentes**
1. Manter Edital de Processo Seletivo de Coordenador de Polo	– Cadastrar Edital; – Consultar Edital; – Editar Edital; – Excluir Edital.

c) Incluir agrupamentos e funcionalidades individuais identificadas no Formulário de Casos de Uso.

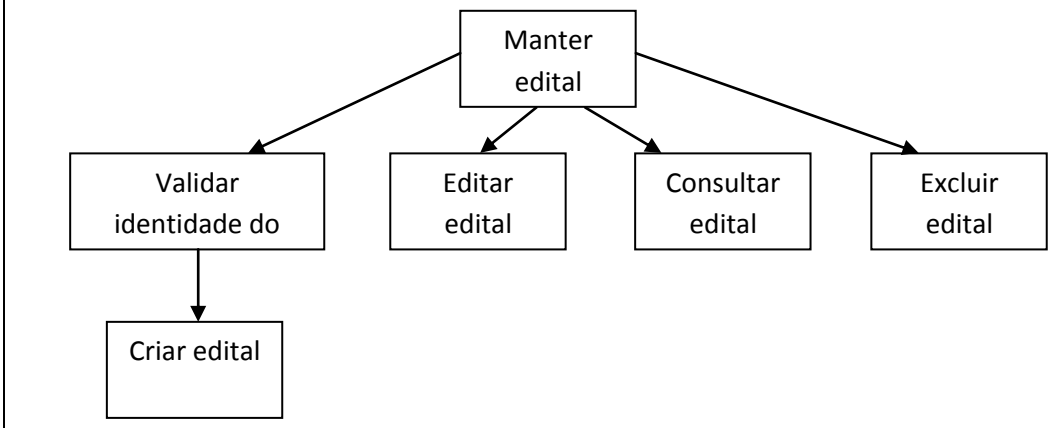
O formulário de casos de uso deve ser preenchido, conforme demonstrado a seguir:

FUNCIONALIDADES		
Nome do Caso de Uso	**Ator(es)**	**Descrição**
Manter edital de processo seletivo de coordenador de polo	Reitor IES	O sistema deve permitir a criação, edição, consulta e exclusão do edital do concurso.
Criar candidatura a coordenador de polo	Candidatos a coordenador de polo	O sistema deve permitir que o candidato a coordenador de polo cadastre seu currículo e calcule automaticamente a nota técnica.

d) Inclua os agrupamentos no Formulário de Casos de Uso – Fluxo de Execução
Uma árvore do fluxo de execução dos agrupamentos deve ser incluída para permitir uma maior compreensão sobre os casos de uso.

O exemplo abaixo demonstra esta etapa, indicando sua sequência e paralelismo:

Caso de Uso:	Manter edital de processo seletivo de coordenador de pólo



e) Verifique a consistência dos nomes e faça modificações se necessário

Conforme verificado no formulário de casos de uso demonstrado no item anterior, a funcionalidade “Envio de Currículos” foi renomeada para “Criar candidatura a coordenador de polo” por representar o cadastro de informações e um cálculo automático de nota, e não o envio de um arquivo como o nome sugere.

f) Inclua os atores envolvidos em cada caso de uso no Formulário de Casos de Uso.

Conforme verificado no formulário de casos de uso acima os atores Reitor IES e Candidatos a coordenador de polo foram incluídos nos casos de uso onde possuem participação.

g) Responda as questões de verificação de defeitos.

P2.1*Com base no conhecimento sobre o contexto do sistema e no documento de visão, o inspetor pode detectar alguma funcionalidade de algum caso de uso que foi omitida.*

Isso deve ser reportado como uma omissão, conforme o exemplo:

DISCREPÂNCIAS		
N°	Tipo	Descrição
8	Omissão	A funcionalidade edição de currículos não foi incluída no documento.

P2.2 *Se a descrição do fluxo de execução de alguma funcionalidade causa dúvida na interpretação e/ou possui termos vagos isso corresponde a uma ambiguidade e deve ser registrado no Formulário de Discrepâncias.*

DISCREPÂNCIAS		
N°	Tipo	Descrição
9	Ambiguidade	A funcionalidade "Manter critérios de pontuação automática" é descrita com o objetivo de "gerenciar os critérios", o que não permite identificar quais ações são realmente necessárias ao contexto.

P2.3*Caso as funcionalidades possuam restrições, o documento deve descrever de forma clara como essas restrições devem ser verificadas. Caso isto não ocorra isso corresponde a uma omissão.*

DISCREPÂNCIAS		
N°	Tipo	Descrição
10	Omissão	Não foi definido como a restrição de verificação da identidade do reitor será feita para permitir a criação de um novo edital.

P2.4*Podem existir funções descritas no documento e que não fazem parte do escopo do sistema. Nestes casos, uma informação estranha deve ser reportada.*

DISCREPÂNCIAS		
N°	Tipo	Descrição
11	Informação Estranha	A informação de que o edital deve ser arquivado na biblioteca da instituição não compõe o escopo da solução do sistema.

4.1.3 Verificação das Interações via Matriz CRUD

Após a execução dos dois passos da Etapa 1 da técnica, uma terceira verificação deve ser realizada, agora relacionando as interações das entidades identificadas no modelo de classes (passo 1) e as funcionalidades iniciais do sistema (passo 2). A matriz CRUD[10] resultante do exemplo de aplicação proposto é apresentada na Tabela 3.

Ao analisar a matriz CRUD de acordo com a proposta definida no capítulo 3, foram identificados alguns possíveis problemas na construção do documento de visão utilizado.

O primeiro ponto refere-se às entidades que não são cadastradas em nenhuma funcionalidade do sistema, direta ou indiretamente. Duas interpretações podem ser derivadas disto: a primeira sugere que os dados destas entidades são provenientes de um sistema externo (ou qualquer outra fonte de dados como, por exemplo, um arquivo) e carregados no sistema em questão; a segunda sugere uma falha na construção do documento de visão, ao omitir estas funcionalidades.

