
Pós-Graduação em Ciência da Computação

**MIKE CHRISTIAN DE SOUSA ARAUJO**

# UMA ABORDAGEM ORIENTADA A ASPECTO PARA ESCRITA DE HISTÓRIA DO USUÁRIO COM GHERKIN

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

MIKE CHRISTIAN DE SOUSA ARAUJO

# UMA ABORDAGEM ORIENTADA A ASPECTO PARA ESCRITA DE HISTÓRIA DO USUÁRIO COM GHERKIN

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação do Centro de Informática da Universidade Federal de Pernambuco como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Henrique Emanuel Mostaert Rebelo

Recife
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

A663a Araujo, Mike Christian de Sousa
Uma abordagem orientada a aspecto para escrita de história do usuário com Gherkin / Mike Christian de Sousa Araujo. – 2017.
87 f.: il., fig., tab.

Orientador: Henrique Emanuel Mostaert Rebelo.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndice.

1. Engenharia de software. 2. Engenharia de requisitos. I. Rebelo, Henrique Emanuel Mostaert (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2018-076

**Mike Christian de Sousa Araujo**

# Uma abordagem orientada a aspecto para escrita de história do usuário com Gherkin

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 03 de março de 2017.

Aprovado em: 03/03/2017.

**BANCA EXAMINADORA**

---

Profª. Carla Taciana Lima Lourenço Silva Schuenemann
Centro de Informática / UFPE

---

Prof. Célio Andrade de Santana Júnior
Centro de Artes e Comunicação

---

Prof. Henrique Emanuel Mostaert Rebêlo
Centro de Informática / UFPE
(Orientador)

*Dedico este trabalho e todas as minhas conquistas à Deus e às minhas amadas mães que são minhas inspirações.*

# Agradecimentos

Às minhas mães Maria do Socorro Araújo Costa e Marcilene Tiago de Sousa por todo cuidado e ensinamentos que me passaram, por serem exemplos de mulheres guerreiras, honestas, perseverantes, por serem minhas mães, minhas inspirações.

Às minhas tias Gerlania, Gilvania e Giselda, ao meu pai Manoel Paixão Araujo Costa Neto, a minha irmã Mayara, ao meu primo Tomás e a toda minha família pelo incentivo e apoio demonstrado.

Aos amigos pela compreensão e apoio durante este período, ao Wanderson pela parceria de sempre e a família CAPAU pelo apoio que recebei de todos na reta final, em especial aos parceiros do AP104.

Agradeço ao meu orientador pela paciência, ensinamentos e pelo tempo dedicado a este trabalho. Agradeço, também, aos membros da banca por suas valiosas considerações feitas a esta pesquisa.

À minha pequena e eterna borboleta pelo carinho, cuidado, apoio e motivação recebidos durante esta jornada.

*“Sem motivação é difícil viver. . . ..”*
*(Validuaté)*

# Resumo

O entendimento das características do *software* e a criação de testes são importantes etapas na busca pela construção de sistemas de *software* com qualidade, confiança e que atenda as expectativas do usuário. O *Behavior Driven Development* (BDD) se utiliza da notação *Gherkin* (*Given-When-Then*) para escrever testes de aceitação por meio da criação de histórias e cenários. No entanto, a escrita de bons testes depende do entendimento dos interesses levantados pelos usuários para o sistema e como estes estão conectados. Neste sentido, o presente trabalho buscou investigar o uso de *Aspect-Oriented Requirements Engineering* (AORE) como uma iniciativa para auxiliar nas especificações de testes baseados em histórias do usuário, como acontece em BDD, e, assim, apresentar uma abordagem com etapas definidas para identificar e modularizar interesses transversais durante a escrita dos testes criados com a notação *Gherkin*, e possibilitar que os testes escritos reflitam as várias ligações existentes entre os requisitos. Para isso, foi realizado um levantamento na literatura para obter conhecimento teórico-científico sobre a aplicação de Orientada a Aspecto na Engenharia de Requisitos e na identificação das características dos modelos propostos de aplicação de AORE. Com base nos conceitos e prática do BDD e das abordagens genéricas de aplicação dos conceitos de orientação a aspecto na especificação de requisitos, foi definido o passo a passo que compõe a abordagem proposta no trabalho. A abordagem foi aplicada no exemplo demonstrativo *Automated Teller Machine* (ATM) para ilustrar as etapas da abordagem na especificação dos testes criados com a notação *Gherkin*. Assim, este trabalho é uma tentativa de estabelecer uma abordagem que possa auxiliar na especificação dos testes criados com a notação *Gherkin* para identificar e modularizar interesses transversais.

Palavras-chave: História do Usuário. Engenharia de Requisito. Orientação a Aspecto. *Gherkin*.

# Abstract

The software characteristics understanding and creating tests are important steps in the search for the software systems construction with quality, confidence and that meets user expectations. Behavior Driven Development (BDD) uses Gherkin (Given-When-Then) notation to write acceptance tests by creating stories and scenarios. However, writing good tests depends on understanding the interests raised by users for the system and how they are connected. In this sense, this work investigated the use of Aspect-Oriented Requirements Engineering (AORE) as an initiative to assist in tests specification based on user stories, as in BDD, and thus present an identify and modularize crosscutting concerns during tests writing created with the Gherkin notation, and enable the written tests to reflect various links between requirements. For this, a survey was carried out in the literature to obtain theoretical-scientific knowledge about Aspect Oriented in Requirements Engineering application and models characteristics identification of proposed AORE application. Based on the concepts and BDD practice, and generic approaches of orientation concepts application to aspect in requirements specification, the step-by-step that composes the approach proposed in the work was defined. The approach was applied in the Automated Teller Machine (ATM) demonstrative example to attest approach steps in specifying the tests created with the Gherkin notation. Thus, this work is an attempt to establish an approach that can aid in tests specification created with Gherkin notation to identify and modularize crosscutting concerns.



Keywords: User Stories. Requirement Engineering. Aspect Orientation. Gherkin.

# Lista de ilustrações

# Lista de quadros

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| AORE | Aspect-Oriented Requirements Engineering |
| ATDD | Acceptance Test Driven Development |
| ATM | Automated Teller Machine |
| BDD | Behaviour Driven Development |
| DDD | Domain Driven Design |
| OA | Orientação a Aspecto |
| OO | Orientação a Objetos |
| POA | Programação Orientada a Aspectos |
| RCT | Requirements Composition Table |
| TCH | Tabela de Composição das Histórias |
| TDD | Test-Driven Development |
| TIT | Tabela de Interesses Transversais |

# Sumário

# 1 INTRODUÇÃO

A busca pela construção de sistemas de *software* com qualidade é algo recorrente nas pesquisas sobre engenharia de *software*, seja com vistas na qualidade do produto final, ou por melhorias durante o processo de desenvolvimento. Desse modo, o entendimento sobre as características que o *software* possuirá é uma importante etapa do processo de construção.

É importante que no processo de construção os usuários descrevam as características funcionais e não funcionais desejadas para o sistema, de modo que tais informações fiquem claras, livres de interpretações múltiplas, fáceis de entender, completas e consistentes(PRESSMAN, 2011) .

São muitos os envolvidos neste processo de implementação das características do *software*, com visões diferentes, demandando assim, atenção especial nesta etapa inicial, na qual valida-se o esforço para melhorar a identificação e entendimento de tais características. Sendo importante ressaltar, que a identificação preliminar das principais características, relacionadas as funcionalidades ou a qualidade, é essencial para produzir algo dentro das expectativas dos envolvidos no projeto (SOMMERVILLE, 2011).

Outra importante etapa é a realização de testes, decorrente da tentativa de evitar ou diminuir significativamente os defeitos (*bugs*, inconsistência nos dados, entre outros) e possibilitar um entendimento nítido das funcionalidades que estão sendo ou serão criadas, proporcionando mais qualidade e confiabilidade ao sistema que será construído. (RESTIVO, 2015) aborda que o teste de *software* é uma das atividades mais importantes do desenvolvimento, visto que, por meio desses, as partes interessadas (*stakeholders*), conseguem avaliar a qualidade do produto desenvolvido, seja pelo incremento da confiança no código, ou por meio da identificação de erros para posteriores correções.

Assim, a procura para atender as necessidades qualitativas na construção de *software* e de adaptar as novas situações nesse processo de construção, considerando as multiplicidades de agentes envolvidos, tem-se o surgimento de metodologias, ferramentas e técnicas de desenvolvimento, como o *Behavior Driven Development* (BDD) e a *Aspect-Oriented Requirements Engineering* (AORE).

O BDD é uma metodologia de desenvolvimento de *software* recente, de boa aceitação na comunidade, que busca especificações executáveis de um sistema por meio de testes claramente escritos e de fácil entendimento, estabelecendo uma linguagem ubíqua entre *stakeholders* e a equipe de desenvolvimento, de modo a auxiliar na especificação dos testes (NORTH, 2014). Smart (2015) destaca que a utilização das ferramentas de BDD pode ajudar no processo de transformação dos requisitos

em especificações executáveis de testes, auxiliando o desenvolvedor a verificar os recursos e a gerar documentação do *software*.

Já o paradigma da orientação a aspecto surge fortemente como uma alternativa para encapsulamento de interesses na etapa de codificação, passando nos últimos anos a ser pensado também em outras etapas do desenvolvimento como na engenharia de requisitos, na qual a separação de interesse é aplicada (MOREIRA *et al.*, 2013). Vários trabalhos abordam sobre a importância de serem identificadas tais interesses, antes mesmo da implementação do *software*, ou seja, no início da sua concepção, no entendimento dos requisitos.

Jacobson e Ng (2004 *apud* SOMMERVILLE, 2011) reforçam que os interesses e exigências do sistema são derivados das prioridades, expectativas e exigências das partes interessadas no sistema, pois, o desempenho do sistema, a facilidade de manutenção, por exemplo, podem ser interesses levantados como desejados de alguns *stakeholders*.

Na AORE a aplicação dos conceitos de orientação a aspecto busca identificar e analisar interesses transversais (crosscutting concerns), aspectos, nas fases inicias do desenvolvimento, com o intuito de entender seus potencias efeitos em fases posteriores. Sommerville (2011) destaca que a programação tem sido um dos focos das pesquisas em orientação a aspectos. No entanto, considera que os interesses transversais são igualmente importantes em outras fases do processo de desenvolvimento do *software*. Logo, as pesquisas têm-se voltado para investigar a utilização da orientação a aspecto na engenharia de requisitos e no *design* dos sistemas, assim como nas etapas de teste, verificação e validação, de programas orientados a aspecto.

A forma como os requisitos são entendidos pode impactar seriamente no desenvolvimento das funcionalidades e na criação dos testes (MITCHELL; BLACK, 2015). Neste sentido, a escrita das especificações dos testes, com base no levantamento dos requisitos do usuário, apresenta-se como um desafio, na medida em que é preciso a utilização de novas técnicas com critérios bem definidos para a especificação de testes com base nos requisitos do usuário.

As especificações dos testes devem representar bem a funcionalidades do sistema e como elas estão relacionadas, de modo que fique claro estas relações e como uma funcionalidade pode interferir na execução de outras, ou necessitar da execução de outras partes do sistema.

## 1.1 Caracterização do problema

Na prática, conseguir ter um bom entendimento dos requisitos do *software* não é uma tarefa fácil, pois cada parte interessada possui sua interpretação do que deseja,

podendo até existir conflitos entre as características levantadas por cada usuário (SOMMERVILLE,2011).

O BDD se utiliza da notação *Gherkin* (*Given-When-Then*) para escrever testes de aceitação por meio da criação de histórias e seus respectivos cenários, proporcionando uma linguagem simples e de fácil entendimento por todos os participantes do processo (NORTH, 2014).

Mitchell e Black (2015) afirmam que se deve tentar mitigar os riscos existentes à qualidade do produto (*software*) antes da execução dos testes e que isso deve ser feito por uma gama de atividades. Afirmam ainda que nos casos em que requisitos não estejam bem escritos, deve-se melhorar a qualidade destas informações antes de executar os testes, pois, caso contrário, isto geraria num último nível código ruins.

É essencial identificar e entender todos os interesses levantados pelo os usuários para o sistema e como estes estão conectados. Realizar uma modularização de interesses transversais (aspectos), neste momento, é importante, pois se não realizada tais interesses serão refletidos para outras etapas (MOREIRA et al., 2013). Além disso, lidar com aspectos nos requisitos permite que pontos críticos do sistema sejam determinados no início do desenvolvimento.

Sendo assim, a preocupação de modularizar os interesses também deve existir no momento das especificações das histórias e de seus respectivos cenários criados com notação *Gherkin* no BDD, visto que estas histórias representam as funcionalidades e os cenários os comportamentos que foram descritos pelos usuários, que se tornarão especificações executáveis, ou seja, especificações testáveis.

Assim, o uso de AORE pode ser uma boa iniciativa para especificações de testes baseados em histórias de usuário, como acontece em BDD, possibilitando que os testes escritos reflitam as várias ligações existentes entre as funcionalidades levantadas pelo usuário. Além disso, permite que isso seja visto o quanto antes, ainda na especificação textual e não somente na codificação dos testes.

Neste sentido a questão central deste trabalho é investigar se AORE pode auxiliar na especificação dos testes criados com a notação *Gherkin* (Given/When/Then) para identificar interesses transversais.

## 1.2 Objetivos

O Objetivo geral deste trabalho é investigar a aplicação dos conceitos de AORE para auxiliar na especificação dos testes criados por meio da escrita de histórias e seus respectivos cenários com a notação *Gherkin* em busca de identificar e modularizar interesses transversais. Como forma de atingir o objetivo geral, são estabelecidos os seguintes objetivos específicos:

- Fazer o levantamento da literatura para investigar como AORE está sendo aplicada;
- Reunir as principais características dos modelos e técnicas encontrados de aplicação de AORE na literatura;
- Apresentar uma abordagem com etapas definidas para identificar e modularizar interesses transversais durante a escrita dos testes criados com a notação *Gherkin*;
- Demonstrar a aplicação da abordagem com um exemplo demonstrativo.

## 1.3 Metodologia

O método de pesquisa para atingir os objetivos pretendidos está dividido em três etapas conforme a Figura 1: Levantamento da Literatura, Proposta da Solução e Demonstração da Solução.


**Figura 1 – Método de pesquisa**

Fonte: Autor

**Etapa 1. Levantamento literário**

A fim de agregar conhecimento teórico-científico sobre a temática pesquisada, a primeira etapa desse trabalho foi realizar o levantamento exploratório sobre a aplicação de Orientação a Aspecto na Engenharia de Requisitos, com vistas na identificação das principais características apresentadas pelos modelos e técnicas da aplicação de AORE no processo de construção de um *software*.

Por meio de pesquisa da literatura específica com abordagem dos temas, buscou-se trabalhos já realizados, que pudessem contribuir na identificação e entendimento das características e vantagens da aplicação de orientação a aspecto nesta fase inicial do desenvolvimento do *software* (fase dos requisitos). Assim como identificar os pontos comuns estabelecidos pelos modelos e abordagem propostos de AORE, de modo que permita verificar se AORE pode auxiliar na especificação dos testes criados com a notação *Gherkin* para identificar interesses transversais.

Para realizar a pesquisa foi utilizado motores de busca automáticos (*IEEExplore, ACM Digital library, Science Direct, Springer*) e buscas manuais em eventos nacionais, como, por exemplo, Congresso Brasileiro de Software, Simpósio Brasileiro de Engenharia de Software e Workshop em Engenharia de Requisitos , e em eventos internacionais, como, por exemplo, *International Conference on Aspect-Oriented Programming* e *International Conference on Aspect-Oriented Software Development*. Os eventos e motores de busca escolhidos representam fontes relevantes para computação na área de engenharia de *software*.

Na procura por trabalhos nos buscadores automáticos foram utilizados os termos: "*Requirements Engineering* ou *Software Requirements* ou *Software Requirements Specification* ou *Requirements*" e "*Aspect Oriented Programming* ou *Aspect Orientation* ou *Early Aspects* ou *Aspect-Oriented Requirements*"e "*Behavior Driven Development* ou BDD" e "*Software Testing* ou *Acceptance Testing* ou *Aspect-Oriented Testing*".

**Etapa 2. Proposta da solução**

Essa etapa consistiu em definir o passo-a-passo que compõe a abordagem proposta no trabalho, para auxiliar na especificação dos testes criados com a notação *Gherkin*. As etapas da abordagem estão embasadas nos conceitos e prática do BDD e abordagens genéricas de aplicação dos conceitos de orientação a aspecto na especificação dos requisitos conforme ilustrado na Figura 2.

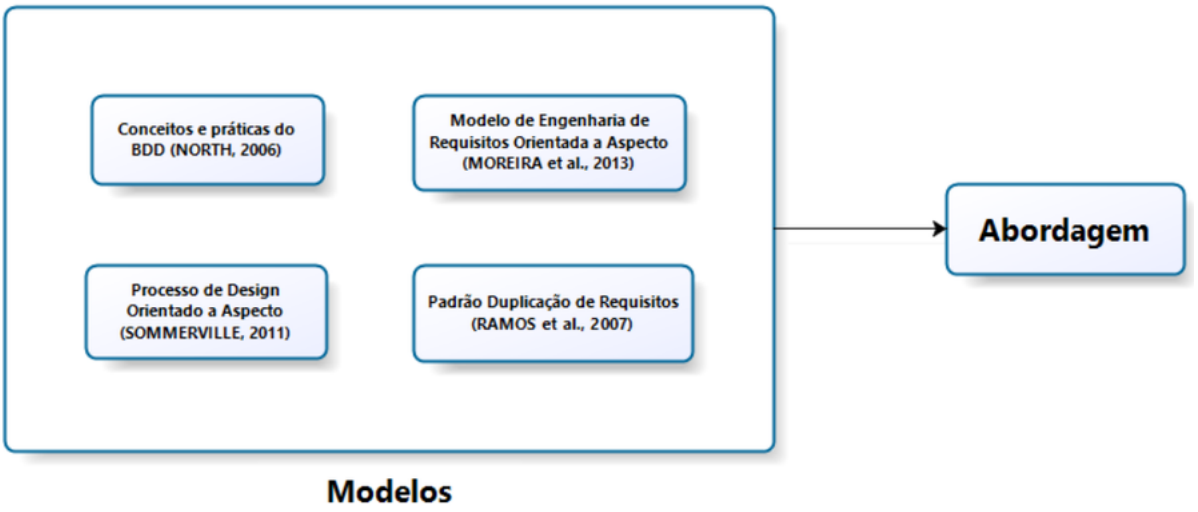


**Figura 2 – Modelos que compõem a abordagem proposta**

Fonte: Autor

Os modelos escolhidos definiram processos claros e independentes, fato que possibilitou um estabelecimento claro da abordagem de como proceder para identificar e modularizar interesses transversais na especificação dos testes criados com a notação *Gherkin*.

**Etapa 3. Demonstração da solução**

Considerando o uso comum na literatura selecionou-se o exemplo demonstrativo desse trabalho para aplicação dos conceitos de orientação a aspecto. Além disso, contribuiu para o uso desse exemplo, a presença de interesses transversais na sua documentação, o fato de o exemplo está modelado no paradigma Orientado a Objetos (OO), ou seja, com algumas modularizações aplicadas e sua documentação completa e acessível no endereço eletrônico: http://www.math-cs.gordon.edu/courses/cs211/ATMExample/index.html.

A abordagem foi aplicada no exemplo demonstrativo Automated Teller Machine (ATM) conforme ilustrado na Figura 3. De posse da documentação do exemplo escolhido, as etapas da abordagem foram aplicadas para demonstrar o uso da abordagem na especificação dos testes criados com a notação *Gherkin*.

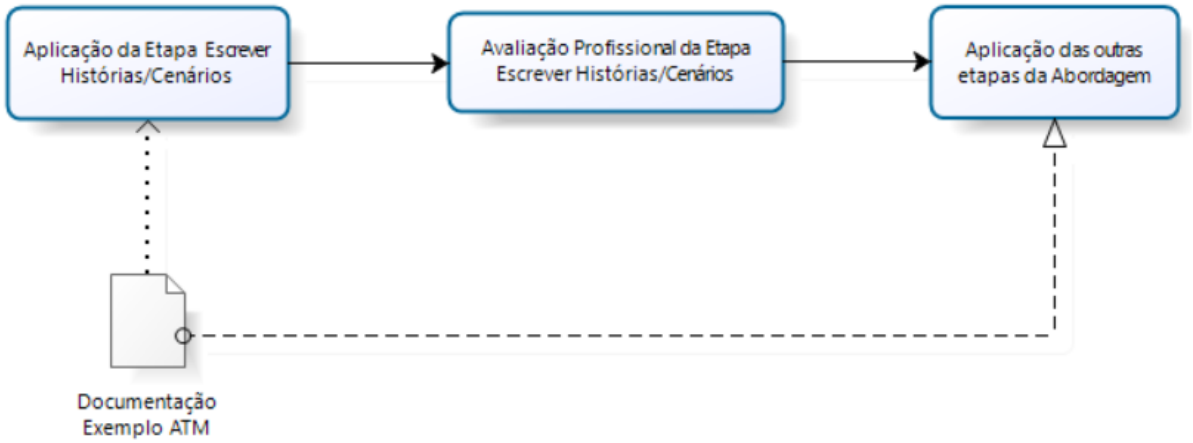


**Figura 3 – Fluxograma da aplicação da abordagem no exemplo demonstrativo**

Fonte: Autor

Ressalta-se que como forma de reafirmar que os testes foram escritos corretamente, a primeira etapa da abordagem passou por avaliação de dois profissionais experientes na área de testes e, também, com experiência em trabalhar com a metodologia ágil BDD e que, portanto, com conhecimentos técnicos e práticos na especificação de testes.

## 1.4 Organização do trabalho

O trabalho está organizado em 5 capítulos, além da introdução, os citados a seguir:

- Capítulo 2: São apresentados os conceitos necessários para o entendimento da abordagem estabelecida neste trabalho;
- Capítulo 3: É descrita a abordagem estabelecida neste trabalho para identificar e modularizar interesses transversais durante a escrita dos testes criados com a notação *Gherkin;*
- Capítulo 4: É utilizado um exemplo demonstrativo para aplicar as etapas definidas pela abordagem proposta neste trabalho;
- Capítulo 5: São feitas as considerações finais, bem como são ressaltadas as contribuições, limitações e direcionamentos para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Serão apresentados os conceitos necessários para o entendimento da abordagem estabelecidade neste trabalho. São conceitos sobre orientação a aspecto e sua aplicação nos requisitos (Seção 2.1), sobre testes de aceitação e as características da especificação desse tipo de teste (Seção 2.2) e sobre as características da metodologia ágil *behaviour driver development* (Seção 2.3).

## 2.1 Orientação a aspecto

Complexidade é uma característica presente no processo de desenvolvimento de *software*. Requisitos complexos de *software* precisam ser divididos em partes menores, e modo que representem as suas várias características, funcionalidades. Estas funcionalidades são conhecidas como interesses principais do sistema quando representam as funcionalidades mais importantes do sistema e interesses transversais quando representam interesses como segurança, controle de concorrência e gerenciamento de transações que perpassam por outras partes (funções) do sistema (LADDAD, 2010), com ilustrado na Figura 4.

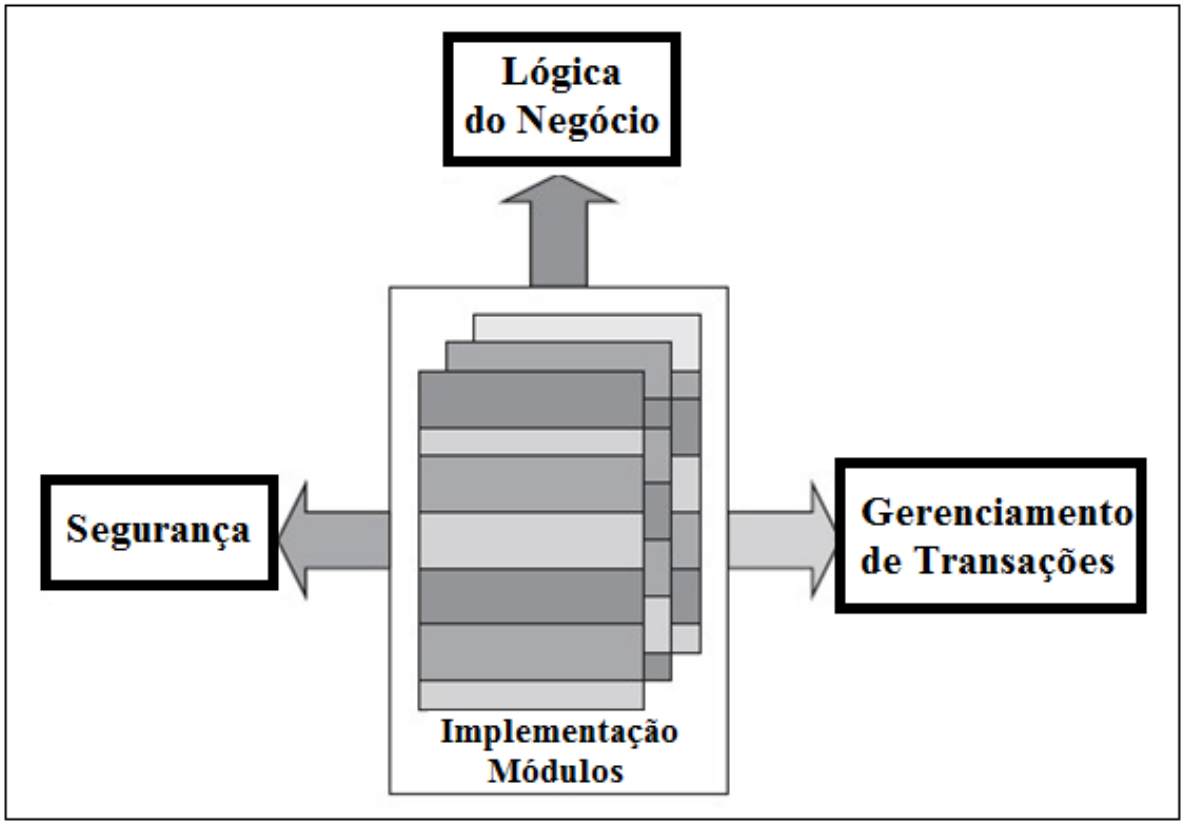


**Figura 4 – Visão de um sistema com vários interesses.**

Fonte: Laddad (2010)

O paradigma da orientação a aspecto (OA) surge como alternativa para ajudar a resolver problemas antigos de modularização de *software*, de forma a complementar o paradigma orientado a objetos na resolução de situações que envolvam separação de interesses (KICZALES, 1997).

Dentre as características importantes que se pode citar deste paradigma é que os interesses transversais passam a ser pensados em unidades únicas (aspectos) que reúnem característas específicas em uma só unidade lógica. Isso permite aumentar a possibilidade de reuso, de modularidade (separação dos interesses em partes específicas) e de coompreensão dos requisitos do sistema (KUMAR, 2016).

Sommerville (2011) propõe um *generic aspect-oriented design process* na qual define etapas para que um sistema seja projetado para suportar as funcionalidades principais e requisitos relacionados a qualidade de serviço, desempenho e confiabilidade, com identificação e separação dos aspectos, como estes irão compor o sistema central e uma análise de possíveis conflitos dessa composição, além de propor a definição de padrões para nomear as entidades do sistema (Figura 5).

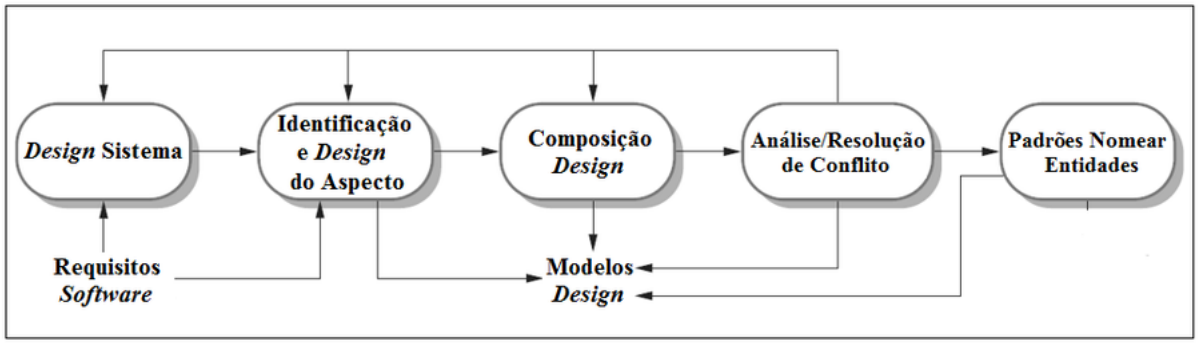


**Figura 5 – Processo genérico de *design* orientado a aspecto.**
Fonte: Sommerville (2011)

Segundo Kiczales (1997) a orientação a aspecto permite trabalhar separadamente os aspectos ortogonais que possam existir na construção de um sistema, possibilitando que cada aspecto de interesse do sistema seja trabalhado isoladamente, o que facilita a implementação de tais descrições em separados. É importante ressaltar que o paradigma da orientação a aspecto não surge para substituir nenhum outro, como por exemplo a orientação a objetos, mas deve ser pensado como um paradigma complementar, podendo ser aplicado em outros paradigmas pela suas características genéricas.

A ideia é proporcionar um melhor entendimento das características do sistema, visto que tais características, aspectos, passam a estar isoladas e, assim, permitindo um melhor tratamento e compreensão das funcionalidades que o sistema irá possuir. Além de facilitar o tratamento de mudanças nas características do sistema que possam surgir no decorrer do processo de sua construção.

### 2.1.1 Orientação a aspectos nos requisitos

No processo de desenvolvimento de um sistema, um interesse representa as funcionalidades, comportamentos, ou qualquer outra característica que seja levantada ou considerada relevante pelas várias pessoas que contribuem para a construção do sistema (RESTIVO, 2015).

Os requisitos que fazem parte de um sistema são fornecidos por uma variedade de pessoas que estão envolvidas (*stakeholders*) no processo de levantamento dessas informações e que possuem visões particulares sobre o sistema. Estes *stakeholders* fornecem vários tipos de interesses (SOMMERVILLE, 2011):

- Interesses funcionais: requisitos relacionados com uma determinada funcionalidade que faz parte do sistema;
- Interesses de qualidade do serviço: os comportamentos não-funcionais de um sistema, como por exemplo: desempenho, confiabilidade e disponibilidade;
- Interesses políticos: interesses que se referem as políticas globais estabelecidas na utilização de um sistema, como interesses que envolvam acesso restrito as partes do sistema e que estejam relacionados às regras de negócios;
- Interesses do sistema: informações relacionadas aos atributos do sistema como sua capacidade de manutenção ou sua configurabilidade;
- Interesses organizacionais: relacionados ao cumprimento de metas (construir o sistema dentro do orçamento) e prioridades organizacionais (manter a reputação da organização).

É importante reconhecer que os requisitos que compõem o sistema são variados e que representam as diferentes visões, dos diferentes interessados. Pensar numa abordagem para separar os requisitos em partes diferentes, irá facilitar o entendimento do que cada requisito representa para o sistema e como eles estão relacionados.

Ramos *et al*. (2007) propõe um Padrão para Requisitos Encapsulados como uma solução independente e que pode ser instanciada para qualquer abordagem que produza um documento de requisitos com objetivo de eliminar duplicações que possam ocorrer durante a explicitação dos requisitos.

A solução proposta pelos autores define que se deve identificar e analisar os requisitos duplicados, criar uma nova estrutura para os encapsular, depois removê-los da estrutura original, verificar se as funcionalidades não foram afetadas com a remoção e atualizar as referências de dependência, se houver necessidade (Figura 6).

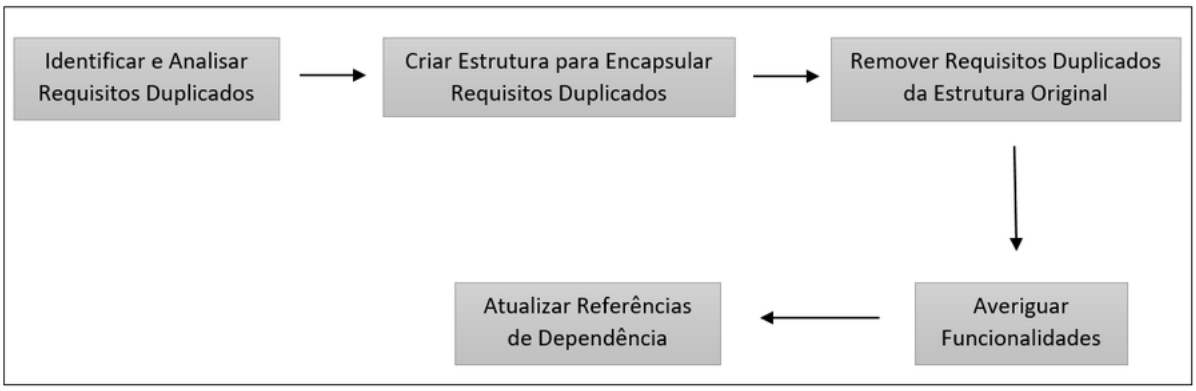


**Figura 6 – Etapas do padrão de requisitos encapsulados**

Fonte: Ramos (2007)

É bem comum que a visão modular do sistema seja pensada na etapa de implementação do sistema, na codificação. No entanto, ela também pode ser aplicada na fase inicial do sistema com objetivo de identificar requisitos que possam ter suas características transversais.

A discussão sobre a aplicação de orientação a aspecto na engenharia de requisitos teve início no congresso de Engenharia de Requisitos em Lymerick na Irlanda, em 1999, vindo a surgir, posteriormente, a área de estudo *Early Aspects* que tem um olhar voltado para gerir interesses transversais nos momentos iniciais do desenvolvimento, desde a engenharia de requisitos até o *design* de arquitetura (TEKINERDOğAN *et al.*, 2005).

O objetivo de aplicar os conceitos de orientação a aspecto, incialmente pensado para a programação, é verificar a existência de interesses transversais o quanto antes, assim como estes devem ser modularizados (aspectos) e posteriormente interligados, evidenciando uma clara separação dos interesses e das influências que podem gerar nas outras partes do sistema (MOREIRA *et al.*, 2013).

Com essa modularização, os impactos gerados a outros requisitos, por mudanças em um requisito modularizado fica mais visível, facilitando o mapeamento desses impactos para a implementação ou para a arquitetura do sistema, por exemplo.

Moreira *et al*. (2013) define um *general aspect-oriented requirements engineering framework* que apoia o mapeamento e influência dos aspectos encontrados no levantamento de requisitos, para as partes posteriores do desenvolvimento do *software* (Figura 7).

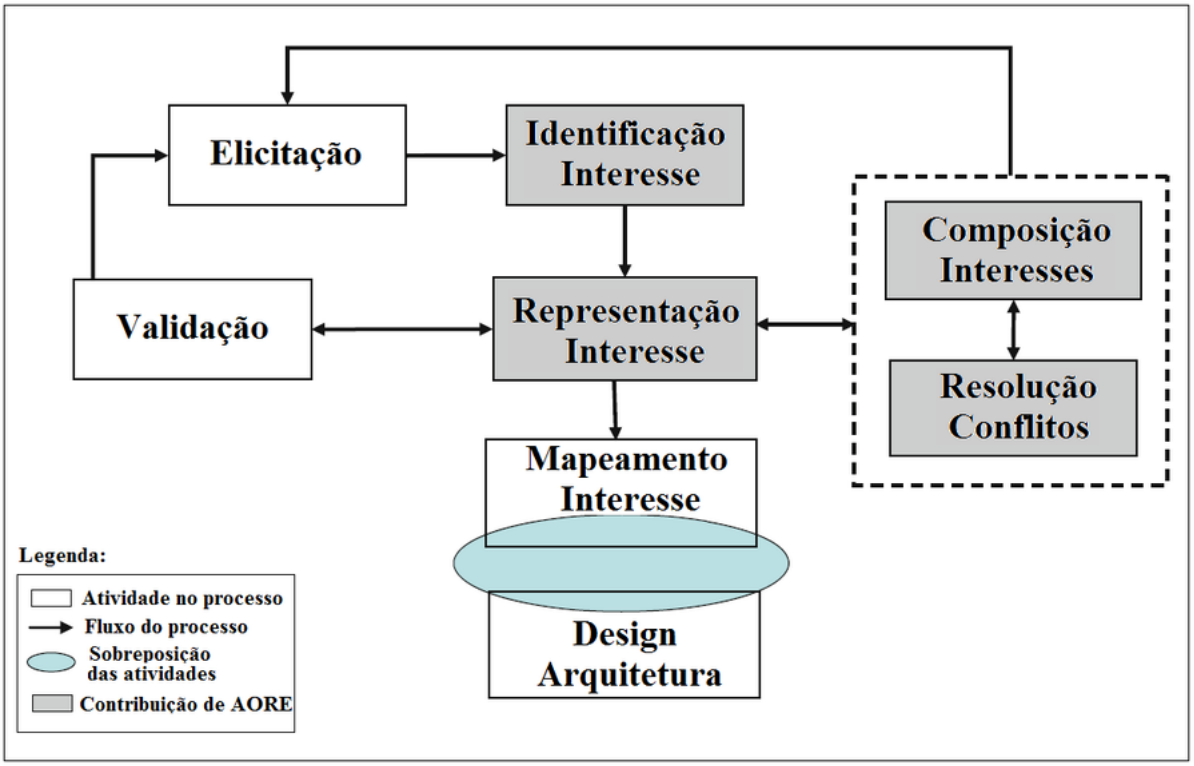


**Figura 7 – Framework genérico para engenharia de requisitos orientada a aspecto**

Fonte: Moreira (2013)

Conforme a estrutura geral de AORE definida pela autora, as principais contribuições de AORE é na identificação de interesses transversais e suas representações, na definição de composição desses interesses e na resolução de conflitos que possam existir. Além disso, tais interesses devem ser mapeados para o ciclo de desenvolvimento de um sistema de *software*.