No exemplo foram identificadas três entidades que não possuem funcionalidade *create* definida e não se relacionam com nenhuma atividade: “IES”, “Polo” e “Reitor”. Como não foi especificada, esta informação pode ser considerada omissa, o que indica que a lista de funcionalidades não está completa, sendo necessária uma averiguação junto aos *stakeholders* do projeto.

[10] Termo utilizado para referir-se as quatro operações básicas de uma entidade: Create (criação), Retrieve (consulta), Update (alteração), Delete (exclusão).

Foram também identificadas diversas entidades sem *update* ou *delete* estabelecidos. São elas: “Coordenador de Polo”, “Experiência Profissional”, “Documentação Comprobatória” e “Formação”. O inspetor deve averiguar se é coerente ao contexto do sistema que as informações sejam permanentes. Isto de fato pode ocorrer como, por exemplo, em entidades que armazenam o histórico das informações de uma entidade.

Um ponto que deve ser observado com atenção é a abrangência das atividades CRUD em relação ao caso de uso, pois este pode ser realizar uma alteração parcial ou completa da entidade. Sendo assim, apresentar um *update* pode não ser suficiente para as necessidades do domínio. Por exemplo, a entidade “Candidato” é atualizada na funcionalidade “Incluir Nota Política no Currículo”, mas de acordo com o caso de uso apenas alguns atributos são atualizados. Deve ser verificado se a atualização dos outros atributos da entidade está realmente fora do escopo da solução.

Tabela 3 – Matriz CRUD das entidades em relação às funcionalidades do exemplo de aplicação.

Funcionalidade / **Entidade**	Manter Edital de Processo Seletivo de Coordenador de Polo	Criar Candidatura a Coordenador de Polo	Finalizar Processo Seletivo de Coordenador de Polo	Incluir Nota Política no Currículo	Manter Critérios de Pontuação Automática dos Currículos
Processo Seletivo			U		
Coordenador de Polo			C		
IES					
Polo					
Edital	CRUD				
Reitor					
Candidato		C		U	
Documentação Comprobatória		C			
Critério Pontuação					CRUD
Experiência Profissional		C			
Formação		C			
Memorial Descritivo		C		U	

[Fonte Própria]

4.2 Etapa 2 – Verificação dos Casos de Uso em relação ao Modelo de Classes

A segunda etapa da técnica possui o objetivo de verificar as inconsistências entre o modelo de casos de uso em relação ao modelo de classes.

O produto final desta verificação é chamado de Formulário de Discrepâncias (ver anexoII) e é onde serão mantidos os defeitos encontrados durante a aplicação desta etapa.

4.2.1 – Inclusão de Referências

Para a inclusão de referências nos modelos de caso em função dos modelos de classe, o inspetor é recomendado a seguir um passo a passo elaborado para auxiliá-lo a empregar a metodologia adequada. Este guia encontra-se representado na seção 3.4,Guia de verificação entre Modelos de Casos de Uso e Modelos de Classe.

A execução deste guia será simulada a seguir utilizando como exemplo o caso de uso Cadastrar Currículo, presente no anexo IV. Os fragmentos mais relevantes serão extraídos do texto e copiados neste documento a fim de permitir uma maior compreensão por parte do leitor.

1) Ler o documento de casos de uso utilizado.

   Conforme descrito anteriormente, a aplicação da técnica demonstrou a necessidade da leitura completa antes do início da inspeção. Entre os objetivos principais encontra-se o ganho de conhecimento no domínio da aplicação e o aumento da eficácia da técnica, que identificará termos mais relevantes ao sistema.

2) Incluir marcações referenciando classes, atributos, operações, relacionamentos, enumerações e restrições do modelo de classes e estados e operações de guarda dos diagramas de estado.

   As marcações deverão ser incluídas no Modelo de Caso de Uso após uma busca por ocorrência exata ou similar no diagrama de classes. A palavra que gerou a ocorrência deve ser tachada para que o inspetor visualize que ela está

apta a corresponder a um conceito do sistema e não somente do domínio da aplicação.

Os exemplos a seguir demonstram os itens identificados pela aplicação da técnica para cada item:

Tipo de Informação	Exemplos extraídos do caso de uso
Classe	↑Edital
Atributo	↑Candidato.código
Operação	↑Curriculo.salvar()
Relacionamentos	ProcessoSeletivo→Curriculo
Enumeração	↑NívelProfissional
Estados	↓EstadoEdital.recebendoCurriculos
Operação de Guarda	↓EstadoEdital.extenderPrazoRecebimentoCurriculos()
Restrição	↑Candidato.genero.FaixaValores:{masculino,feminino}

1) Incluir defeitos encontrados no Formulário de Discrepâncias

Caso o inspetor identifique alguma discrepância entre o modelo de casos de uso e o modelo de classes (por ter encontrado dificuldades em incluir uma marcação seja pela falta de informação, definição ou por sua irrelevância) ele deverá utilizar a marcação [DEFEITO - TipoDeDefeito] ao lado do item cujo defeito foi identificado e incluir uma correspondência no Formulário de Discrepâncias junto com seu tipo e uma breve descrição do problema, incluindo a marcação cujo defeito foi originado.

O trecho a seguir, retirado do Formulário de Discrepância gerado pela aplicação da técnica, será utilizado para exemplificar o problema.

Formulário de Discrepâncias		
N°	**Tipo**	**Descrição**
1	Omissão	O conceito de ↑ProcessoSeletivo é relevante para o sistema, porém não foi incluído no modelo de classes.
3	Ambiguidade	A RN2 não apresenta nível de detalhe suficiente para determinar qual data de início e fim deve ser considerada. Marcação não foi possível.
6	Inconsistência	O memorial descritivo do candidato encontra-se representado em ↑Curriculo.memorial.
7	Inconsistência	Os itens ↑Edital e ↑Edital.edital possuem a mesma definição apesar de representarem conceitos distintos.
8	Informação Estranha	O ↑Candidato.RG não é utilizado pelo sistema e não está contido em seu modelo de classes.