### 2.1.2 Separação de interesses

O ato de se pensar na modularização das características dos sistemas durante a sua construção não é algo novo. Isso já se pensava desde da década de 50 com o encapsulamento das funcionalidades em unidades através de sub-rotinas, com a utilização de mecanismos como os procedimentos e o surgimento do paradigma de orientação a objetos na década de 60 (SOMMERVILLE, 2011).

O problema é que todas essas soluções possuem alguma dificuldade em trabalhar com um certo tipo de interesse que existe quando se está no processo de construção de um sistema: os que são transversais (Figura 8). Quando se realiza o processo de modularização, pode ocorrer que alguns interesses não consigam se alinhar com o critério usado, gerando problemas conhecidos como emaranhamento e espalhamento (RESTIVO, 2015).

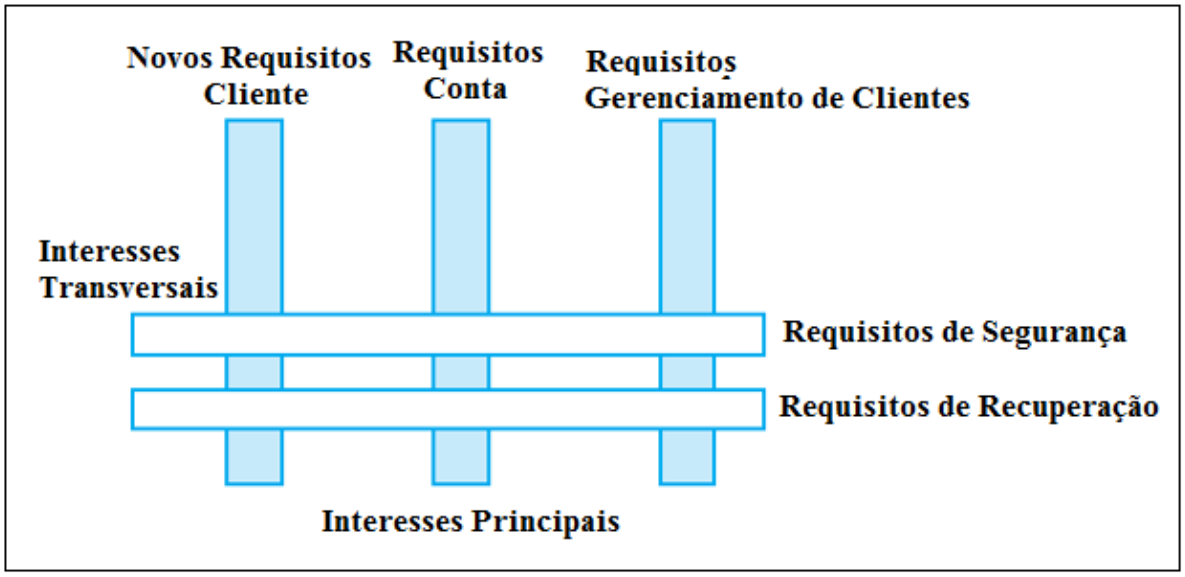


**Figura 8 – Exemplo de interesses transversais.**

Fonte: Restivo (2015)

Quando um interesse se encontra espalhado e entrelaçado com outros interesses, aquele é considerado um aspecto, e, portanto, deve ser modularizado. O que caracteriza um interesse espalhado é quando um interesse está pertencendo a vários outros módulos, ou seja, tendo as suas características e comportamentos dispersos em outros módulos, e o emaranhamento é quando um mesmo módulo está resolvendo preocupações de outro módulo, ou seja, existe interesses entrelaçados neste módulo (Figura 9) (RESTIVO, 2015).

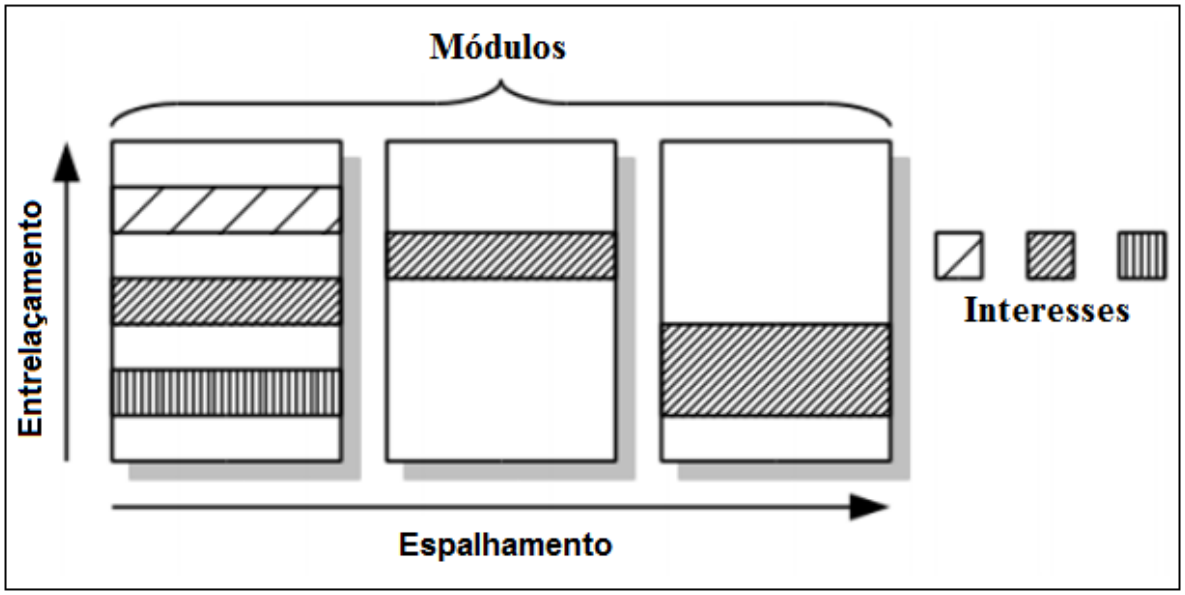


**Figura 9 – Espalhamento e entrelaçamento de interesses.**

Fonte: Restivo (2015)

Na programação orientada a aspectos (AOP) é considerado que um interesse é implementado de forma modular quando todas as informações (código) que o representam estão contidas em um só lugar, quando a forma de interação (*interfaces*) com as outras partes do sistema está bem definida, de modo que é possível fazer modificações nas características do interesse sem que suas *interfaces* sejam violadas e que também não haja violação às demais *interfaces* dos outros módulos, e, além disso, quando a união destes interesses modularizados sejam capazes de formar o sistema como um todo (KICZALES; MEZINI, 2005).

Este processo de modularização dos interesses pertencentes a um sistema, em partes menores e claramente definidos, deve envolver as seguintes etapas (SILVA, 2007):

- Identificação de interesses: levantamento dos interesses que o sistema vai incluir;
- Representação dos interesses separadamente: separar interesses em unidades que reflitam uma determinada característica do sistema, ou um conjunto de unidades que representem tais características;
- Composição de interesses: definir como as unidades separadas irão interagir para formar o sistema, ou seja, como serão interligadas.

Observa-se que tão importante quanto identificar e separar os interesses transversais, é definir como esses interesses irão compor as outras partes do sistema, ou seja, como será estabelecida a comunicação entre as partes do sistema. Isso irá permitir que o sistema seja facilmente entendido, pois está divido em vários módulos, sendo que cada um é responsável por realizar a implementação de um interesse do sistema, possibilitando que novos interesses ou futuras modificações nos interesses já existentes, sejam mais fáceis de implementar, pois estarão isolados e poderão ser vistos de forma independente, contribuído para uma melhor coesão e um menor acoplamento dos módulos que compõem o sistema.

Assim, orientação a aspecto introduz o conceito de aspectos para representar os interesses que estão espalhados e misturados com outros interesses do sistema, com vista a ajudar no entendimento e modularização das características que um sistema possui. Portanto, é importante entender os vários conceitos que formam o paradigma da orientação a aspecto para a sua utilização.

### 2.1.3 Aspetos, joinpoints e pointcuts

Serão apresentados alguns dos conceitos de OA que são essenciais para o seu entendimento e aplicação. Os conceitos apresentados apesar de advindos da programação, não são restritos a esta etapa, são explicações genéricas dos principais conceitos de OA, podendo ser aplicados independente da etapa que se esteja trabalhando, seja nas primeiras fases ou na implementação.

A programação orientada a aspecto introduziu o termo aspecto para representar o encapsulamento dos interesses transversais em unidades separadas, estabelecendo uma forma de implementação que pode ser aplicada em vários locais diferentes ao longo do código, sem que haja mistura de interesses e, assim, evitando "sujeiras" na codificação (RESTIVO, 2015).

Os aspectos possuem uma característica diferente dos outros tipos de abstrações, pois o próprio aspecto possui uma especificação de onde e como ele será executado. Diferente das abstrações de OO, por exemplo, em que observando a declaração de um método não é possível saber em quais pontos do *software* ele precisar ser invocado. Já os aspectos, possuem uma declaração que define onde o aspecto será integrado (tecido) no *software* (SOMMERVILLE, 2011).

Para que os aspectos possam ser invocados, é necessário que alguns pontos específicos sejam identificados nos outros módulos (MOREIRA *et al.*, 2013). O Aspecto só poderá ser composto mediante estes pontos de ligações (*joinpoints*). Os *joinpoints* são os pontos de encontro, nos outros módulos, de um ou mais interesses modularizados (aspecto). Na AORE se pode citar como exemplos de *joinpoints* bem definidos os *goal notes*, ou interesses identificáveis por identificações únicas (ID), ou ainda através de uma identificação gramatical ou semântica nas partes do texto (MOREIRA *et al.*, 2013).

Para que um aspecto possa ser "combinado" com as outras partes do sistema é preciso definir um *pointcut*. Este irá definir através de um conjunto de *joinpoints* como um aspecto irá compor os outros requisitos (FASSBENDER; HEISEL; MEIS, 2015).

Na programação Orientada a Aspecto é comum dizer que um aspecto aconselha (*Advice*) outros interesses. Ou seja, um *advice* é a implementação do aspecto, é a forma como este irá interferir, se manifestar, no comportamento dos interesses não aspectuais, por meio da captura dos locais onde ele irá afetar (*pointcut*) seguindo uma ordem temporal, condicional ou incondicional (MOREIRA *et al.*, 2013)**.** A seguir, no Quadro 1, é apresentado a descrição desses principais conceitos (SHESHASAAYEE; JOSE, 2015)**.**

| TERMO | DEFINIÇÃO |
|---|---|
| *Aspect* | Uma estrutura para representar um interesse transversal. |
| *Joinpoint* | Um ponto de interesse em algum artefato do ciclo de vida do software através do qual dois ou mais interesses podem ser compostos. |
| *Pointcut* | Uma regra usada para capturar os *joinpoints* de outros interesses. |
| *Advice* | O comportamento do aspecto a ser executado quando um *joinpoint* é capturado. |

Quadro 1 – Descrição dos principais conceitos de OA

Assim, na engenharia de requisitos orientada a aspecto, um aspecto é um requisito transversal (*cross-cutting*), que passa a ser entendido como um elemento que corta outros requisitos e que deve ser pensado separadamente. Isso leva a uma clara distinção entre os interesses transversais e os interesses principais que compõem o sistema. Para combinar os requisitos *cross-cutting* com os demais requisitos é preciso definir as regras de capturas dos *joinpoints* que irão permitir esta integração (FASSBENDER; HEISEL; MEIS, 2015).

Os conceitos pertencentes a orientação a aspecto, quando usados na AORE, podem ser aplicados com diferentes interpretações, dependendo da abordagem utilizada (MOREIRA *et al.*, 2013). O certo é que qualquer um desses conceitos pode ser introduzido na engenharia de requisitos, e que o uso dessas noções representa a presença de uma perspectiva orientada a aspecto nessa etapa .

## 2.2 Teste de software

Neste capítulo serão apresentados os conceitos suficientes sobre teste de aceitação para o entendimento deste trabalho.

A realização de teste é importante na atividade de desenvolvimento de *software,* pois visa a tentativa de evitar ou pelo menos diminuir a quantidade de defeitos (MITCHELL; BLACK, 2015), além de possibilitar um entendimento mais claro das funcionalidades que serão ou que estão sendo criadas. E, assim, gerar mais qualidade e confiabilidade ao sistema construído.

A etapa de teste pode ser entendida como uma forma de averiguar erros e realizar suas correções, assim como pode ser uma forma de passar mais credibilidade as pessoas envolvidas no projeto de que as funcionalidades estão sendo codificadas da forma correta (RESTIVO, 2015).

A ideia desta etapa é vasculhar possíveis erros nas funcionalidades, nos códigos, de modo que se possa levantar o máximo de possibilidades, comportamentos, que façam aquele código sair do fluxo esperado e assim gerar um erro (RESTIVO, 2015). Logo, o sucesso de um teste é quando ele revela uma falha. Assim, o processo de realização dos testes é um processo em que todos as partes interessadas avaliam a qualidade do produto desenvolvido.

Os testes são geralmentes categorizados como testes de caixa preta e de caixa branca (RESTIVO, 2015). Quando, para realizar o teste, se faz necessário o conhecimento do funcionamento interno do código está se realizando um teste de caixa branca, que tem como objetivo encontrar os erros no código que não foram vistos durante o entendimento dos requisitos e que geralmente são causados por uma má interpretação dos requisitos. Já os testes de caixa preta é quando as funcionalidades são testadas, sem conhecimento do seu funcionamento interno, para verificar se estão exercendo o que foi estabelecido na especificação dos requisitos.

Os testes não são feitos apenas na finalização do desenvolvimento ou de cada módulo que o compõe, mas sim durante todo o processo de desenvolvimento do mesmo. Para isso, existe vários tipos de testes com objetivos diferentes e que estão relacionados a cada fase do desenvolvimento: teste de aceitação, teste de sistema, teste de integração e teste de unidade. Isso permite que a atividade de teste possa ser realizada em vários níveis (Figura 10) (SOMMERVILLE, 2011).

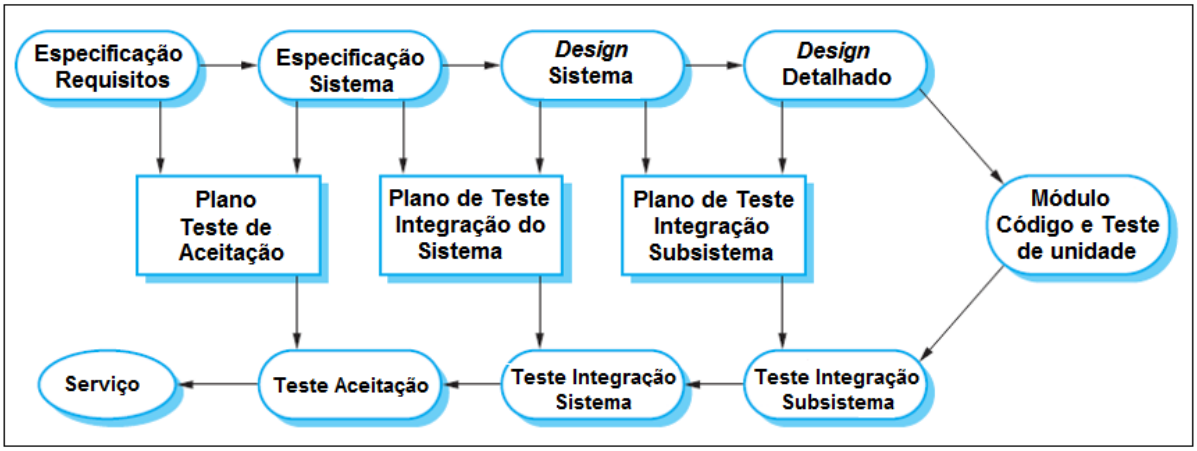


**Figura 10 – Distribuição das atividades de teste no desenvolvimento.**
Fonte: Sommerville (2011)

### 2.2.1 Teste de aceitação

Como se pode observar na Figura 10, os testes de aceitação são teste que estão ligados as fases de levantamento de requisitos e especificação do sistema. Testes de aceitação são testes realizados com base nas especificações das funcionalidades do sistema fornecido pelo cliente, conhecido como histórias dos usuários.

Os testes de aceitação são utilizados para reproduzir as exigências feitas pelos *stakeholders*, e, assim, certificar que o sistema expressa as exigências feitas e que só assim pode ser considerado aceitável (OKOLNYCHYI; FÖGEN, 2016). Portanto, este tipo de teste só é bem-sucedido se evidenciar que um requisito foi adequadamente implementado e atendeu ao que foi solicitado.

Com base nas expectativas desejadas e descritas pelos *stakeholders* sobre o produto que será construído, é que o teste será criado, e o produto só estará pronto quando todos os critérios levantadas pelo cliente estiverem satisfeitos (NAIK; TRIPATHY, 2008)**.**

Deste modo, o teste de aceitação é a verificação se todos os critérios determinados pelos envolvidos no projeto foram satisfeitos. Os critérios de aceitação devem ser claramente definidos e de conhecimentos de todos para evitar qualquer argumentação de que o *software* não está funcionando como solicitado.

A realização de teste de aceitação gera alguns benefícios como (FELLER, 2015):

- Pode ser usado para documentar os requisitos de *software*;
- Pode ser implementado como especificações executáveis;
- Proporciona um meio de comunicação sem distorções entre a equipe e os *stakeholders*;
- Os *stakeholders* podem avaliar o *software* durante o desenvolvimento;
- Facilita a refatoração;
- Proporciona desenvolvimento com boas características de testabilidade e baixo acoplamento dos módulos.

Assim, o quanto antes se tiver certeza que o sistema está funcionando como esperado pelo *stakeholders* e que os defeitos aqui encontrados estão sendo evitados de serem encontrados nas fases posteriores e, desse modo, gerar um trabalho maior para corrigi-los e adequá-los as expectativas dos interessados (BILAL *et al.*, 2016).

### 2.2.2 Especificação dos testes de aceitação

Os requisitos de um sistema descrevem as características e comportamentos que o sistema espera-se ter após a sua construção. Logo, durante este processo e, também, nas etapas posteriores, é necessário avaliar o que vai ser desenvolvido ou

o que está em desenvolvimento, para ter certeza que se constrói a coisa certa e da forma correta, sem erros e conforme o cliente solicitou.

A linguagem utilizada na criação de testes de aceitação deve ser simples e próxima da linguagem natural, com intuito de evitar problemas de interpretação e de ambiguidades, além de possibilitar conexão com o código do sistema que será executado como teste (BERNARDO, 2011).

A descrição dos testes de aceitação deve ser capaz de deixar claro qual é o seu objetivo, assim como, ser capaz de representar o estado atual do sistema. Além disso, devem ser fácil de manter e confiável, de modo que camadas de abstrações possam ser usadas para facilitar tais características (SMART, 2015). Assim, um bom teste de aceitação deve observar algumas características:

- **Comunicação clara**: a escrita e os termos utilizados (semântica) para descrever os testes de aceitação são importantes, pois precisam ser claros o suficiente para que todos os stakeholders entendam quais comportamentos do negócio estão sendo implementados no teste, além do que, essa descrição será um artefato que poderá ser utilizado futuramente para esclarecimento de como tais funcionalidades foram implementadas.
- ***Feedback* significativo**: em caso de falha, deve ser de fácil entendimento o motivo pelo qual ocorreu tal falha, qual o real objetivo do teste e como é o seu funcionamento.
- **Confiável**: o teste deve ser capaz de passar confiança, de modo que só poderá ser satisfeito se atender todos os critérios de aceitação estabelecidos pelo cliente, e recusados caso isso não ocorra.
- **Fácil de manter**: o teste deve ser construído de modo que atualizações ou modificações que se fizerem necessárias não sejam capazes de desestabilizá-lo.

A escrita dos testes de aceitação pode ser feita utilizando a linguagem textual *Gherkin* (Figura 11). É uma Linguagem de domínio específico que auxilia na criação de documentação e automação de teste, através da representação dos comportamentos do sistema, com linguagem simples e estruturada, sem a necessidade de especificar como tais comportamentos serão implementados (CEZERINO; NASCIMENTO, 2016).

```
Narrative:
In order to Comunicar efetivamente ao negócio algumas funcionalidades
As a equipe de desenvolvimento
I want to usar Behaviour-Driven Development

Scenario: Um cenário é uma coleção de passos executáveis
Given passos que representam uma condição prévia para um evento
When passos representam a ocorrência do evento
Then passos representam o resultado do evento

Scenario: Outro cenário explorando diferentes combinações de eventos
Given uma pré-condição
When ocorre um evento negativo
Then um resultado deve ser capturado
```

**Figura 11 – Notação *Gherkin* para escrita de uma história.**

Fonte: JBehave (2015)

A estrutura *Gherkin* para descrição das histórias é bem parecida com a de um Caso de Uso. Toda história possui um título, uma descrição sobre de que se trata a história e cenários que são descritos em uma sequência de passos predefinidos com o objetivo de reproduzir os comportamentos do sistema e as possíveis interações com este (CEZERINO; NASCIMENTO, 2016).

Para cada comportamento que se pensar que a funcionalidade possa ter, é criado um cenário para representá-lo. Cada cenário terá uma breve descrição do que faz, ou seja, do que estar testando. Um cenário está divido basicamente em três partes (Dado-Quando-Então)(SMART, 2015):

- Dado uma precondição para o comportamento testado ocorrer.
- Quando este comportamento ocorrer.
- Então espera-se um resultado (ação) da ocorrência do comportamento.

Seguindo esta estrutura de escrita dos cenários, deve-se definir os passos que representam a pré-condição para o evento testado ocorrer, os passos quando da ocorrência do evento e, por fim, definir os passos da execução depois que o evento finalizar sua ocorrência.

Assim, os requisitos do sistema são representados e entendidos através de escrita das histórias e de seus comportamentos que serão testados, e que refletem os pontos de vista das várias pessoas envolvidas neste processo.

Essa forma de escrever testes de aceitação permite uma compreensão compartilhada do funcionamento do sistema entre os usuários do negócio e a equipe de desenvolvimento, além disso, facilita a transformação dessas especificações em algo executável (JBEHAVE, 2015).

É importante ressaltar a qualidade da especificação do teste de aceitação, pois, neste momento, conflitos podem ser encontrados e avaliados, evitando que passem para fases posteriores e se tornem mais custosos para corrigir, visto que erros neste momento significa insatisfação das necessidades do cliente, e se não consideradas poderão gerar modificações na arquitetura ou implementação (BILAL et al., 2016).

## 2.3 Desenvolvimento dirigido por comportamento

Diante de toda a particularidade na criação de um *software*, buscando atender e se adaptar as novas situações no desenvolvimento, metodologias de desenvolvimento, ferramentas e técnicas são criadas. Como exemplo disso, têm-se as metodologias ágeis que é uma forma de desenvolver buscando a flexibilidade ao novo, valorizando mais os indivíduos e *software* funcionando, e a colaboração com o cliente (BECK *et al.*, 2001).

Entre estas metodologias, o BDD , criada por Dan North, é uma metodologia de desenvolvimento de *software* recente e que cada vez mais está ganhando destaque entre a comunidade. Surgiu da necessidade de complementar a metodologia *Test Driven Development* (TDD), criada por Kent Beck, em face a questionamentos feitos por Dan North. North (2006) percebeu que sempre enfrentava as mesmas dificuldades durante a implementação de TDD nas equipes, como , por exemplo, por onde começar testar, o que seria testado, o quanto seria testado, e até questões simples como nomear os testes adequadamente e o quanto o motivo da falha de um teste era de fácil entendimento.

Além disso, North percebeu que cada parte de código escrito deve possuir uma relação de valor para o negócio. Ou seja, o *software* deve ser construído, codificado, de modo que cada parte da solução desenvolvida reflita valor para o negócio. Assim, passou a pensar como poderia implementar este conceito em BDD afim de que o desenvolvimento de *software* atenda aos objetivos do negócio.

O BDD tem como foco o comportamento do sistema. Para isso utiliza-se da criação de exemplos concretos do comportamento do sistema para ajudar a compreender como os recursos irão fornecer valor ao negócio (SMART, 2015). Diferentemente de TDD em que o foco está nas funcionalidades, em testar de forma efetiva se determinada função, ação, do sistema está funcionando corretamente (NORTH, 2006).

Com a prática de TDD consegue-se garantir que o código está funcionando corretamente, no entanto se os testes realizados não descreverem de forma adequada e completa os possíveis comportamentos do sistema, isso poderá gerar uma falsa sensação de segurança a aplicação (NORTH, 2006).

O BDD busca tornar as práticas de desenvolvimento com testes mais acessí-

veis e intuitivas, mudando o foco e o vocabulário do termo teste para o conceito de comportamento (JBEHAVE,2015). Assim, as histórias são escritas e entendidas com base na idealização de seus comportamentos. Basicamente o BDD funciona como ilustrado na Figura 12. Cada história representa uma funcionalidade do sistema e suas características são levantadas com base nos possíveis comportamentos que possam ocorrer. Os comportamentos são representados por cenários nas histórias. A metodologia defende que essas descrições sejam executáveis, ou seja, que sejam mapeadas para código e se tronem executáveis.

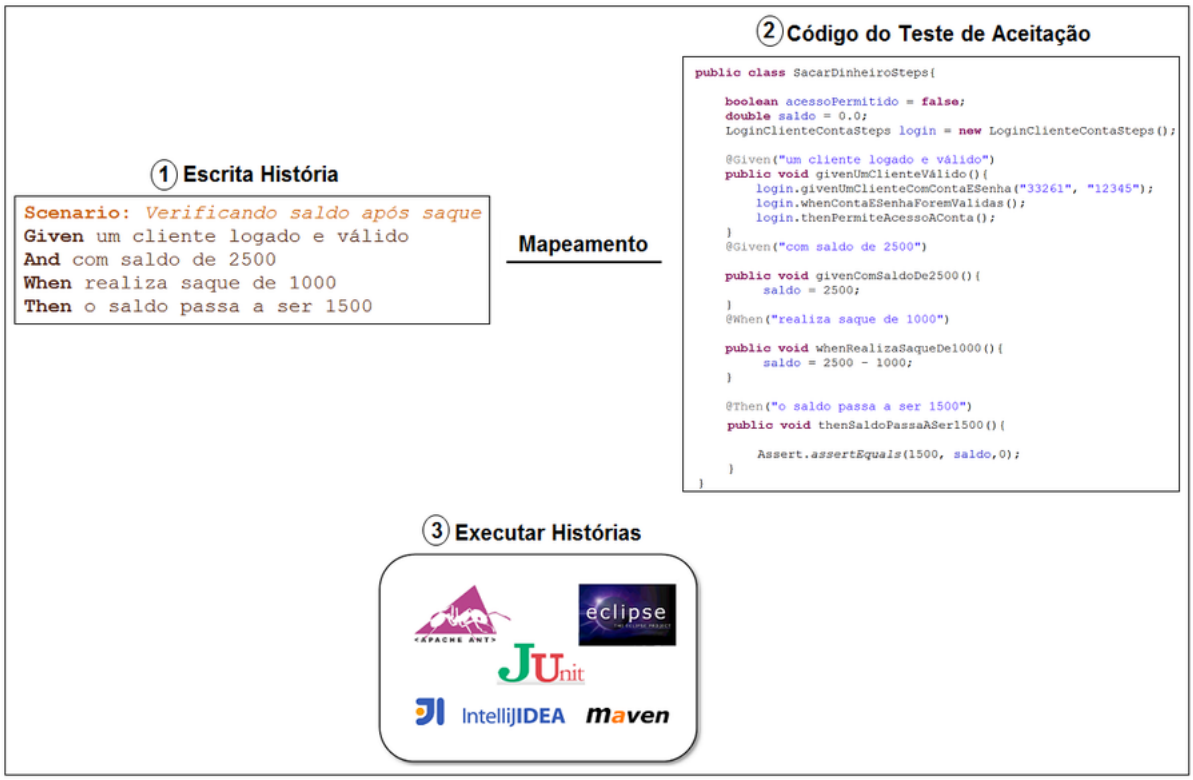


**Figura 12 – Visão geral do funcionamento BDD.**

Fonte: JBehave (2015)

O BDD possui algumas características e vantagens para o desenvolvimento de *software* que colocados em prática podem ser úteis para construir e entregar *software* que agregue ainda mais valor para negócio.

### 2.3.1 Principais características

BDD é um conjunto de práticas de engenharia de *software* projetado para ajudar equipes a construir e entregar *software* com mais valor, qualidade e rapidez. Fundamenta-se em práticas ágeis consolidadas, incluindo, em particular, *Test-Driven Development* (TDD), *Domain-Driven Design* (DDD) e *Acceptance Test Driven Development* (ATDD) (SMART, 2015).

O TDD estabelece que o desenvolvimento de cada funcionalidade inicia com a criação de testes que irão naturalmente falhar visto que foi criado antes da própria funcionalidade, e em seguida é feita a codificação para que se passe no teste e uma posterior refatoração para garantir a manutenibilidade do código (BECK, 2002).

Já o DDD é uma abordagem voltada para trabalhar a evolução do *software* com base nos conceitos do negócio, fornecendo estruturas básicas e terminologias objetivando tomadas de decisões em projetos de *software* que lidam com domínios complexos, colocando como foco central do projeto o domínio do negócio, visando a proporcionar colaboração criativa entre as partes técnicas e as partes do domínio do negócio (COMMUNITY, 2007).

Por sua vez, ATDD é uma metodologia para desenvolvimento dirigido pela criação de teste de aceitação que representam os requisitos das partes interessadas (FELLER,2015). Deste modo, os requisitos são transformados em testes que averiguam as funcionalidades, ou seja, primeiro os requisitos são especificados em forma de testes de aceitação e só depois as funcionalidades são criadas com base nestes testes (PUGH,2010 apud FELLER 2015).

Assim, o BDD faz uso destes conceitos para criar comportamentos concretos das histórias dos usuários, de modo que o comportamento de uma história é o seu teste de aceitação (NORTH,2006). Portanto, se todos estes testes de aceitação forem atendidos, é dito que o sistema está se comportando corretamente.

O mais o importante para BDD é fornecer uma linguagem comum e estruturada em expressões simples com objetivo de facilitar a comunicação entre os integrantes da equipe e os *stakeholders* (SMART, 2015). E, deste modo, minimizar as dificuldades encontradas entre a especificação, design, implementação e confirmação do comportamento de um sistema, permitindo a entrega incremental de *software* que atenda aos objetivos do negócio (NORTH, 2014)

Algumas características principais de BDD são: linguagem ubíqua, processo iterativo de decomposição, descrição em texto livre com templates de histórias e cenários de usuário, teste de aceitação automatizado com regras de mapeamento, código compreensível de especificação orientado a comportamento e orientação a comportamento em diferentes fases (SOLÍS; WANG, 2011).

#### 2.3.1.1 Comunicação

Permitir comunicação 'limpa' e eficaz entre grupos de pessoas de contextos diferentes sempre uma tarefa difícil, visto que cada grupo possui sua forma de se expressar tecnicamente. O BDD tenta eliminar tal custo de tradução que ocorre na comunicação entre equipe de programadores e os especialistas do negócio (KUDRYASHOV; STEAD;

NORTH, 2015).

Para isso faz uso do conceito de linguagem ubíqua introduzida por Eric Evans no DDD, no qual promove o uso desta linguagem para facilitar o entendimento das pessoas de negócios durante a construção e modelagem do sistema (SMART, 2015). Evans também alerta que muitos projetos de *software* sofrem de má comunicação entre especialistas do negócio e a equipe de desenvolvimento.

O BDD, portanto, faz uso deste recurso para que analistas de negócios possam descrever requisitos evitando interpretações ambíguas e equivocadas. Como forma de implementar este conceito o BDD descreve critérios de aceitação (cenários) para as histórias de usuários em uma linguagem simples e estruturada (Figura 13).

```
Scenario --> Um cenário é uma coleção de passos executáveis
Given --> Passos que representam uma condição prévia para um evento
When --> Passos representam a ocorrência do evento
Then --> Passos representam o resultado do evento
```

**Figura 13 – Escrita de um cenário com notação Gherkin.**
Fonte: JBehave (2015)

#### 2.3.1.2 Processo iterativo com participação dos *stakeholders*

A dificuldade de entendimento, incerteza, dos requisitos do *software* é algo comum na maioria dos projetos, assim como a definição das reais funcionalidades que precisam ser entregues. Em BDD a equipe se envolve em conversas com usuários finais e também com os demais interessados no projeto, de modo que estes possam contribuir e influenciar com informações sobre o comportamento do sistema e, assim, construir junto com os especialistas de negócio o entendimento de quais características o *software* necessita ter (SMART, 2015).

#### 2.3.1.3 Templates para escrita de histórias e cenários de usuário

Apenas citar exemplos sobre comportamento do sistema não é suficiente para representar claramente as características do *software*. Pois cada parte interessada pode representar tais características usando a forma técnica e conveniente para cada um (KUDRYASHOV; STEAD; NORTH, 2015). Para isso, é proposto uma forma única,(template) para que as histórias e cenários sejam escritos .

Este vocabulário comum pode ser expresso seguindo o padrão: *Given-When-Then*. Este padrão é conhecido como *Gherkin*, no qual um arquivo de texto contém uma breve descrição dos requisitos relacionados a uma funcionalidade, recurso, e vários cenários exemplificando o funcionamento (comportamento) de tal recurso (Figura 14) (SMART, 2015).

```
Narrative:
In order to administrar meu dinheiro com eficiência
As a cliente do banco
I want to transferir dinheiro entre as minhas contas sempre que eu precisar

Scenario: Transferência com saldo insuficiente
Given minha conta corrente com saldo de 1000,00
When eu transferir 1500,00 da conta corrente para conta poupança
Then eu deveria receber uma mensagem: saldo insuficiente
```

**Figura 14 – Exemplo de uma história de usuário com notação Gherkin.**

Fonte: Smart (2015)

#### 2.3.1.4 Especificações executáveis

A base para construção do sistema são as histórias de usuários e os cenários correspondentes a cada história levantada pela equipe e pelo cliente. Cada história possui comportamentos que são os critérios de aceitação, de modo que se todos os critérios de aceitação forem atendidos, assume-se que o sistema está se comportando corretamente (NORTH, 2006).

Mas fazer toda esta tarefa manualmente seria demorado e ineficiente. O tempo gasto com a realização de testes, assim como os custos da sua criação podem ser diminuídos com a sua automação, o que permitiria melhorar a confiabilidade, diminuir o custo de repetição e taxa de defeitos (FELLER, 2015 *apud* RUMPE, 2003).

Entretanto, deve-se ter atenção (principalmente em grande baterias de testes) na criação de baterias de testes para que sejam completas e que os testes realmente representem os variados comportamentos do *software* (BERNARDO, 2011), e, desse modo, possam garantir qualidade ao sistema.