4.2.2 – Análise dos casos de uso utilizados

Identificamos que o caso de uso utilizado na aplicação da técnica não está conforme a definição de Ivar Jacobson.

Há itens do fluxo que não representam interações entre o sistema e seus atores, mas apenas cálculos (no exemplo, cálculo de notas) e validações que se encontram descritos como transações do sistema, enquanto deveriam compor alguma das existentes. O fragmento abaixo destaca o momento onde este problema foi identificado:

> (...)4. O sistema valida ↑Curriculo.validar() as informações inseridas pelo ator. [RNG1][RNG2][RNG3][RNG4][RNG9][RNG10]
>
> 5. O sistema registra ↑Curriculo.salvar() as informações do currículo.
>
> 6. O sistema calcula a nota do currículo do candidato (...)

Além disso, não há um fluxo principal único, com as opções alternativas descritas nos fluxos alternativos. Em alguns casos eles estão descritos como opções no próprio fluxo principal, conforme o exemplo:

(...) 3. O ator informa os dados do currículo.

a. Caso o ator deseje inserir mais registros de ↑Curriculo.experienciaProfissional (...)

b. Caso o ator deseje inserir mais registros de ↑Curriculo.formação (...)

Grande parte dos fluxos alternativos do sistema corresponde a uma única ação e as referências às regras de negócio relacionadas. Por consequência, se torna difícil compreender todas as condições que fazem o caso de uso permanecer no fluxo principal e a quantidade de fluxos alternativos existentes, acarretando, posteriormente, problemas na etapa de projeto. Um fragmento com o problema relatado se encontra a seguir:

(...)4. O sistema valida ↑Curriculo.validar() as informações inseridas pelo ator. [RNG1][RNG2][RNG3][RNG4][RNG9][RNG10] (...)

Todos estes pontos nos permitem observar que o modelo utilizado na aplicação não segue a padronização formal proposta por Ivar Jacobson. Com isso não é possível garantir a consistência do modelo de caso de uso utilizado para a verificação, apenas a consistência dos dados presentes em relação ao modelo de classes.

4.3 – Resultados Obtidos na Aplicação

Os resultados obtidos na aplicação da técnica nos artefatos do exemplo proposto demonstram que é possível identificar um número significativo de defeitos. Além disso, a distribuição dos defeitos encontrados aponta que é possível identificar todos os tipos definidos na taxonomia, tanto na aplicação da etapa 1 quanto na etapa 2 da técnica (ver Tabela 4).

Tabela 4 – Quantidade de defeitos encontrados na aplicação da técnica.

Etapa da Técnica		**Quantidade de Defeitos por Tipo**			
		Omissão	Inconsistência	Ambiguidade	Informação Estranha
Etapa 1	Projetista	1	3	1	1
	Usuário	3	0	1	2
Etapa 2	Inclusão de Referências	3	3	2	1
Total de Defeitos		7	6	3	4

[Fonte Própria]

5 CONCLUSÃO

5.1 Análise dos Resultados

Ao longo deste projeto, estudos foram realizados para conhecer as técnicas de inspeção de software e suas especializações a cada tipo de projeto. Tais especializações variam de acordo com suas metas de qualidade, tipos de problemas encontrados e características da empresa desenvolvedora do software, como nível financeiro e nível de experiência de seus profissionais.

A verificação de artefatos UML ganha destaque como método empregado nas inspeções, pois seus produtos servem como insumo para as fases seguintes do SDLC, evitando a transmissão e ampliação dos defeitos provenientes do levantamento de requisitos e análise do sistema.

Com o conhecimento adquirido foi desenvolvida uma nova técnica de inspeção de softwares, reunindo parte das técnicas abordadas e melhorias identificadas na sua aplicação. A abordagem proposta consiste em aplicar a técnica em duas etapas: a primeira, na fase de levantamento de requisitos, utiliza cenários customizados da PBR para verificar o documento de visão do negócio e construir o modelo inicial de classes e a lista inicial de casos de uso do sistema; a segunda, na fase da análise do sistema, propõe uma verificação dos casos de uso em função do modelo de classe para identificar suas inconsistências.

Um exemplo da aplicação da técnica foi realizado para este estudo com a finalidade de avaliar as características da técnica desenvolvida. Foi simulada a inspeção em alguns artefatos de um sistema para identificar seus pontos positivos e negativos. As vantagens identificadas vão além das previstas pela utilização das técnicas selecionadas, como, por exemplo, o aumento do conhecimento do domínio, produto da PBR, e o aprimoramento dos artefatos produzidos pela equipe de análise, produto da verificação dos modelos.

Consideradas em conjunto, as etapas proporcionam as seguintes vantagens: boa relação custo-benefício na correção de defeitos com sua detecção precoce no ciclo de vida de desenvolvimento de softwares;cobertura dos principais artefatos gerados na fase de análise e seus principais tipos de defeitos; baixa complexidade na sua execução

através da utilização de um conjunto de instruções simples; não possui restrições quanto ao tamanho da equipe de instrutores ou nível de experiência individual; auxilia na construção dos artefatos iniciais aderentes a documentação do projeto; o nível de cobertura fica a critério do responsável pela inspeção, que determina a quantidade de vezes e os artefatos que deverão ser analisados, além de quais das etapas serão utilizadas na inspeção.

Por outro lado, algumas desvantagens foram identificadas. A principal delas diz respeito ao tempo gasto no processo de inspeção. Na execução da técnica desenvolvida foram gastas 6 horas para que os artefatos propostos fossem verificados, sendo 4 horas para execução da primeira etapa e 2 horas para a execução da segunda etapa, como demonstrado na Tabela 4. Este tempo foi considerado elevado em relação ao planejamento inicial deste projeto, que visava a construção de uma técnica que fosse de rápida aplicação, considerando que o tempo é um dos fatores que desmotiva a realização de inspeções frequentes nas empresas.