A geração automática de testes de aceitação proporciona uma especificação executável do produto. Os testes são realizados no nível mais alto do sistema e com todos os componentes da aplicação. Portanto, verificando, se mediante uma seqüência de passos realizada pelo usuário o sistema está se comportando corretamente (FELLER, 2015 *apud* MAJCHRZAK; SIMON, 2012). Além disso, a automação dos testes de aceitação pode servir para auxiliar nos testes de regressão, pois acaba por assegurar que durante a evolução do *software* os requisitos do cliente não são violados (FELLER, 2015 *apud* NEGARA; STROULIA, 2012).

Outra vantagem da automatização dos testes de aceitação é que todo foco ou dedicação que poderia ser dedicada a realização desta etapa (teste de aceitação) pode passar a ser investida no entendimento do módulo em desenvolvimento, permintindo concentração e encorajamento de todos nos objetivos do projeto, melhorando a qualidade do produto e a produtividade da equipe (OKOLNYCHYI; FÖGEN, 2016).

Para realizar a automação desses testes de aceitação existem várias ferramentas que aplicam os conceitos de BDD, e em várias linguagens de programação: JBehave para Java, RSpec para Ruby, PHPSpec para PHP, Cucumber, etc. As histórias (requisitos) são escritas em linguagem simples, no qual o framework faz a conversão gerando os testes e exibe o resultado de forma textual, “amigável” (SMART, 2015).

#### 2.3.1.5 Código compreensível por todos

O código deve fazer parte da documentação do sistema e as especificações devem fazer parte do código (SOLÍS; WANG, 2011). Logo os relatórios produzidos pelas especificações executáveis deixam de ser meras documentações técnicas e passam a ser disponibilizados com vocabulário familiar para os usuários.

Além dos critérios de aceitação automatizados proporcionarem uma excelente documentação para todos da equipe, também será de grande utilidade para outros desenvolvedores, pois terá uma documentação sempre atualizada, fácil e barata de manter, e que conterá representações do código que manifesta com mais clareza a intenção por trás de cada especificação (SMART, 2015).

#### 2.3.1.6 Foco no comportamento

Os testes de aceitação automatizados são etapas importantes da implementação de BDD, mas o comportamento dos sistemas não é tratado apenas nesta etapa e sim em todas as outras fases: planejamento, análise e implementação (SOLÍS; WANG, 2011). Deste modo procura-se inicialmente entender as metas do negócio para através delas identificar comportamentos do sistema que possam ajudar a atingir tais metas (KUDRYASHOV; STEAD; NORTH, 2015). Assim, as especificações do sistema (histórias do usuário) são descritas com base nos objetivos do negócio, sendo elencado um conjunto de exemplos, cenários, para as características que foram levantadas.

### 2.3.2 Benefícios

A abordagem de BDD pode ser vista como implementação de duas etapas: representação de comportamentos do sistema através de exemplos e a automatização destes exemplos em especificações executáveis (KUDRYASHOV; STEAD; NORTH, 2015). Ou seja, a primeira parte é a descrição da forma como os usuários podem interagir com *software* e como ele deve reagir (criação de cenários), já a segunda parte é a transformação destas descrições em código para verificar e validar tal comportamento.

BDD assume que para o *software* atender os requisitos do negócio ele deve identificar quais são os objetivos que o negócio pretende obter com as funcionalidades

que serão criadas e que deve conseguir, junto aos *stakeholders*, identificar metas que sejam específicas, mensuráveis, alcançáveis, relevantes e com prazos (SMART, 2015).

BDD também assume que o *software* só pode ajudar o negócio a atender seus objetivos quando apoiado no comportamento das pessoas. Para isso, o BDD se utiliza de uma técnica chamada Mapeamento de Impacto para elencar as alternativas para se chegar ao objetivo do negócio como base no comportamento das pessoas (KUDRYASHOV; STEAD; NORTH, 2015). Assim, para cada objetivo é identificado, quem (pessoas) e o que (impactos), poderia contribuir ou dificultar para atingir o objetivo, assim como o que poderia ser feito para apoiar ou evitar os impactos gerados pelos usuários.

Desta forma, BDD trabalha a partir do comportamento do negócio, ou seja, a partir dos impactos gerados pelos usuários , em como os usuários interagem com o *software* e como este reage a tais interações, permitindo que o esforço dedicado durante o desenvolvimento seja para identificar características que sejam essenciais para o negócio. Como consequência de um desenvolvimento com foco no negócio, o BDD consegue entregar *software* com valor para os *stakeholders*, visto que toda a sua concepção e funcionalidades foram criadas com base nas características primordiais para o negócio.

O fato do desenvolvimento do *software* contar com a participação de todos os interessados e em todo o processo, desde dos testadores, desenvolvedores e as pessoas que possam ser direta ou indiretamente afetadas pelas funcionalidades que serão criadas, isso proporciona um entedimento compartilhado das caracteríaticas do *software* e uma boa comunicação (SOLÍS; WANG, 2011). A equipe procura dialogar com os usuários finais e demais interessados para chegar ao senso comum das características que o *software* precisa possuir. A equipe não coleta simplesmente uma lista de funcionalidades que precisam ser implementa, mas sim procura identificar através de conversas com os *stakeholders*, características que possam auxiliar os objetivos do negócio.

O resultado das conversas da equipe é a transformação dessas histórias de usuários em especificações executáveis de forma automática e uma boa documentação para todos. Assim, tais informações geradas podem ser usadas pelos desenvolvedores para verificar como os recursos existentes funcionam, os testadores e analistas de negócios podem observar como foram implementados os requisitos solicitados, os proprietários do produto e gerentes de projeto podem usar tais informações para avaliar o estado atual do projeto, verificar o seu progresso e ajudar na tomada de decisões (SMART, 2015). Além disso, os usuários podem se valer desta forma de exposição de informações para ver o que o sistema é capaz de fazer e como ele funciona.

Todo a comunicação feita entre a equipe e os stakeholders é através de uma

forma simples e estruturada com a utilização de *templates* para escrita das características e comportamentos levantados. Assim é estabelecido uma única forma de diálogo entre a equipe e o cliente, evitando que os mesmos utilizem termos técnicos de suas respectivas áreas, descrições com sentidos vagos ou que possam ter vários significados, logo proporcionando uma comunicação com menos ruídos (KUDRYASHOV; STEAD; NORTH, 2015).

Como todo o código gerado é derivado das histórias de usuário que foram escritas com uma linguagem simples, estruturada e comum a todos (linguagem ubíqua), o código ser torna de fácil compressão tanto para a equipe (desenvolvedores e testadores) quanto para os *stakeholders*, pois possui termos que são familiares aos especialistas de negócios (SMART, 2015). Assim, os stakeholders conseguem entender facilmente o funcionamento do *software* e os desenvolvedores possuem uma boa fonte de informações técnicas (documentação) com especificações claras.

## 2.4 Síntese do capítulo

Neste capítulo, por meio das pesquisas feitas em buscadores automáticos e buscas manuais, foram apresentados conceitos necessários para os entendimentos deste trabalho com relação a Orientação a Aspecto e sua aplicação na Engenharia de Requisitos, assim como, apresentação da metodologia ágil BDD e suas características relacionadas a especificação de teste no formato textual usando a notação *Gherkin*, para escrita de histórias de usuário e cenários, que representam as funcionalidades e comportamentos, respectivamente, levantados pelas partes interessadas.

A pesquisa também pode evidenciar a importância e o crescente uso dos conceitos de Orientação a Aspecto nas fases inicias do desenvolvimento (requisitos, projeto e arquitetura), surgindo uma linha de pesquisa específica (*Early Aspects*) para estudar esta aplicabilidade de orientação a aspecto.

Também verificou-se que a metodologia BDD traz uma forma diferenciada de olhar para criação dos testes das aplicações, procurando utilizar especificações baseadas em exemplos para levantar a funcionalidades e comportamentos esperados por cada função, por meio de uma linguagem simplificada e de fácil compreensão por todos.

Deste modo, as informações coletadas de cada área evidenciam a relevância da proposta deste trabalho em investigar se AORE pode auxiliar na especificação dos testes escritos com a notação *Gherkin*, visto que o processo de identificação e modularização de interesses transversais presentes nas histórias escritas com a notação já poderia ser realizado durante a escrita dos testes, antes mesmo da codificação, ou seja, antes que as especificações textuais se tornem testes executáveis.

# 3 ABORDAGEM PROPOSTA

Nesta seção será apresentada a proposta desse trabalho, uma abordagem orientada a aspectos para a identificação e modularização de interesses transversais durante a escrita de testes de aceitação criados como histórias de usuário, usando a notação *Gherkin*, segundo os conceitos e práticas da metodologia ágil *behaviour driver developement* .

A abordagem, além dos conceitos e práticas do BDD, foi motivada por um modelo de engenharia de requisitos orientada a aspecto (MOREIRA *et al.*, 2013), pela definição de um processo de *design* orientado a aspecto (SOMMERVILLE, 2011) e por um padrão que descreve uma solução para duplicação de requisitos (RAMOS *et al.*, 2007).

Foi realizado um *running example* (MOREIRA *et al.*, 2013) para auxiliar na explicação das etapas da abordagem proposta. O running example "*security requirements updating*" refere-se as mudanças que podem ocorrer dos requisitos de segurança durante a construção ou evolução de um sistema, como, por exemplo, adição, remoção ou alteração destes aspectos de segurança. A ocorrência de mudanças nos requisitos de segurança pode levar a falhas em outras partes do sistema e, assim, interferir na satisfação de outros requisitos do sistema e dos próprios requisitos de segurança já estabelecidos.

O *running example* utilizado refere-se a três entidades de um sistema: Empregado, Usuário e Função. Um empregado possui atributos como nome, idade e salário. O salário do empregado é calculado de acordo com sua idade e a idade para aposentadoria. A entidade Usuário possui os atributos nome de usuário e autorizado (*boolean*). Os usuários possuem permissões para realizar determinadas tarefas, por isso na realização de alguma operação deve ser verificado suas permissões. Além disso, um empregado possui uma função. No *running example* é estabelecido como requisito de segurança manter protegido os dados dos salários dos empregados de acessos não autorizados de usuários para alterações ou visualizações.

## 3.1 Visão geral da abordagem

A abordagem estabelece um conjunto de etapas que deve ser seguido para aplicar os conceitos de orientação a aspecto na criação das histórias, conforme ilustrado na figura 7. Todas as etapas da abordagem deste trabalho foram motivadas pelas características e práticas do desenvolvimento dirigido por comportamento (BDD) e da orientação a aspecto.

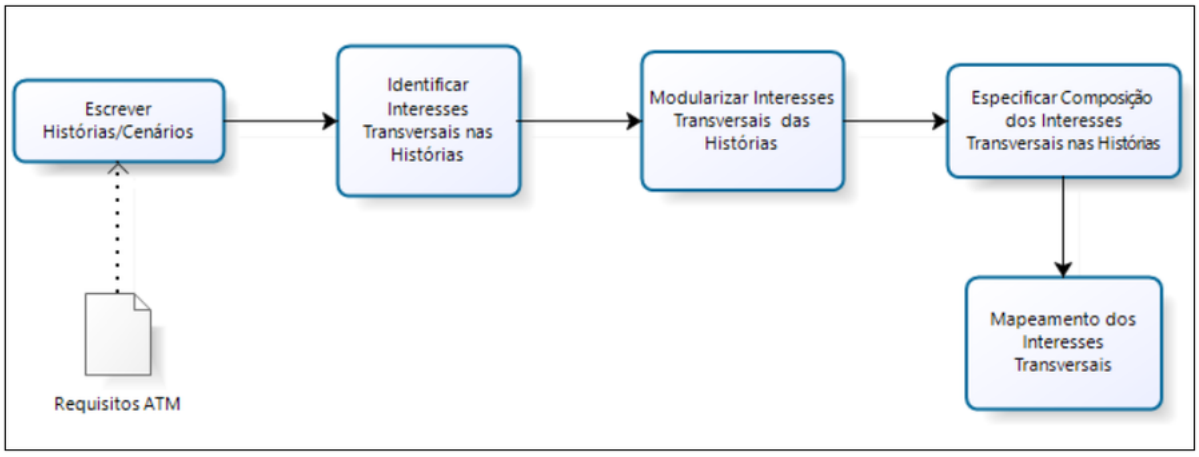


**Figura 15 – Fluxograma da abordagem**

Fonte: Autor

As referências utilizadas pelo trabalho para montar as etapas da abordagem, foram usadas por se colocarem como abordagens genéricas de aplicação dos conceitos de orientação a aspecto e da escrita de requisitos. O fato de definirem processos claros e independentes possibilitou que a abordagem estabelecesse claramente como proceder para identificar e modularizar interesses transversais na escrita dos teste de aceitação.

As próximas seções irão descrever as etapas que compõem a abordagem proposta neste trabalho: Seção 3.1.1 – como escrever as histórias que representam os interesses do sistema; Seção 3.1.2 – como identificar os interesses transversais nas histórias escritas; Seção 3.1.3 – como modularizar os interesses identificados na etapa anterior; Seção 3.1.4 – definição de como os interesses irão compor o sistema; Seção 3.1.5 – apresenta a importância do mapeamento dos interesses transversais para as fases posteriores do desenvolvimento de um sistema.

### 3.1.1 Etapa 1 - Escrita das histórias

Conforme estabelece a metodologia BDD, as especificações das histórias representam as funcionalidades do sistema e são compostas por um conjunto de cenários, que representam diversos exemplos de comportamentos que uma determinada funcionalidade do sistema possa ter (JBEHAVE, 2015). Para a escrita das histórias será utilizado a notação *Gherkin*, na qual a especificação da história é iniciada com uma narrativa (*In order to - As a - I want to*) e a criação dos cenários com o padrão *Given-When-Then*. A utilização desta notação para a escrita das histórias e cenários do *running example* está demonstrada nas Figuras 16 e 17.

```
Narrative:
In order to manipular as funcionalidades do empregado
As a usuario
I want to conseguir realizar operações

Scenario: C01-Deve cadastrar o empregado
Given uma solicitação para cadastrar um empregado
When informado nome,idade e salário
Then o empregado deve ser cadastrado

Scenario: C02-Deve calcular o salário do empregado na ativa
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for menor que a idade da aposentadoria
Then o salario dev ser igual 1.000,00
acrescido de 50,00 por cada ano que faltar para aposentadoria

Scenario: C03-Deve calcular o salário do empregado aposentado
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for maior ou igual a idade da aposentadoria
Then o salario deve ser igual a 1.000,00

Scenario: C04-Espera obter o valor do salário de um empregado
Given solicitação para ver o salário de um empregado
And o usuário seja autorizado a realizar a operação
When informado o empregado que deseja obter o salário
Then o acesso aos dados do salário do empregado é permitido

Scenario:  C05-Espera alterar o valor do salário de um empregado
Given solicitação para alterar o salário de um empregado
And o usuário seja autorizado a realizar a operação
When informado o empregado que deseja alterar o salário
Then a operação é permitida
```

**Figura 16 – História "Gerenciar Empregado" do *running example* escrita com a notação Gherkin.**

Fonte: Autor

```
Narrative:
In order to manipular as funcionalidades do usuario
As a usuario
I want to conseguir realizar operações

Scenario:  C06-Deve cadastrar usuario
Given solicitação para cadastrar um usuário
When informado nome de usuário e o seu tipo de autorização
Then o usuário deve ser cadastrado

Scenario: C07-Deve verificar permissão do usuário
Given o usuário realize uma operação
When a operação for compatível com a autorização lhe dada
Then o usuário está autorizado a realizar a operação
```

**Figura 17 – História "Gerenciar Usuário" do *running example* escrita com a notação Gherkin.**

Fonte: Autor

### 3.1.2 Etapa 2 - Identificação dos interesses transversais nas histórias

A separação e identificação dos interesses do sistema é uma tentativa de organizar cada elemento (parte) que compõem o sistema com sua respectiva função, de modo que esta função seja única. Logo, é possível entender melhor um interesse do sistema de forma isolada, sem necessidade do entendimento das outras partes e, assim, mudanças que possam ocorrer serão localizadas em poucas partes do sistema, facilitando a manutenção e a evolução (SOMMERVILLE, 2011).

Após a escrita das histórias, o próximo passo é analisar as características expressas nas histórias no intuito de identificar o que poderia representar um interesse que seja transversal a outras histórias.

Se uma interesse atravessar vários requisitos, ela pode ser considerada um aspecto (TEKINERDOgAN et al., 2005). Um aspecto pode representar uma função do sistema (log de operações), um critério de decisão (escolha da arquitetura), um critério de qualidade (desempenho), ou critérios políticos (regras de negócios) (RASHID; MOREIRA; ARAÚJO, 2003).

É importante fazer uma correlação entre os interesses transversais (crosscutting concerns) e os requisitos para deixar claro a relação existente entre os dois, visto que o comportamento das funcionalidades podem ser afetadas por interesses que estejam espalhados por outras partes (RASHID *et al.*, 2002). Uma forma de representar este impacto, que os interesses transversais possam causar sobre as partes que perpassam, é por meio de uma *Requirements Composition Table* (RCT) (CHERNAK, 2013).

A utilização de uma RCT proporciona a todos os *stakeholders* do projeto de *software* uma forma de comunicação clara e simples, gerando uma linguagem comum entre todos os envolvidos, através de uma visão macro e estruturada das funcionalidades do *software* em forma de tabela. As funcionalidades principais são representadas nas colunas e os interesses transversais são representados nas linhas (figura 18). Desta forma, os interesses transversais são facilmente identificados e visíveis.

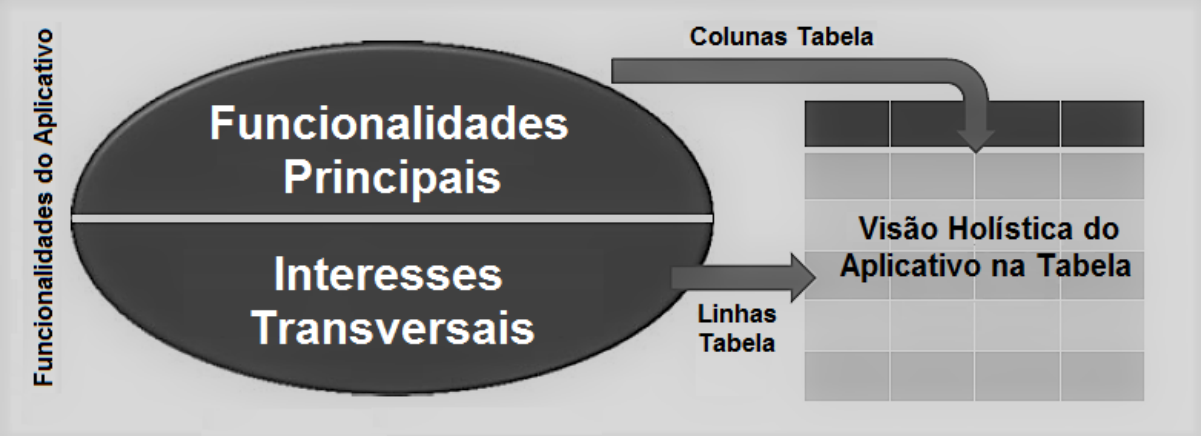


**Figura 18 – Dimensões da RCT**

Fonte: Chernak (2013)

Com base nas características citadas anteriormente da tabela de composição dos requisitos, é criado uma Tabela de Composição das Histórias (TCH). A TCH irá relacionar quais cenários compõem cada história, de modo que os cenários serão listados nas colunas e as histórias serão listadas nas linhas. A Tabela 1 representa as histórias e seus respectivos cenários para o *running example*.

**Tabela 1 – Tabela de composição das histórias do *running example.***

| **Histórias** | **Cenários** | | | | | |
|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 |

| **Histórias** | **Cenários** | | | | | |
|---|---|---|---|---|---|---|
| Gerenciar Empregado | **X** | **X** | **X** | **X** | | |
| Gerenciar Usuário | | | | | **X** | **X** |

Fonte: Autor

A tabela exibe uma matriz de composição das histórias, onde fica fácil e visível identificar quais cenários compõem uma história e, consequentemente, quais cenários pertencem a várias histórias.

Se uma coluna da tabela TCH possuir várias linhas marcadas, ou seja, se um cenário estiver presente em mais de uma história, é considerado que existe um espalhamento da característica que aquele cenário representa.

Portanto, é evidente que se um cenário pertence a várias histórias, ele está espalhado, e levando em consideração que uma história representa uma funcionalidade e, portanto, encapsula comportamentos referentes a esta funcionalidade, então, caso os cenários estejam espalhados em várias histórias, eles também estarão entrelaçados em outras funcionalidades.

No entanto, os interesses transversais podem estar presentes na escrita dos próprios cenários das histórias e não evidenciados em um cenário propriamente. Por exemplo, na TCH do *running example* não foi evidenciado nenhum cenário que esteja presente nas duas histórias. No entanto, na descrição dos cenários **C04** e **C05** da história “**Gerenciar Empregado**“ pode ser notado a presença do interesse “**verificar permissão do usuário**” (Figura 19). Ou seja, este interesse corresponde ao cenário **C07** da história “**Gerenciar Usuário**“ que verificar a permissão do usuário e, portanto, é um interesse transversal.

```
Narrative:
In order to manipular as funcionalidades do empregado
As a usuario
I want to conseguir realizar operações

Scenario: C01-Deve cadastrar o empregado
Given uma solicitação para cadastrar um empregado
When informado nome,idade e salário
Then o empregado deve ser cadastrado

Scenario: C02-Deve calcular o salário do empregado na ativa
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for menor que a idade da aposentadoria
Then o salario dev ser igual 1.000,00
acrescido de 50,00 por cada ano que faltar para aposentadoria

Scenario: C03-Deve calcular o salário do empregado aposentado
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for maior ou igual a idade da aposentadoria
Then o salario deve ser igual a 1.000,00

Scenario: C04-Espera obter o valor do salário de um empregado
Given solicitação para ver o salário de um empregado
And o usuário seja autorizado a realizar a operação  Interesse Permissão
When informado o empregado que deseja obter o salário
Then o acesso aos dados do salário do empregado é permitido

Scenario:  C05-Espera alterar o valor do salário de um empregado
Given solicitação para alterar o salário de um empregado
And o usuário seja autorizado a realizar a operação  Interesse Permissão
When informado o empregado que deseja alterar o salário
Then a operação é permitida
```

**Figura 19 – Presença do interesse "permissão" na história "Gerenciar Empregado" do *running example*.**

Fonte: Autor

Por isso é necessário analisar todos os cenários das histórias para averiguar se algum cenário, em sua escrita, possui características que não é do seu interesse, ou seja, que não é da sua responsabilidade e, portanto, não deveria ser implementado por ele.

Após essa análise, a TCH deve ser atualizada, se necessário, para identificar possíveis cenários que no primeiro momento não estavam "marcados" como pertencentes a uma determinada história. Além disso, nesta análise, é possível identificar interesses que não são de responsabilidade das histórias, e que não pertencem a nenhuma outra história, ou seja, pode-se encontrar novos aspectos que ainda não estão representados como cenários de nenhuma história atual.

Uma Tabela de Interesses Transversais (TIT) pode ser criada para demonstrar em quais histórias e cenários os interesses transversais encontrados estão presentes (tabela 2).

**Tabela 2 – Tabela de Interesses Transversais do *running example*.**

| **Interesses Transversais** | **Histórias** | |
|---|---|---|
| | Gerenciar Empregado | Gerenciar Usuário |
| Permissão/Segurança | **2 Cenários** | **0 Cenários** |

Fonte: Autor

Assim, a TIT evidencia quais interesses transversais (aspectos) estão presentes no sistema e como estão distribuídos nas funcionalidades (histórias) e nos seus comportamentos (cenários).É importante ressaltar que espalhamento e entrelaçamento pode ocorrer dentro da própria história, pois se um interesse estiver presente em mais de um cenário da mesma história, ele estaria espalhado e também entrelaçado.

**Heurística para identificação dos interesses transversais**

Os interesses podem estar associados a diversas características do sistema, desde de estar relacionada a uma funcionalidade (quando um interesse é alguma parte de uma funcionalidade do sistema) , a estar relacionada a qualquer parte de interesse ou foco do *software* estabelecidos como importante pelos vários stakeholders, como, por exemplo, um tempo rápido de resposta do sistema, a facilidade de manter e evoluir o *software* (SOMMERVILLE, 2011). Assim, existem interesses diversos que podem estar espalhados por várias outras partes do *software* e, portanto, realmente definir, neste momento, o que seja um interesse que influencie outras partes do *software*, seja uma tarefa dificultosa decorrente dos interesses estarem possivelmente associados a diversas características do sistema.

Deste modo, como uma forma de estabelecer alguns passos a serem seguidos para facilitar a identificação de tais interesses , foi estabelecido uma heurística para identificação dos interesses transversais. A heurística foi construída com base nos conceitos de entrelaçamento (tangling) e espalhamento (scattering) dos interesses que estejam presentes nas várias partes dos componentes do sistema.

Sendo assim, segue os critérios estabelecidos:

- Fazer o levantamento das histórias e dos cenários que compõem o sistema. Para auxiliar nesta tarefa, criar uma Tabela de Composição das Histórias (TCH) que irá permitir uma visão simples dos cenários que compõem as histórias;
- Analisar cada história para verificar a existência de novos comportamentos que não estão representados nas histórias atuais, e de comportamentos descritos no corpo dos seus cenários que não são de suas responsabilidades. Caso necessário, atualizar a TCH;
- Analisar a TCH para verificar se existe cenários espalhados e entrelaçados. Se uma coluna da TCH possuir mais de uma marcação, este cenário está presente em mais de uma história e, portanto, espalhado e entrelaçado;
- Montar uma Tabela de Interesses Transversais (TIT) para explicitar os aspectos identificados, assim como, seus espalhamentos e entrelaçamentos;
- Inserir na TIT os comportamentos transversais identificados que não estão representados como cenário de nenhuma das histórias atuais, se houver.

Dessa forma, esses passos irão auxiliar na identificação dos interesses transversais durante a escrita das histórias, assim como, os artefatos gerados (TCH e TIT) irão permitir a identificação e visualização do espalhamento e entrelaçamento dos interesses. No *running example*, com o uso da heurística foi possível evidenciar o requisito de segurança existente e em quais pontos ele está presente, ou seja, em quais histórias este interesse está inserido.

### 3.1.3 Etapa 3 - Modularização dos interesses transversais das histórias

Esta etapa consiste em definir uma forma de separar os interesses que são transversais às histórias e, assim, deixar claro uma modularização inicial desses interesses no momento da escrita das histórias e seus cenários.

Esta separação clara, favorece um melhor entendimento de como o *software* é composto e como os seus módulos estão relacionados. Isso auxilia na formação de uma visão de como os artefatos gerados nas fases posteriores podem ser afetados por tais interesses transversais, e como esse mapeamento de impacto pode ser trabalhado afim de manter uma melhor coesão e um menor acoplamento dos seus módulos (RASHID; MOREIRA; ARAÚJO, 2003).

Deste modo, os interesses identificados que caracterizem um aspecto serão retirados das histórias e dos cenários pelo os quais estão espalhados e entrelaçados, e serão modularizados em histórias específicas, **HistóriasAspectos**.

Sendo assim, os interesses que possam representar comportamentos semelhantes ou que façam parte de um determinado aspecto irão compor uma nova história, que, portanto, irá representar um aspecto identificado na fase anterior.

Assim, será criada uma **HistóriaAspecto** que irá representar um interesse transversal e que conterá todos os cenários (**CenáriosAspectos**) que foram identificados e que representam o comportamento daquele interesse transversal. No caso do *runing example* será criado uma *HistóriaAspecto* para representar o interesse transversal identificado como "Permissão do Usuário". Deste modo, o cenário **C07** será retirado da história "*Gerenciar Usuário*" e inserido na **HistóriaAspecto** "*Permissão*" (Figura 20).

```
Narrative:
In order to verificar as permissões do usuário
As a usuario
I want to ter acesso as operações

Scenario: C07-Deve verificar permissão do usuário
Given o usuário realize uma operação
When a operação for compatível com a autorização lhe dada
Then o usuário está autorizado a realizar a operação
```

**Figura 20 – HistóriaAspecto "Permissão" do *running example*.**

Fonte: Autor

Além disso, esta modularização das histórias em partes independentes, isolando cenários com interesses diferentes, ou seja, os retirando da mesma abstração lógica, permite que tais histórias possam ser melhor entendidas, reutilizadas e modificadas de forma independente.

Para o *running example*, a modularização do interesse transversal relacionado a segurança irá contribuir na realização de possíveis alterações dos requisitos de segurança, visto que agora este interesse está isolado em única história (**HistóriaAspecto "*Permissão*"**). Por exemplo, se as alterações dos salários dos empregados passassem a ser realizadas somente por usuários que tivessem a autorização de "administrador", bastaria modificar a **HistóriaAspecto "*Permissão*"** adicionando mais um cenário para averiguar essa nova situação (Figura 21).

```
Narrative:
In order to verificar as permissões do usuário
As a usuario
I want to ter acesso as operações

Scenario: C07-Deve verificar permissão do usuário
Given o usuário realize uma operação
When a operação for compatível com a autorização lhe dada
Then o usuário está autorizado a realizar a operação

Scenario: C08-Deve permitir que somente usuários
com autorização de administrador alterem salários
Given o usuário solicite alterar salário de um empregado
When o usuário possuir autorização de administrador
Then o usuário está autorizado a realizar a operação
```

**Figura 21 – HistóriaAspecto "Permissão" do *running example* modificada para atender um novo requisito de segurança.**

Fonte: Autor

### 3.1.4 Etapa 4 - Especificação da composição dos interesses transversais nas histórias

Durante um processo de concepção de um sistema com orientação a aspecto, além de ser preciso o levantamento e separação dos interesses que representam os aspectos que serão implementados, também é necessário entender como estes aspectos serão compostos com as outras partes do sistema, de modo a garantir que não haja ambiguidade nesta composição (SOMMERVILLE, 2011).

Feito a modularização, separando os interesses transversais, que foram isolados em histórias específicas, dos demais interesses que compõem as funcionalidades principais do sistema, é necessário estabelecer os pontos de ligações entre essas partes separadas.

Ou seja, é necessário informar de alguma modo que um determinado cenário de uma história necessita da execução de um determinado cenário(s) de uma outra história (aspecto) para continuar, ou, simplesmente, que este(s) cenário(s) seja(m) executado(s) antes ou depois daquele cenário em execução.

É preciso identificar os pontos (*join points*) em alguma parte dos módulos do sistema, que dois ou mais interesses transversais possam ser compostos (Figura 22). A identificação destes pontos representa o local onde os aspectos serão invocados nos outros módulos, ou seja, onde os aspectos irão compor as outras partes do sistema (MOREIRA et al., 2013).

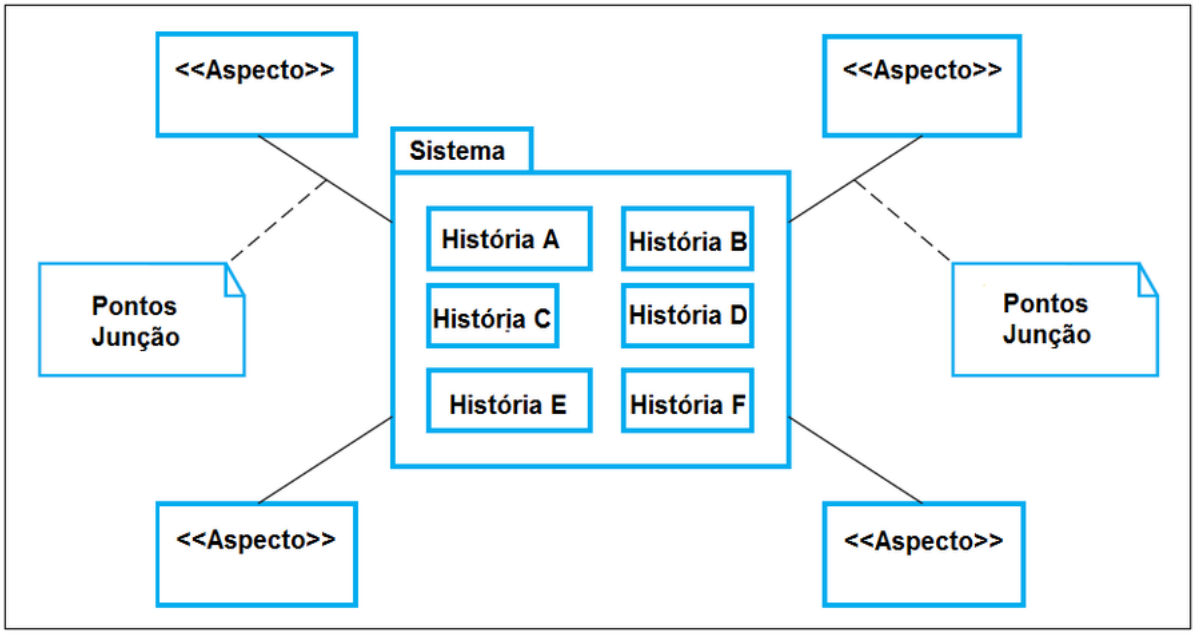


**Figura 22 – Visão geral da composição dos aspectos nas histórias**

Fonte: Sommerville (2011)

É necessário especificar como os interesses transversais que foram modularizados serão integrados (compostos) aos módulos que tais interesses, antes, cruzavam. Logo, a especificação da composição dos aspectos deve definir a forma e as circunstâncias (temporais, condicionais ou incondicionais) de como os interesses transversais serão invocadas nos pontos de ligações que foram identificados nos outros módulos que antes cortavam (MOREIRA et al., 2013).

Deste modo, com intuito de definir a forma de como as **HistóriasAspectos** serão referenciadas nas demais histórias que compõem o sistema, é estabelecido a seguinte estrutura para especificação da composição de um aspecto: **[ OPERADOR-TEMPORAL CENÁRIO-AFETADO: CENÁRIO-ASPECTO ]**. Esta declaração deverá ser inserida nas descrições da **HistóriaAspecto** e delimitada por colchetes. Caso haja a necessidade de inserir mais de uma declaração, visto que um aspecto pode influenciar várias histórias, as declarações devem ser separadas por “ponto e vírgula”. Se várias histórias são afetadas por um mesmo **CenárioAspecto**, elas podem ser listadas em uma mesma declaração, devendo apenas ser separadas por “vírgula”. A declaração de composição do aspecto é composta por três partes:

- A primeira parte determina em qual circunstância temporal (antes, durante ou depois da execução do cenário) o aspecto será invocado – Quadro 2;
- A segunda refere-se ao local onde o aspecto irá interagir (pontos de ligações que foram identificados) – Quadro 3;

- E a última parte informa qual ação o aspecto irá executar, ou seja, qual **Cenário-Aspecto** da **HistóriaAspecto** será executado quando a circunstância temporal for satisfeita para o ponto indicado**.**

| OPERADOR TEMPORAL | DESCRIÇÃO |
|---|---|
| BEFORE | O cenário-aspecto será executado **antes** da execução do cenário-afetado. |
| AFTER | O cenário-aspecto será executado **depois** da execução do cenário-afetado. |
| AROUND | O cenário-aspecto será executado **durante** a execução do cenário-afetado, depois do seu início e antes do seu fim. |

Quadro 2 – Operadores Temporais de uma declaração da composição dos aspectos

Fonte: Autor

| REPRESENTAÇÕES DOS CENARIOS-AFETADOS | DESCRIÇÃO |
|---|---|
| IDEstoria.IDCenario | Identifica o cenário de uma estória que é afetado pelo aspecto. |
| IDEstoria.{IDCenario1, IDCenario2, IDCenarioN} | Identifica vários cenários de uma estória que são afetados pelo aspecto. |
| IDEstoria.* | Informa que todos os cenários de uma estória que é afetado pelo aspecto. |

Quadro 3 – Formas de representar os cenários afetados por aspecto na declaração da composição dos aspectos

Fonte: autor

A declaração de composição do aspecto referente a **HistóriaAspecto** "Permissão" do *running example* está ilustrado na Figura 23. A declaração informa que antes da execução dos cenários **C04** e **C05** da história "Gerenciar Empregado", o **CenárioAspecto C07** da **HistóriaAspecto** "Permissão" será executado**.**

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ before GerenciarEmpregado.{C04,C05}: C07 ]

Narrative:
In order to verificar as permissões do usuário
As a usuario
I want to ter acesso as operações

Scenario: C07-Deve verificar permissão do usuário
Given o usuário realize uma operação
When a operação for compatível com a autorização lhe dada
Then o usuário está autorizado a realizar a operação
```

**Figura 23 – Declaração de composição do AspectoPermissao do *running example*.**

Fonte: Autor

Com base no conceito de *Aspect-Aware* definido por Kiczales e Mezini (2005), e para manter a legibilidade da composição das HistóriasAspectos, poderá também ser inserido uma declaração de ciência em cada história que possuir uma interação com algum aspecto (HistóriasAspectos). Assim, em cada ponto da história que houver a interação será inserida esta declaração para informa que naquele ponto específico de um cenário, há interação de um **CenárioAspecto**. É estabelecido a seguinte estrutura para declaração *Aspect-Aware* na história afetada: **{ ESTÓRIA-ASPECTO – OPERADOR-TEMPORAL CENARIO-ASPECTO}**. Caso haja a necessidade de inserir mais de uma declaração *Aspect-Aware* em um mesmo ponto, estas serão separadas por “ponto e vírgula”. E se mais de um **CenárioAspecto** for executado na mesma declaração, estes devem ser separados por “vírgula”. A declaração *Aspect-Aware* é composta por três partes:

- A primeira parte determina qual aspecto (**HistóriaAspecto**) interagi no ponto que a declaração for inserida;
- A segunda refere-se em qual circunstância temporal o **CenárioAspecto** da **HistóriaAspecto** será executado para aquele ponto específico (Quadro 4);
- E a última parte informa qual ação o aspecto irá executar, ou seja, qual **CenárioAspecto** será executado para aquele ponto específico.

| OPERADOR TEMPORAL | DESCRIÇÃO |
|---|---|
| BEFORE | O cenário-aspecto será executado **antes** da execução do ponto que a declaração for inserida. |
| AFTER | O cenário-aspecto será executado **depois** da execução do ponto que a declaração for inserida. |
| AROUND | O cenário-aspecto será executado **durante** a execução do ponto que a declaração for inserida. |

Quadro 4 – Operadores temporais de uma declaração *Aspect-Aware*
Fonte: autor

Já a declaração *Aspect-Aware* do *running example* é demonstrada na Figura 24. A declaração informa que naquele ponto há a interação da **HistóriaAspecto** “Permissão” e que seu **CenárioAspecto “C07”** será executado.