Tabela 5 – Tempo gasto na aplicação da técnica no cenário proposto.

Fase	**Tempo Gasto**
Verificação do documento de visão do negócio e construção de artefatos iniciais da análise.	4h
Verificação dos casos de uso em relação ao modelo de classes.	2h
	Tempo Total: 6h

[Fonte própria]

Outros pontos negativos encontrados foram: a técnica é totalmente manual; envolve um conjunto extenso de instruções ao inspetor o que torna sua execução trabalhosa; contempla poucos artefatos UML e não prevê uma visão completa do sistema em sua segunda etapa.

Levando em consideração que a técnica foi aplicada em um conjunto limitado de artefatos e de uma única fonte, não é possível afirmar que tais vantagens e desvantagens sejam verdadeiras para outros contextos. Estudos mais aprofundados devem ser realizados para confirmar tais indícios.

5.2 Trabalhos Futuros

Algumas necessidades foram percebidas ao longo deste projeto, mas não foram contempladas por não fazerem parte do escopo deste trabalho.

Para reduzir o esforço necessário para a aplicação da técnica, deve ser considerada a utilização de apoio ferramental que auxilie o inspetor a realizar as instruções propostas. Suas principais funcionalidades compreenderiam o apoio à criação do modelo inicial de classes, a geração da matriz CRUD a partir dos produtos gerados na primeira etapa do processo, a identificação automática dos conceitos presentes no modelo de classes e diagrama de estados (classes, atributos, operações etc.) que representam insumos da segunda etapa e sua sugestão na inclusão de referências proposta para a verificação dos modelos.

Como complemento à técnica, propõe-se o desenvolvimento de um fluxo de gerenciamento de inconsistências, que permita aos envolvidos controlar a resolução das discrepâncias encontradas e analisar os problemas identificados, permitindo o aprimoramento do processo de desenvolvimento de softwares. Outra proposta compreende a utilização das marcações criadas na segunda etapa da técnica como forma de rastreabilidade entre casos de uso e modelos de classes e conexão bidirecional entre eles, permitindo que as alterações em um modelo conheçam as implicâncias ao outro e então sejam replicadas ou canceladas para que mantenham a consistência dos dados verificados.

REFERÊNCIAS BIBLIOGRÁFICAS

BARNARD, J. ; PRICE, A. Managing Code Inspection Information. IEEE Software, 1994.

BASILI, V.; GRENN, S.; LAITENBERGER, O.; LANUBILE, F.; SHULL, F.; SØRUNGÜARD, S.; ZELKOWITZ, M. Lab Package for the Empirical Investigation of Perspective-Based Reading, 1998.

BASILI, V.; SHULL, F.; LANUBILE, F. On IEEE Transactions on Software Engineering. In Building Knowled get through Families of Experiments, 1999. Vol. 25, n. 4, pages 456-473.

BERLING, T. Increasing Product Quality by Verification: Validation Improvements in an Industrial Setting, Doctoral Thesis, Lund University, Sweden, 2003.

BERTINI, L.A; KIRNER, T.G.; MONTEBELO, M.I.; LARA, I.A R. Técnicas de Inspeção de Documentos de Requisitos de Software: um Estudo Comparativo. IX Workshop onRequirementsEngineering, Rio de Janeiro, 2006.

BIFFL, B.; BED, F.; LAINTENBERGER, O. On the International Conference on Software Engineering. In Investigating the Cost-Effectiveness of Reinspections in Software Development, 2001. Pages .155-164.

BOOCH G.; JACOBSON I.; RUMBAUGH J. Unified Modeling Language Specification, OMG. Version 1.3, 2000.

CHRISTENSON D.A.; HUANG S.T.; LAMPEREZ A.J. Statistical quality control applied to code inspections. IEEE Journal on Selected Areas in Communications, 1990.

CIOLKOWSKI, M.; DIFFERDING, C.; LAITENBERGER, O.;M'UNCH, J. Empirical Investigation of Perspective-Based Reading: A Replicated Experiment, ISERN Report No. 97-13, 1990.

CORRÊA, A. L.; SOUZA, M. S. Curso engenharia de requisitos, unidade 1, slide 1. PUC-RJ, 2011.

FAGAN M. E. Design: code inspection to reduce erros in program development. IBM Systems Journal.Volume:15issue: 3, 1976.

GLINZ, M. A Lightweight Approach to Consistency of Scenarios and Class Models, Institut für Informatik.Universit ät Zürich, Switzerlan, 2000.

GUNNAR, O.; PALMKVIST, K. Use Cases: Patterns; Blueprints. Addison- Wesley, 2005.

JACOBSON, I. Use Cases; Architecture, 2008.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. The Unified Software Development Process. 1999.

JACOBSON, I.; MAGNUS, C.; PATRICK, J.; GUNNAR, O. Object-Oriented Software Engineering, Addison-Wesley, 1992.

KALINOWSKI, M.; SPÍNOLA, R.O.; TRAVASSOS, G.H. In Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software. On Simpósio Brasileiro de Qualidade de Software, Brasília, 2004.

KELLY J. C.; SHERIF, J. S.; HOPS, J. Analysis of defect densities found during software inspections, Journal of Systems ; Software, 1992.

KOSCIANSKI A.; SOARES M. S. Qualidade de Software, Ed. Novatec, 2nd edition, 2007.

KÖSTERS G.; SIX H. W.; WINTER M. Coupling Use Cases ; Class Models as a Means for Validation ; Verification of Requirements Specifications. ISOWARE GmbH, Hagen, Germany, 2001.

LAHTINENJ. Application of the perspective-based reading technique in the nuclear I&C contexto, 2012.

LAITENBERGER O. An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents, 2000.

MAFRA S. N.; TRAVASSOS G. H. Técnicas de Leitura de Software: Uma Revisão Sistemática, 2005.

MAFRA, S. N. Leitura Baseada em Perspectiva: Visão do Projetista Orientada a Objetos, 2006.