```
Narrative:
In order to manipular as funcionalidades do empregado
As a usuario
I want to conseguir realizar operações

Scenario: C01-Deve cadastrar o empregado
Given uma solicitação para cadastrar um empregado
When informado nome,idade e salário
Then o empregado deve ser cadastrado

Scenario: C02-Deve calcular o salário do empregado na ativa
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for menor que a idade da aposentadoria
Then o salario dev ser igual 1.000,00
acrescido de 50,00 por cada ano que faltar para aposentadoria

Scenario: C03-Deve calcular o salário do empregado aposentado
Given a idade do empregado
And a idade estabelecida para aposentadoria
When a idade do empregado for maior ou igual a idade da aposentadoria
Then o salario deve ser igual a 1.000,00

Scenario: C04-Espera obter o valor do salário de um empregado
Given solicitação para ver o salário de um empregado !-- DECLARAÇÃO Aspect-Aware
And o usuário seja autorizado a realizar a operação { AspectoPermissao: before C07 }
When informado o empregado que deseja obter o salário
Then o acesso aos dados do salário do empregado é permitido

Scenario:  C05-Espera alterar o valor do salário de um empregado
Given solicitação para alterar o salário de um empregado !-- DECLARAÇÃO Aspect-Aware
And o usuário seja autorizado a realizar a operação { AspectoPermissao: before C07 }
When informado o empregado que deseja alterar o salário
Then a operação é permitida
```

**Figura 24 – Declaração *Aspect-Aware* do *running example*.**
Fonte: Autor

Caso seja inserido declarações *Aspect-Aware* em cada história que for afetada por um aspecto, se houver necessidade de fazer alguma modificação no aspecto, talvez

seja necessário que todas as histórias que possuírem tal aspecto sejam modificadas. No entanto, se for utilizada apenas a declaração de composição do aspecto, caso se deseje saber quais aspectos interagem em uma determinada história, será necessário acessar cada **HistóriaAspecto** para ter essa informação. Deste modo, deve ser analisado qual a melhor abordagem para especificação de composição dos aspectos identificados.

Determinar uma estrutura clara na escrita do teste de como uma **HistóriaAspecto** compõem as outras histórias, também pode evitar que exista outras referências a um mesmo aspecto, o que geraria uma ambiguidade de composição do aspecto com as outras partes do sistema, podendo ocorrer a execução de comportamento inesperado, visto que uma determinada história poderia sofre a interferência de um aspecto que não corresponda ao interesse transversal que o compõem.

Deste modo, as especificações da composição dos aspectos é que poderá permitir que o mapeamento (*weaving*) dos aspectos identificados nesta fase possam ser pensados para as próximas fases, como por exemplo, para a fase de codificação, transformando especificações textuais em especificações executáveis, permitindo que os códigos gerados das especificações das **HistóriaAspecto** possam interagir com os demais códigos gerados das outras histórias não-aspecto.

Para o *running example*, a declaração de composição do "**AspectoPermissao**" (Figura 23) irá permitir, por exemplo, saber exatamente em quais pontos deve-se averiguar quando alguma modificação for realizada na **HistóriaAspecto "Permissão"**: se qualquer modificação for feita no cenário **"C07"** da **HistóriaAspecto "Permissão"**, os cenários **"C04"** e **"C05"** da **História "Gerenciar Empregado"** deveriam ser checados. Isso irá contribuir para verificar se os requisitos antes estabelecidos continuam sendo satisfeitos.

### 3.1.5 Etapa 5 - Mapeamento dos interesses transversais das histórias

O desenvolvimento dirigido por comportamento defende que as histórias de usuários e seus respectivos cenários que representam o comportamento do sistema sejam especificações executáveis, ou seja, que aquelas histórias escritas em formato de texto se transformem em códigos (SMART, 2015). Logo, as histórias não seriam apenas descrição textual do comportamento do sistema, ou uma mera documentação das características do programa, mas sim, especificações que possam ser executadas.

Tornando as especificações textuais em código executáveis, os comportamentos que foram descritos nos cenários poderão ser testados para verificar se todos estão funcionando adequadamente. Existem várias ferramentas que realizam esta tarefa de mapear as histórias para códigos executáveis.

Objetivo desta etapa é mapear os interesses transversais identificados e modula-

rizados nas etapas anteriores para código. No entanto, esta etapa não foi contemplada neste trabalho, visto que seu objetivo é a identificação e separação dos interesses transversais durante a escrita das histórias e de seus respectivos cenários.

Entretanto esta etapa foi incluída como uma fase da abordagem, pois não basta apenas pensar nos interesses transversais durante a processo de engenharia de requisitos para identificá-los e separá-los, mas também é necessário que tais interesses sejam mapeados para as fases posteriores como arquitetura, projeto, implementação, manutenção e evolução (MOREIRA et al., 2013).

Portanto, o mapeamento seria fazer a composição, tecelagem (*weaving*), das partes transversais, que foram isoladas, com as outras partes em que elas cruzavam. Ou seja, seria a integração das **HistóriasAspecto** com as demais histórias que possuem tais aspectos cruzados.

A tecelagem dos aspectos é a inclusão do comportamento de um aspecto (**CenárioAspecto**) nos pontos de ligação (*joint Points*) que foram especificados. Assim, os aspectos passam a ser integrados ao sistema para que os interesses transversais sejam executados nos lugares certos da versão final do sistema (MOREIRA et al., 2013).

É importante ressaltar que o mapeamento dos interesses transversais para os outros módulos não é necessariamente uma regra (MOREIRA et al., 2013). Às vezes, apenas a especificação dessa composição, que é quando se determina quais elementos aspectuais, de que forma e em quais circunstancias temporais, condicionais ou incondicionais estes elementos irão interferir em outros módulos, já seja algo suficiente para que se possa ter noção das interações que as partes aspectuais e não aspectuais podem sofrer. E assim, poder permitir que seja feito projeções de possíveis restrições e influências que os aspectos, de algum modo, possam gerar as outras partes, sem integrar fisicamente.

# 4 EXEMPLO DEMONSTRATIVO DA ABORDAGEM

Nesta seção será utilizado um exemplo demonstrativo para aplicar as etapas definidas pela abordagem proposta neste trabalho, na qual é utilizando os conceitos de orientação a aspecto para auxiliar na escrita das histórias e seus cenários que representam especificações textuais dos testes criados com a notação *Gherkin*.

O exemplo ilustrativo utilizado é o A*utomated Teller Machine* (ATM), que simula o funcionamento de um caixa de banco automático. Foi escolhido por ser comum o seu uso na literatura para aplicação dos conceitos de orientação aspecto e por sua documentação completa e disponível no endereço eletrônico: http://www.math-cs.gordon.edu/courses/cs211/ATMExample/index.html. De forma resumida, o escolha do exemplo demonstrativo teve as seguintes motivações:

- É muito utilizado na literatura de orientação a aspecto;
- A presença de interesses transversais nas descrições do exemplo;
- Completude da documentação encontrada sobre o exemplo: casos de uso, diagramas de classes, diagramas de sequência, casos de teste, entre outros artefatos;
- E fato de o exemplo está modelado segundo paradigma Orientado a Objetos, ou seja, algumas modularizações já foram aplicadas.

## 4.1 Exemplo demonstrativo ATM

O exemplo ilustrativo ATM refere-se à simulação de algumas operações que um cliente pode executar em um caixa de banco automático, de modo que poderá interagir com sistema usando um cartão magnético. O cliente pode realizar depósitos em suas contas, saques, transferências, consultas a informações das suas contas, assim como, receber recibos das operações realizadas. A documentação contém mais informações sobre o funcionamento e detalhes de execução das transações. Abaixo a modelagem dos casos de usos para o exemplo demonstrativo (Figura 25).

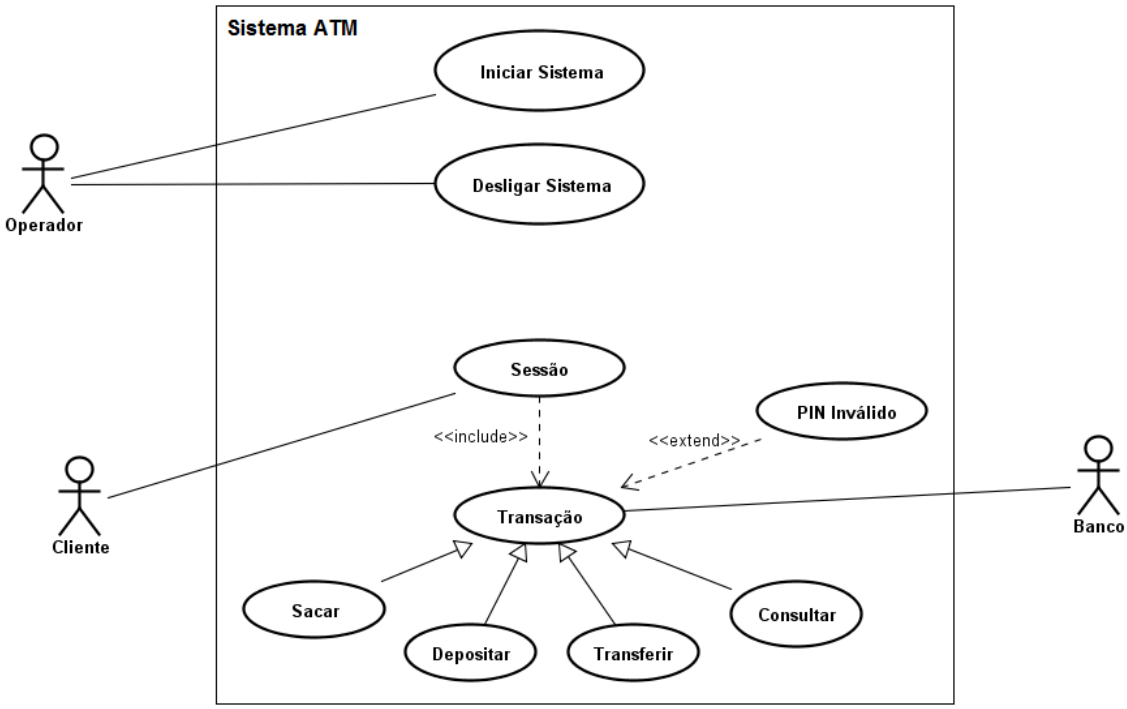


**Figura 25 – Casos de Usos do exemplo demonstrativo ATM.**

Fonte: Documentação ATM

Como se pode observar pela modelagem dos casos de uso, algumas modularizações já foram inseridas neste momento:

- A criação do caso de uso "*Sessão"* para gerenciar as sessões criadas pelo cliente na máquina;
- A generalização feita dos casos de uso "*Sacar"*, "*Depositar"*, "*Transferir"*, "*Consultar*" para o "*Transação"*;
- E a criação de um caso de uso extensivo chamado de "*PIN Inválido*" que é responsável por tratar as exceções que possam ocorrer durante a validação da senha do cliente nas transações que ele realizar.

## 4.2 Aplicação da abordagem

As próximas seções irão tratar de como foi escrito as histórias e seus respectivos cenários, do processo realizado para identificação dos interesses transversais, como foi realizado a modularização desses interesses transversais em HistóriasAspectos (aspectos), e como foi realizada a especificação de composição das HistóriasAspectos com as demais histórias do exemplo demonstrativo.

### 4.2.1 Escrita das histórias e cenários ATM

A escrita das histórias e dos seus respectivos cenários foram baseadas nas informações que descrevem as funcionalidades e nos demais artefatos contidos na documentação. Também foi utilizado a seguinte abordagem de correlação da estrutura dos casos de testes (presentes na documentação) com a estrutura da escrita dos cenários (*Given-When-Then*), conforme ilustrado na Figura 26 (FELLER, 2015 *apud* PUGH, 2010).

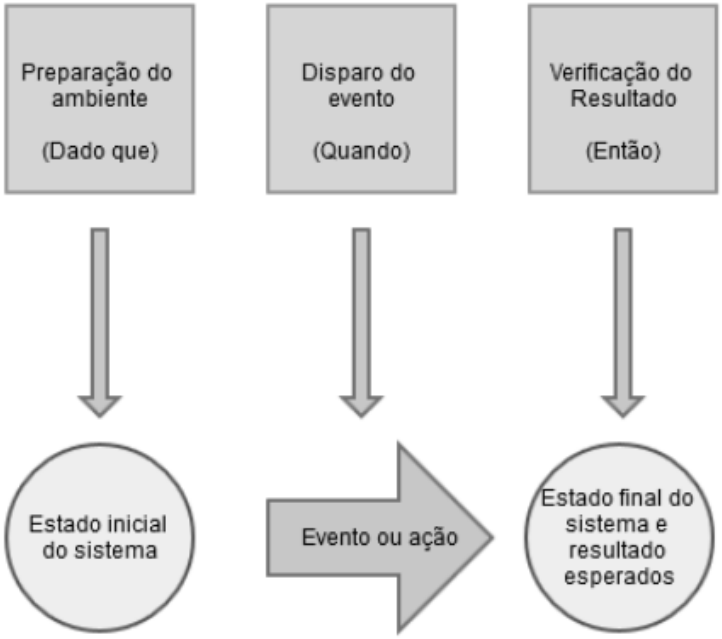


**Figura 26 – Mapeamento dos Casos de Teste para os Cenários**
Feller (2015)

A documentação fornece vários casos de testes que foram escritos para os casos de usos existentes. Os casos de uso são artefatos utilizados para representar as funcionalidades de um sistema e como as pessoas (atores) utilizam tais funcionalidades para interagir (BEZERRA, 2015).

As histórias, por sua vez, também são formas de representar as funcionalidades existentes no sistema, e os cenários, os possíveis comportamentos que elas podem ter. Deste modo, para cada caso de uso foi criado uma respectiva história para representa aquele interesse do sistema. Assim como foi feito com os casos de testes levantados para cada caso de uso, estes passaram, nas histórias, a representar os seus comportamentos esperados, tronando-se seus cenários.

Conforme demonstrado na Figura 27, as informações dos casos de testes foram mapeados para os cenários das histórias. Os casos de teste foram descritos no formato de uma tabela, onde as informações estão organizadas nas colunas: caso de uso, função testada, estado inicial do sistema, entrada e saída esperada .

| Caso de Uso | Função Testada | Estado Inicial | Entrada | Saída Esperada |
|---|---|---|---|---|
| *Iniciar Sistema* | *O sistema é iniciado quando ativar a chave de ligar.* | *Sistema desligado.* | *Ativar Chave* | *Sistema liga e solicita montante inicial em dinheiro.* |

**Figura 27 – Exemplo do formato da tabela dos casos de teste ATM.**

Fonte: Documentação do exemplo ATM.

O mapeamento foi feito da seguinte forma para os cenários: as informações contidas nas colunas '*Função Testada'* serviram para compor a descrição do cenário; as informações descritas nas colunas '*Estado Inicial'* foram utilizadas na cláusula '*Given'* do cenário; as informações descritas nas colunas 'Entrada*'* foram inseridas na cláusula '*When'* do corpo do cenário; e as descrições das colunas '*Saída Esperada'* serviram para compor a cláusula *'Then'* do cenário.

Assim foram criadas as seguintes histórias do exemplo ilustrativo ATM: *Inicializar Sistema, Desligar Sistema, Sessão, Transação, Saque, Depósito, Transferência, Consulta e PIN Inválido*. Na Figura 28, um exemplo de como ficou a história *Inicializar Sistema*. As demais histórias do exemplo demonstrativo ATM encontram-se no apêndice A .

```
Narrative:
In order to ligar uma máquina ATM
As a operador da máquina ATM
I want to inicializar o sistema da máquina adequadamente

Scenario: C01-Deve inicializar sistema na máquina ATM
Given o sistema desligado
When ativar a chave de ligar
Then sistema solicita montante inicial em dinheiro.