MELO, S.M. Inspeção de Software. Universidade de São Paulo (USP). <http://moodle.stoa.usp.br/mod/resource/view.php?id=12776>, Acesso em 15 de dez. de 2012.

MELO, W.; SHULL, F.; TRAVASSOS, G. H. In Software Review Guidelines, On Systems Engineering; Computer Science Program - PESC ES-556/01. COPPE/UFRJ, 2001.

NETO A. F.; HIGA W.; FURLAN J. D. Engenharia da Informação – Metodologia, Técnicas e Ferramentas. McGraw-Hill, 1998.

POHL K.; HAUMER P. Modeling Contextual Information About Scenarios. 3rd Workshop on Requirements Eng. On Foundation for Software Quality, Spain, 1997.

PORTER, A.; VOTTA, L. Comparing Detection Methods for Software Requirements Inspection: A Replication Using Professional Subjects, Empirical Software Engineering, 1998. Capitulo 3(4), páginas 355–380.

PRESSMAN, R. S. Engenharia de Software. McGraw-Hill, 2002.

RUFUS, O. O. Reading techniques for Software Inspection: Review; Analysis, 2010.

SHULL, F. Developing Techniques for using software documents: a series of empirical studies. Maryl; University, 1998.

SHULL, F.; CARVER, J.; TRAVASSOS, G. H.; MALDONADO, J. C.; CONRADI, R.; BASILI, V. Replicated Studies: Building a Body of Knowledge about Software Reading Techniques, 2003.

STAIR, R.; REYNOLDS, G. Princípios de Sistemas de Informação, pages 467-468. Ed. Cengage, 2010.

TRAVASSOS G. H. Reading Techniques for OO Design Inspections, 2002.

TRAVASSOS G. H. Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality. On Conference on Object-Oriented Programming, Systems, Languages, Applications (OOPSLA), Denver, Colorado, 1999.

TRAVASSOS, G. H. Revisão e Inspeção de Software, 2004.<http://lens.cos.ufrj.br:8080/eselaw2007/ESELAW07-TRAVASSOS(SC2).pdf>.Acessado em 20 de jun. de 2013.

XEXÉO, G. Modelagem de Sistemas de Informação: Da Análise de Requisitos ao Modelo de Interface, 2007.

ANEXO I – Documento de Análise Inicial

Documento de Análise Inicial

Responsável:_____ Data: ___/___/____

Nome do Documento Inspecionado: _____

Formulário de Classes

Utilize o formulário abaixo para reportar as classes extraídas do documento de visão sendo inspecionado. Para cada classe extraída, indique seu nome, seus atributos, e suas responsabilidades. Ao preencher os atributos pertencentes à classe, indique o tipo básico (textual, numérico, alfanumérico, data, etc) e o nome do atributo. Caso o atributo represente uma faixa de valores, preencha seus possíveis valores.

CLASSE:					
ATRIBUTOS					
Nome		Tipo		Valores	
Nome		Tipo		Valores	
Nome		Tipo		Valores	
Nome		Tipo		Valores	
RESPONSABILIDADES					

Modelo de Classes

Desenhe aqui o modelo de classes.

Formulário de Participantes

PARTICIPANTES		
É ator?	**Nome**	**Descrição**
☐		
☐		
☐		
☐		
☐		
☐		
☐		

Formulário de Casos de Uso

FUNCIONALIDADES		
Nome do Caso de Uso	**Ator(es)**	**Descrição**

Formulário de Casos de Uso – Fluxo de Execução

Caso de Uso:	

Formulário de Discrepâncias

Taxonomia	Descrição
Omissão	Falta de representação ou definição de informação relevante para o projeto.
Inconsistência	A mesma informação definida de formas diferentes no projeto.
Ambiguidade	Informação com falta de clareza na especificação, causando problemas na sua interpretação.
Informação Estranha	Informação que é fornecida, mas não é necessária ou nunca é utilizada.

DISCREPÂNCIAS		
N°	**Tipo**	**Descrição**

ANEXO II – Documento de Análise Final

Documento de Análise Final

Responsável:_____ Data: ___/___/____

Nome do Documento Inspecionado: _____

Formulário de Discrepâncias		
N°	**Tipo**	**Descrição**

ANEXO III – Documento de Visão

Sistema de Ensino à Distância EaD +

Documento de Visão

Versão 1.1

Histórico de Revisão

Data	Versão	Descrição	Autor
25/07/2013	1.0	Criação do documento	José da Silva
18/08/2013	1.1	Alteração da formatação e inclusão de mais informações	José da Silva

Documento de Visão

1 Introdução

1.1 Finalidade

A finalidade deste documento é definir a visão que os *stakeholders* têm do produto, em termo de suas necessidades e das funcionalidades para atendê-las. O documento contém uma visão geral dos requisitos mais importantes do projeto.

1.2 Escopo do Documento

Este documento de visão se aplica ao *Sistema de Ensino à Distância EaD+*. O projeto tem por objetivo a automação de soluções de informática para o processo seletivo de coordenador de polo.

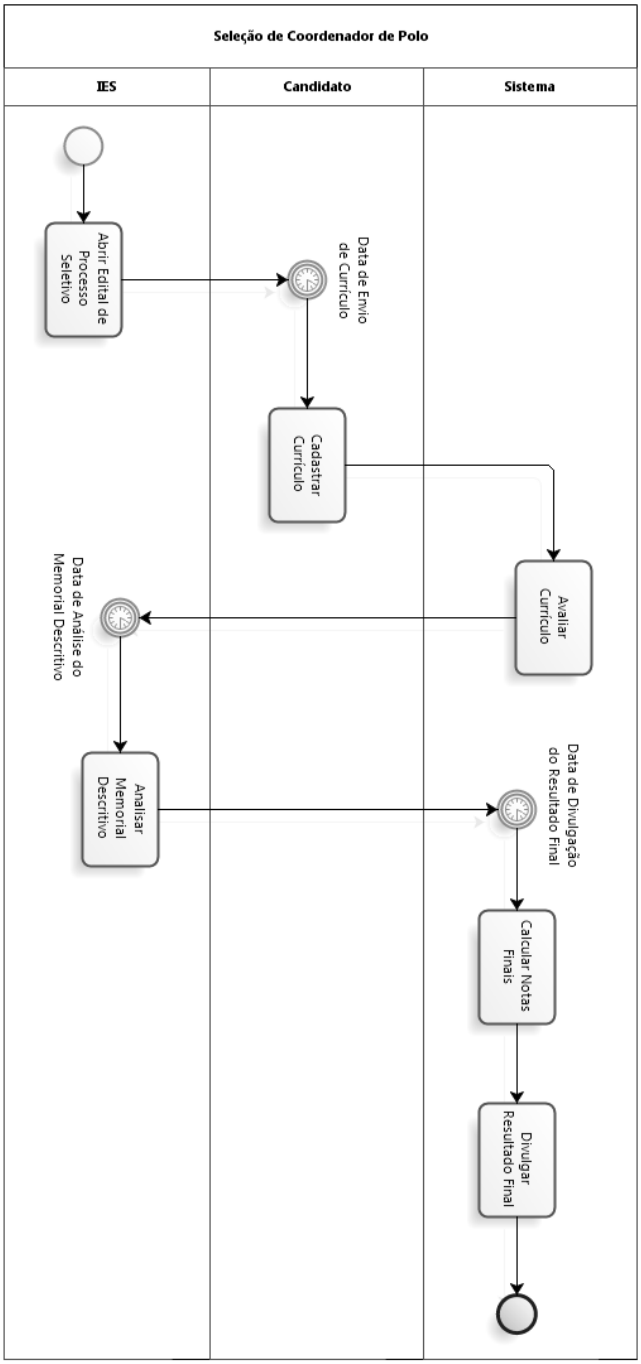
2 Contextualização

O sistema se ambienta em instituições que oferecem cursos de ensino à distância, ou seja, cursos ministrados fora do ambiente da instituição de ensino superior - IES.

O material didático e as aulas das disciplinas dos cursos são disponibilizados em um AVA (Ambiente Virtual de Aprendizagem) na internet, o que elimina a necessidade da presença física do aluno na universidade. Apesar disso, as IES mantêm alguns locais de apoio para possíveis eventos (por exemplo, aplicação de provas), com laboratórios de informática para oferecer acesso a internet aos alunos e biblioteca. Estes locais de apoio são chamados de polos no contexto do sistema.

Os polos são utilizados pela IES, mas não são propriedades dela. Por isso, para garantir que a sua infraestrutura esteja adequada e seu bom funcionamento, a IES mantém um funcionário alocado no polo para gerenciá-lo, chamado Coordenador de Polo. A seleção do coordenador de polo é feita através de um processo seletivo aberto por um edital oficial da IES, podendo qualquer pessoa enviar seu currículo e se candidatar.

3 Macroprocesso do Processo Seletivo de Coordenador de Polo



4 Problemas e Soluções Propostas

Problema	Validações referentes a construção do edital de abertura do processo seletivo são realizadas manualmente.
Pessoas Afetadas	Reitor IES.
Impacto	Edital publicado com informações incorretas ou faltantes.
Solução	Apoio à construção do edital via sistema, incluindo validações automáticas.

Problema	Candidatos precisam enviar o currículo e toda documentação comprobatória por correspondência.
Pessoas Afetadas	Candidatos à Coordenador de Polo.
Impacto	Trabalho e custos adicionais para o candidato.
Solução	Funcionalidade para envio de currículo e documentação comprobatória via sistema.

Problema	Avaliação manual dos currículos.
Pessoas Afetadas	Reitor da IES.
Impacto	Processo seletivo lento e custoso.
Solução	Automatização de parte da avaliação dos currículos. A análise dos currículos será realizada parte pelo sistema, que avalia automaticamente os currículos de acordo com critérios pré-definidos, parte pelo Reitor da IES, que analisa os memoriais descritivos dos candidatos.

Problema	Cálculo manual da nota final dos candidatos.
Pessoas Afetadas	- Reitor da IES. - Concorrentes à Coordenador de Polo.

Impacto	Colocação dos candidatos indevida, em razão de cálculo errado da nota final.
Solução	Automatização do cálculo da nota final e da montagem da lista contendo a colocação dos candidatos.

Problema	Resultado final do processo seletivo divulgado via correspondência.
Pessoas Afetadas	- Reitor da IES. - Candidatos à Coordenador de Polo.
Impacto	Processo seletivo lento e custoso.
Solução	Divulgação do resultado final do processo seletivo via sistema.

5 Principais Usuários

Identificação	**Responsabilidades**
Reitor da IES	Manter o edital de processo seletivo de coordenador de polo. Avaliar o memorial descritivo de cada candidato.
Candidato à Coordenador de Polo	Enviar o currículo e a documentação comprobatória em resposta a um processo seletivo de coordenador de polo.
Aluno	Utilizam os polos para acesso a internet e biblioteca. Realizam provas localizadas nos polos.

6 Visão Geral do Produto

6.1 Perspectiva do Produto

O sistema EaD+ é autossuficiente, ou seja, não possui nenhum vínculo com outros sistemas majoritários. Sua finalidade é a de agilizar e evitar erros nas etapas do fluxo de trabalho envolvido na criação, candidatura, análise e finalização do processo seletivo de coordenador de polo.

6.2 Resumo das Funcionalidades do Produto

Necessidades	**Funcionalidades Correspondentes**
2. Manter Edital de Processo Seletivo de Coordenador de Polo	– Cadastrar Edital; – Consultar Edital; – Editar Edital; – Excluir Edital.
3. Enviar Currículo de Candidato à Coordenador de Polo	– Enviar Currículo.
4. Avaliar Currículo do Candidato Automaticamente	– Avaliar Currículo.
5. Avaliar Memorial Descritivo do Candidato	– Avaliar Memorial Descritivo.
6. Calcular Notas Finais dos Candidatos	– Calcular Notas Finais.
7. Divulgar Resultado de Processo Seletivo de Coordenador de Polo	– Divulgar Resultado Final.

7 Premissas

As máquinas clientes deverão dispor de rede e *browser*.

8 Requisitos Funcionais

RF01 – Permitir a criação de edital para abertura de processo seletivo de coordenador de polo

O sistema deve permitir o cadastramento de edital para abertura do processo seletivo de coordenador para um polo específico, sendo obrigatório informar: polo, nome do edital, data de envio dos currículos, período para análise dos currículos pela IES, data da publicação do resultado final e o edital. A abertura de edital é condicionada a comprovação da identidade do reitor, portanto deve-se garantir que as digitais fornecidas são compatíveis com a do reitor da IES.

RF02 – Permitir o cadastro de currículo do candidato a Coordenador de Polo

O sistema deve permitir o envio de currículos das pessoas que queiram se candidatar ao cargo de coordenador do polo. O candidato deve informar nome completo, CPF, RG, data de nascimento, gênero, endereço e contatos. Além disso, deve informar sua experiência profissional e sua formação. O candidato deve poder cadastrar várias experiências profissionais e formações. O candidato deve redigir um memorial descritivo contendo uma breve descrição de no máximo 1580 caracteres sobre suas experiências profissionais, sua motivação e seus objetivos no cargo pretendido.

RF03 – Permitir a análise dos currículos dos candidatos a coordenador de polo pela IES atuante no polo

As instituições de ensino que utilizam o polo devem participar do processo seletivo do coordenador de polo. O reitor da IES analisa apenas o memorial descritivo dos candidatos não sendo informada de quem é o memorial (medida para evitar eventuais favorecimentos) e registra a nota que pode variar de 0 a 10 e conter até 2 casas decimais, e uma observação opcional a respeito da nota indicada.

RF04 – Permitir a análise automática dos currículos dos candidatos a coordenador de polo

No cadastro de cada currículo de candidato à coordenador de polo, o sistema deve atribuir uma nota ao currículo, de acordo com a experiência profissional, a formação e os critérios de pontuação definidos pela IES.

RF05 – Permitir o controle dos critérios de análise automática do currículo dos candidatos a coordenador de polo

O sistema deve permitir que a IES gerencie os critérios de pontuação automática dos currículos dos candidatos à coordenador de polo. Os critérios de pontuação automática são aplicados para os tipos de formação graduação e pós-graduação e experiência profissional.

RF06 – Permitir a finalização automática do processo seletivo de coordenador de polo

O sistema deve finalizar automaticamente o processo seletivo assim que a data de fim da etapa de análise for atingida. A nota final do candidato deve ser igual à soma das notas técnica e política divididos por 2. Em caso de empate, primeiramente deve ser considerada a nota técnica como critério de desempate, depois a nota da experiência profissional e por último a data de nascimento (o candidato com maior idade possui prioridade).

RF07 – Permitir o envio do resultado do processo seletivo de coordenador de polo à IES atuante no polo

O sistema deve enviar um e-mail para o Reitor da IES contendo a lista dos 5 primeiros colocados do processo seletivo com nome, CPF e telefone de cada candidato, assim que a data de divulgação do resultado final for atingida.

RF08 – Permitir o envio do resultado do processo seletivo de coordenador de polo aos candidatos

O sistema deve enviar um e-mail para cada candidato do processo seletivo contendo a lista (em ordem decrescente de nota) com a colocação final do processo seletivo com nome, CPF, nota do currículo, nota da IES e nota final.

9 Glossário

Termo	Descrição
IES	Instituição de Ensino Superior.
Coordenador de Polo	Pessoa responsável por gerenciar a infraestrutura e as atividades de um polo.
Edital	Documento oficial que contém as normas e condições do processo seletivo.
Nota Técnica	Nota do currículo, calculada pelo sistema.
Nota Política	Nota da IES atuante no polo.

ANEXO IV – Caso de Uso Cadastrar Currículo

Cadastrar Currículo

Descrição:	Esse caso de uso permite cadastrar currículo para determinado processo seletivo de coordenador de polo de uma IES.

- **Requisitos**

  RF02 – Permitir o cadastro de currículo do candidato a Coordenador de Polo

  RF04 – Permitir a análise automática dos currículos dos candidatos a coordenador de polo

- **Atores**

  Candidato

- **Fluxos**

Fluxo Básico B1 - Cadastrar Currículo

a. O caso de uso é iniciado quando o ator indica a intenção de cadastrar seu currículo em um processo seletivo de coordenador de polo.
b. O sistema solicita o preenchimento das informações do currículo. [ED1]
c. O ator informa os dados do currículo.
    a. Caso o ator deseje inserir mais registros de experiência profissional, o sistema habilita áreas para registrar as demais experiências.
    b. Caso o ator deseje inserir mais registros de formação, o sistema habilita áreas para registrar as demais formações.
d. O sistema valida as informações inseridas pelo ator. [RNG1][RNG2][RNG3][RNG4][RNG9][RNG10]
e. O sistema registra as informações do currículo. [ED2]
f. O sistema calcula a nota do currículo do candidato utilizando critérios pré-definidos. Após o cálculo, o sistema registra a nota no sistema.

[ED3][RNG5][RNG6][RNG7]

g. O sistema volta ao passo 2, exibindo a mensagem [MSG1].

ExceçãoE1 - Campos Obrigatórios não preenchidos

1 - O sistema detecta a violação da regra [RNG1]

2 - O caso de uso volta ao passo anterior ao passo chamador e exibe a mensagem [MSG2].

ExceçãoE2 - Data inicial maior que data final

1 - O sistema detecta a violação da regra [RNG2]

2 - O caso de uso volta ao passo anterior ao passo chamador e exibe a mensagem [MSG3].

ExceçãoE3 - CPF já cadastrado

1 - O sistema detecta a violação da regra [RNG3]

2 - O caso de uso volta ao passo anterior ao passo chamador e exibe a mensagem [MSG4].

ExceçãoE4 - Data inicial maior que data final

1 - O sistema detecta a violação da regra [RNG4]

2 - O caso de uso volta ao passo anterior ao passo chamador e exibe a mensagem [MSG5].

ExceçãoE5 - Memorial Descritivo excede o tamanho máximo

1 - O sistema detecta a violação da regra [RNG9]

2 - O caso de uso volta ao passo anterior ao passo chamador e exibe a mensagem [MSG6].

- **Condições**

Pré-condição

1. O ator está no ambiente de cadastro de currículo para seleção de coordenador de polo.
2. O edital de seleção de coordenador de polo está no período de envio de currículo, ou seja, a data de execução do caso de uso é maior ou igual que a data de início do período de envio de currículo e menor ou igual que a data de término de envio de currículo. Enquanto o edital estiver na etapa de análise dos currículos o prazo pode ser estendido pelo reitor o que o coloca novamente na etapa de recebimento de currículos.

Pós-condição

1. Currículo do candidato a coordenador de polo cadastrado no sistema.
2. Nota automática do currículo calculada e cadastrada no sistema.

- **Mensagens**

MSG1

Currículo cadastrado com sucesso.

MSG2

Os seguintes campos obrigatórios não foram preenchidos:

- <<campo_1>>
- <<campo_2>>

MSG3

As datas iniciais devem ser anteriores às datas finais.

MSG4

O CPF informado já possui currículo cadastrado.

MSG5

Insira ao menos 1 graduação no currículo.

MSG6

O memorial descritivo deve conter no máximo 1580 caracteres.

- **Regras de Negócio**

RNG1 – Campos Obrigatórios

Os campos obrigatórios devem ser preenchidos.

RNG2 – Data Início x Data Fim

Data de início deve ser anterior à data de fim.

RNG3 – Cadastro único de currículo por CPF

Apenas um currículo pode ser cadastrado por CPF.

RNG4 – Graduação obrigatória

Pelo menos uma graduação deve ser informada pelo candidato ao coordenador de polo.

RNG5 - Pontuação da Graduação

Deve ser igual à pontuação da graduação de área com maior pontuação. Caso o candidato possua 3 ou mais graduações deverá ser acrescido o valor do campo "ponto +1" ao cálculo:

pontuacao_graduacao_maior_area + [ponto_extra]

RNG6 - Pontuação da Pós-Graduação

Deve ser igual à pontuação da pós-graduação de área com maior pontuação.

RNG7 - Pontuação da Experiência Profissional

Deve ser igual à soma da pontuação obtida em cada experiência profissional informada pelo candidato. Cada experiência será pontuada multiplicando a

pontuação da duração (período em que trabalhou no nível) pela pontuação do nível da experiência profissional:

∑(pontuacao_duracao * pontuacao_nivel)

RNG8 - Código do Candidato

Um código único deve ser gerado para todo candidato a coordenador de polo que enviar o seu currículo, para garantir que a avaliação seja às cegas. Este código deve ser gerado da seguinte forma:

<<ano do processo seletivo>>.<< número do último candidato + 1>>

Exemplo: 2013.16

RNG9 - Tamanho Máximo do Memorial Descritivo

O memorial descritivo do candidato à coordenador de polo não pode exceder 1580 caracteres.

RNG10 - Cálculo da Nota Técnica

A nota técnica é igual ao soma da nota da graduação mais a nota da pós-graduação mais a nota da experiência profissional.

ED1 – Dados Informados no Cadastro de Currículo

Polo* = lista de polos que possuem edital em andamento.

Informações Gerais

- Nome Completo*
- CPF*
- Data Nascimento*
- Logradouro*
- Complemento
- Município*

- Tel. Residencial*

- E-mail*

- Gênero* = {Masculino, Feminino}

- RG*

- CEP*

- N°*

- Bairro*

- UF*

- Tel. Celular

Experiência Profissional

- Instituição*

- Nível* = {Estágio, Operacional, Chefia}

- Cargo/Função*

- Início*

- Fim*

Formação

- Instituição

- Tipo de Formação* = {Curso, Graduação, Pós-Graduação, Especialização, MBA, Mestrado, Doutorado, Pós-Graduação}

- Nome do Curso*

- Término em*

Memorial Descritivo*

* *Campos Obrigatórios*

ED2 – Dados Registrados do Currículo

Candidato

- CPF
- RG
- Código [RNG8]
- Nome
- E-mail
- Sexo

Currículo

- Memorial Descritivo

Formação

- Instituição
- Curso
- Data de Término
- Tipo

Contato

- Número
- DDD
- Tipo

Experiência Profissional

- Cargo
- Instituição

- Data de Início

- Data de Fim

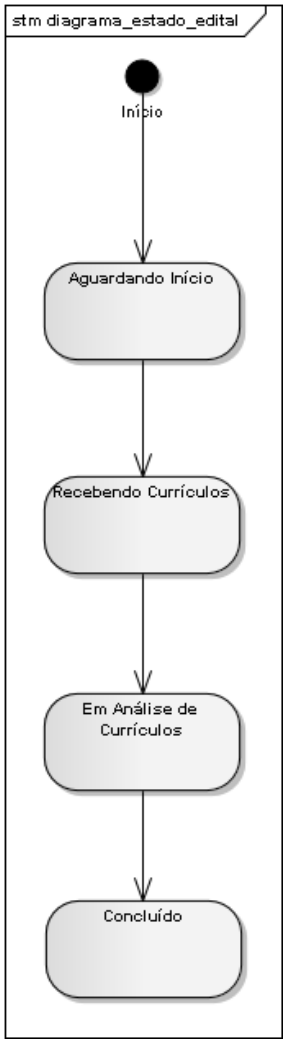
ED3 – Dados Registrados da Nota Automática do Currículo

- Nota

- Valor

- Tipo = “Técnica”

ANEXO V – Diagrama de Estados



ANEXO VI – Modelo de Classes