Scenario: C02-Espera validar o montante inicial em dinheiro
Given sistema solicitando uma quantia em dinheiro
When insirido um valor legítimo
Then sistema liga

Scenario: C03-Deve testar primeira conexão com banco
Given sistema ligado com sucesso
When realizada uma transação teste válida
Then sistema deve informar conexão estabelecida
```

**Figura 28 – História "Inicializar Sistema" do exemplo ATM**

Fonte: Autor

**Validação profissional da etapa**

Como forma de corroborar a abordagem de mapeamento dos casos de testes para os cenários das histórias e como forma de reafirmar que casos de testes foram corretamente escritos, após este procedimento os cenários passaram pela avaliação de dois profissionais experientes na área de testes e , também, com experiência em trabalhar com a metodologia ágil BDD e que, portanto, os mesmos dentêm conhecimentos técnicos e práticos com a escrita de histórias com a notação *Gherkin*.

Os profissionais ratificaram a forma como foi feito o mapeamento dos casos de testes para os cenários e fizeram as seguintes observações:

- As informações contidas nas histórias devem ser claras, pois estas também servem como documentação do sistema;
- Os cenários geralmente são expressos em linguagem de desejo ou iniciados por verbo, sendo mais comum em BDD serem escritos com forma de desejo ou expectativa.

Após as observações colocadas pelos profissionais, os cenários foram corrigidos e a próxima etapa da abordagem foi aplicada.

### 4.2.2 Identificação dos interesses transversais ATM

A intenção desta etapa é identificar possíveis interesses espalhados e entrelaçados pelas histórias. Verificar a existência de interesses que possam representar cenários pertencentes a outras histórias, ou que represente interesses que não deveriam ser tratados naquela história, ou seja, averiguar interesses transversais. Para isso será seguida a heurística definida pela abordagem para identificação de tais interesses transversais. Assim foi criado a Tabela de Composição das Histórias (Tabela 3).

**Tabela 3 – Tabela de Composição das Histórias do exemplo ATM**

| HISTÓRIAS | CENÁRIOS | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | ... | 11 | 12 | 13 | 14 | 15 | ... | 21 | 22 | ... | 28 | 29 | 30 | 31 | 32 | ... | 39 | 40 | ... | 49 |
| Inicializar Sitema | x | x | x | | | | | | | | | | | | | | | | | | | | | | | |
| Desligar Sistema | | | | x | x | | | | | | | | | | | | | | | | | | | | | |
| Sessão | | | | | | x | x | x | | | | | | | | | | | | | | | | | | |
| Transação | | | | | | | | | x | x | x | | | | | | | | | | | | | | | |
| Saque | | | | | | | | | | | | x | x | x | | | | | | | | | | | | |
| Transferência | | | | | | | | | | | | | | | x | x | x | | | | | | | | | |
| Consulta | | | | | | | | | | | | | | | | | | x | x | x | | | | | | |
| Depósito | | | | | | | | | | | | | | | | | | | | | x | x | x | | | |
| PIN Inválido | | | | | | | | | | | | | | | | | | | | | | | | x | x | x |

Fonte: Autor

Na primeira análise da TCH não foi identificado cenários inteiros com escrita literal ou semântica repetida em outras histórias. Isso talvez tenha ocorrido por que as histórias foram derivadas de um estudo de caso já modularizado. No entanto, em uma segunda análise foi observado que mesmo não existindo literalmente um cenário escrito na sua forma padrão (*Given-When-Then*), havia indícios de interesses presentes no corpo de alguns cenários que se referenciavam a cenários pertencentes a outras histórias. Apesar da modularização existente, ainda havia "chamadas" espalhadas em vários cenários de interesses que não eram das suas reponsabilidades, e sim de outras histórias. A história "Consulta", por exemplo, contém em seus cenários interesses que deveria ser tratados em outras histórias: sessão ativa (história *Sessão*), imprimir recibo, registrar log e exibir mensagem apropriada (Figura 29).

```
Narrative:
In order to realizar uma transação de consulta a minhas contas
As a cliente
I want to consultar minhas contas com sucesso

Scenario: C29-O sistema deve solicitar ao cliente a escolha de uma conta para consulta
Given uma sessão ativa e o menu de tipos de transação
When escolhida a transação consulta
Then o sistema exibe o menu com os tipos de conta para consulta

Scenario: C230-Espera-se que o sistema execute uma transação de consulta legítima corretamente
Given uma sessão ativa e o menu com os tipos de conta para consulta
When uma conta é escolhida
Then O sistema imprime um recibo com saldo correto
And registra a transação no log

Scenario: C31-Espera-se que o sistema cancele consulta antes de escolher uma conta
Given uma sessão ativa e o menu com os tipos de conta
When escolher a opção cancelar
Then o sistema exibe uma mensagem apropriada
And oferece ao cliente a opção de fazer outra transação
```

**Figura 29 – Interesses transversais identificados na História "Consulta" do exemplo ATM.**

Fonte: Autor

Após refazer a TCH com esta compreensão (Figura 30), passou-se a referenciar os cenários que estavam sendo invocados em outros cenários, como cenários também pertencentes àquela história. Assim, ficou evidenciado o espalhamento e entrelaçamento dos interesses.

| HISTÓRIAS | CENÁRIOS | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | ... | 11 | 12 | 13 | 14 | 15 | ... | 21 | 22 | ... | 28 | 29 | 30 | 31 | 32 | ... | 39 | 40 | ... | 49 |
| Inicializar Sitema | x | x | x | | | | | | | | | | | | | | | | | | | | | | | |
| Desligar Sistema | | | | x | x | | | | | | | | | | | | | | | | | | | | | |
| Sessão | | | | | | x | x | x | | | | | | | | | | | | | | | | | | |
| Transação | | | | | | | | | x | x | x | | | | | | | | | | | | | x | | x |
| Saque | | | | | | | | x | | | | x | x | x | | | | | | | | | | | | |
| Transferência | | | | | | | | x | | | | | | | x | x | x | | | | | | | | | |
| Consulta | | | | | | | | x | | | | | | | | | | x | x | x | | | | | | |
| Depósito | | | | | | | | x | | | | | | | | | | | | | x | x | x | | | |
| PIN Inválido | | | | | | | | | | | | | | | | | | | | | | | | x | x | x |

**Figura 30 – TCH do exemplo ATM após identificação dos interesses transversais.**

Fonte: Autor

Deste modo, foi montada a Tabela de Interesses Transversais para explicitar em quais histórias os interesses identificados estão presentes. Foram identificados os seguintes interesses transversais: verificação de sessão ativa, geração de Log, tratamento de exceção, emissão de recibo, informativos sobre transação (Tabela 4).

**Tabela 4 – Tabela de Interesses Transversais do exemplo ATM**

| **Interesses Transversais** | **Histórias** | | | | |
|---|---|---|---|---|---|
| | Transação | Saque | Depósito | Transferência | Consulta |
| Sessão Ativa | **0 cenário** | **Todos os cenários** | **Todos os cenários** | **Todos os cenários** | **Todos os cenários** |

| **Interesses Transversais** | **Histórias** | | | | |
|---|---|---|---|---|---|
| Geração de *Log* | **0 cenário** | **1 Cenário** | **1 Cenário** | **1 Cenário** | **1 Cenário** |
| Tratamento de Exceção | **Todos os cenários** | **0 cenário** | **0 cenário** | **0 cenário** | **0 cenário** |
| Emissão de Recido | **0 cenário** | **1 Cenário** | **1 Cenário** | **1 Cenário** | **1 Cenário** |
| Informativos Transação | **0 cenário** | **4 Cenários** | **4 Cenários** | **3 Cenários** | **1 Cenário** |

Fonte: Autor

Através da tabela é possível ver que várias histórias implementam interesses que não são de suas responsabilidades, portanto, estes interesses transversais devem ser modularizados.

### 4.2.3 Modularização dos interesses transversais ATM

Os interesses transversais identificados (aspecto) serão modularizados em HistóriasAspectos, e os comportamentos relativos a tais interesses (CenáriosAspectos) irão compor a nova história. Com a aplicação da abordagem foi possível identificar mais três interesses transversais que não foram evidenciados no documento de requisitos do exemplo ilustrativo ATM: Geração de Log, Emissão de Recibo e Informativos da Transação.

Os outros interesses, verificação de sessão ativa e tratamento de exceção, já foram modularizados em histórias específicas quando da conversão dos casos de testes. Deste modo, o interesse que verifica se uma sessão está ativa foi retirado das histórias que ele cortava e isolado na história de sua responsabilidade (História Sessão), que passou a ser considerada uma HistóriaAspecto, por que algumas de suas responsabilidades são invocadas em outras histórias.

Já a história PIN inválido reuni os cenários que tratam de exceção da senha. Foi encontrado outros tipos de exceções e uma nova história (HistóriaAspecto **Exceção**) foi criada para encapsular o tratamento não apenas da inserção incorreta de senhas,

mas também qualquer falha que ocorra durante uma transação. As Figuras 31 e 32 representam como ficou a HistóriaAspecto **Exceção**.

```
Narrative:
In order to representar as exceções ocorridas
As a cliente
I want to usar ATM sem problemas

Scenario: C40-Espera-se que sistema permita reinserção do PIN
Given cliente inserir PIN incorreto
When tentar realizar uma transação
Then a transação não é permitida
And solicitado reinserção do PIN

Scenario: C41-Espera-se que sistema permita transação após reinserção de PIN correto
Given solicitação do sistema para o cliente reinserir PIN
When inserido o PIN correto
Then a transação é concluída com sucesso

Scenario: C42-Espera-se que sistema permita a execução de nova transação após reinserção de PIN correto
Given reinserção correta do PIN pelo cliente
When cliente executar outra transação
Then a nova transação é concluída com sucesso

Scenario: C43-Espera-se que o sistema permita uma segunda reinserção de PIN
Given solicitação do sistema para o cliente reinserir PIN
When inserido o PIN incorreto
Then uma mensagem adequada é exibida
And o sistema solicita reinserção do PIN pela segunda vez

Scenario: C44-Espera-se que sistema permita transação após a segunda reinserção de PIN correto
Given o sistema solicita reinserção do PIN pela segunda vez
When inserido o PIN correto
Then a transação é concluída com sucesso
```

**Figura 31 – Primeira parte da HistóriaAspecto Exceção do exemplo ATM**

Fonte: Autor

```
Narrative:
In order to representar as exceções ocorridas
As a cliente
I want to usar ATM sem problemas

Scenario: C45-Espera-se que o sistema permita uma terceira reinserção de PIN
Given o sistema solicita reinserção do PIN pela segunda vez
When inserido o PIN incorreto
Then uma mensagem adequada é exibida
And o sistema solicita reinserção do PIN pela última vez

Scenario: C46-Espera-se que sistema permita transação após a terceira reinserção de PIN correto
Given o sistema solicita reinserção do PIN pela terceira vez
When inserido o PIN correto
Then a transação é concluída com sucesso

Scenario: C47-O sistema deve reter cartão do cliente caso ocorra 3 reinserções incorretas do PIN
Given o sistema solicita reinserção do PIN pela terceira vez
When inserido o PIN incorreto
Then uma mensagem adequada é exibida, cartão é retido pela máquina e a sessão termina

Scenario: C48-O sistema deve tratar falha na transação que não seja por senha inválida
Given a execução de uma transação
When ocorrer uma falha qualquer que não senha inválida
Then uma explicacao é exibida
And o sistema perguta ao cliente se deseja outra transação

Scenario: C49-Espera-se que o sistema recuse um cartão ilegível
Given sistema ligado e desocupado
When insiro um cartão ilegível
Then cartão é rejeitado,uma tela de erro é exibida e pode ser iniciada nova sessão
```

**Figura 32 – Segunda parte da HistóriaAspecto Exceção do exemplo ATM**

Fonte: Autor

Para cada novo interesse transversal encontrado foi criado uma HistóriaAspecto (**HistóriaAspecto Log**, **HistóriaAspecto Emissão de Recibo** e **HistóriaAspecto Informativo Transação**) com os seus respectivos CenáriosAspectos, assim como, foram retiradas as chamadas aos CenáriosAspectos das histórias que os invocam. As **HistóriasAspecto** encontram-se no apêndice A. A Figura 33 mostra a história "Consulta" após o isolamento dos interesses transversais que nela existiam.

```
Narrative:
In order to realizar uma transação de consulta a minhas contas
As a cliente
I want to consultar minhas contas com sucesso

Scenario: C29-O sistema deve solicitar ao cliente a escolha de uma conta para consulta
Given a exibição do menu de tipos de transação
When escolhida a transação consulta
Then o sistema exibe o menu com os tipos de conta para consulta

Scenario: C230-Espera-se que o sistema execute uma transação de consulta legítima corretamente
Given a exibição do menu com os tipos de conta para consulta
When uma conta é escolhida
Then o consulta é realizada com sucesso

Scenario: C31-Espera-se que o sistema cancele consulta antes de escolher uma conta
Given a exibição do menu com os tipos de conta
When escolher a opção cancelar
Then o cancelamento é realizado com sucesso
And o sistema oferece ao cliente a opção de fazer outra transação
```

**Figura 33 – História Consulta do exemplo ATM sem referência aos aspectos.**

Fonte: Autor

Assim, os interesses transversais foram modularizados em histórias específicas e retirados das histórias que os invocam. Além disso, novos interesses podem ser inseridos nas novas histórias, fazendo com que cada história trate apenas de interesses que sejam de suas responsabilidades. O Quadro abaixo (Figura 34) mostra como ficou a modularização das histórias para o exemplo ilustrativo ATM.

| HistóriasAspectos | | | | | | |
|---|---|---|---|---|---|---|
| **Sessão** | **Log** | **Exceção** | **Emissão Recibo** | **Informativos Transação** | | |
| **MODULARIZAÇÃO** | | | | | | |
| **Histórias** | | | | | | |
| **Inicializar Sistema** | **Desligar Sistema** | **Transação** | **Transferência** | **Saque** | **Consulta** | **Depósito** |

**Figura 34 – Modularização das histórias do exemplo ATM.**

Fonte: Autor

### 4.2.4 Especificação da composição das HistóriasAspecto ATM

Depois de separado os aspectos encontrados em histórias específicas é necessário estabelecer como elas irão interagir, por meio de seus CenáriosAspectos, como as outras histórias. A abordagem estabelece que esta ligação (Figura 35) é permitida por especificações de composição dos aspectos.

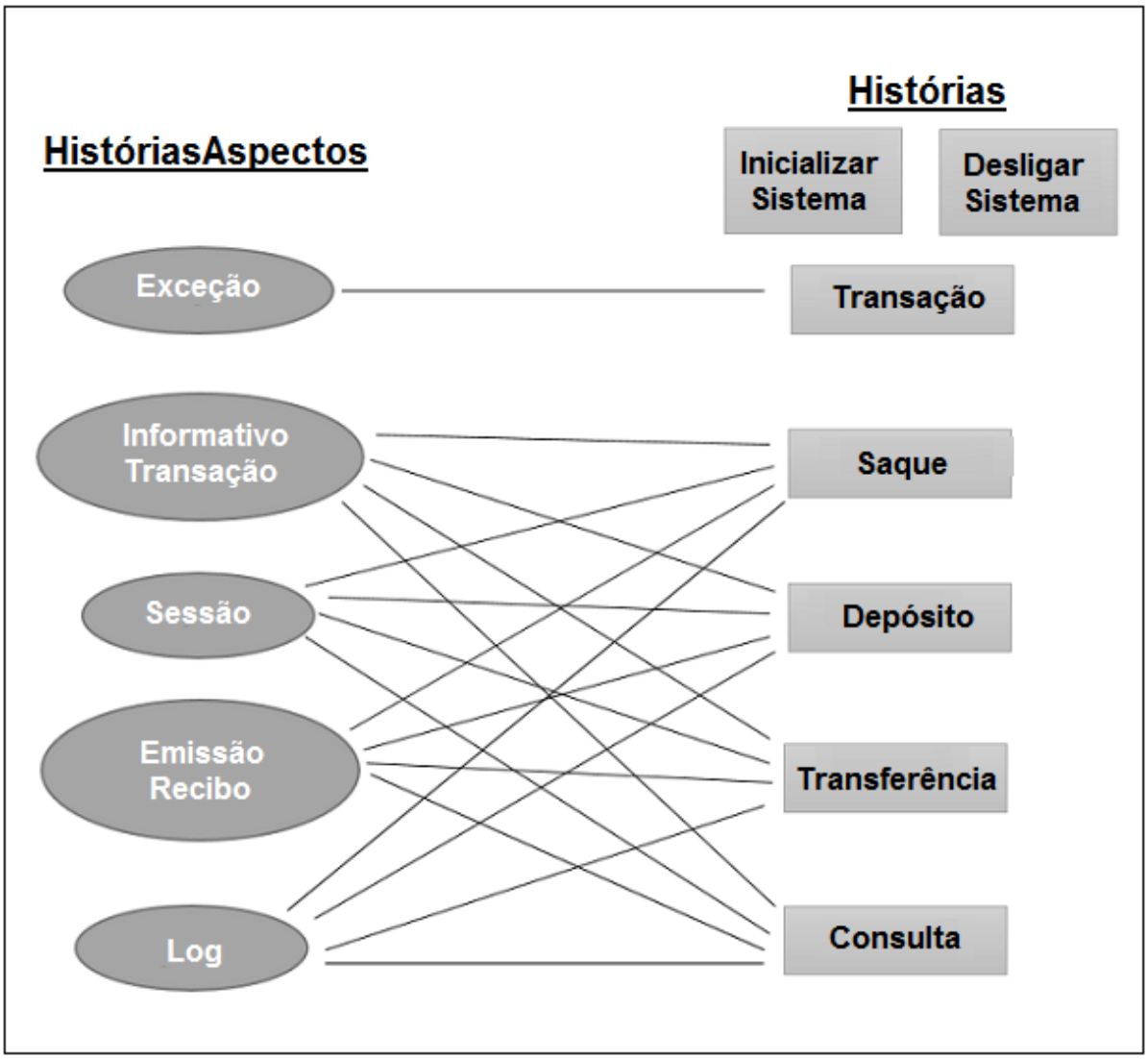


**Figura 35 – Ligações das HistóriasAspectos com as demais histórias do exemplo ATM.**
Fonte: Autor

Em cada HistóriaAspecto deve ter uma declaração informado em quais pontos das outras histórias, de que forma e quais CenáriosAspetos são invocados. Segue a descrição das declarações de composição de cada HistóriaAspecto:

- A **HistóriaAspecto Sessão** integari com as histórias "*Saque", "Depósito", "Transferência" e "Consulta".* Antes da execução dos seus cenários o **CenárioAspecto** "*sessão ativa*" (**C11**) é executado.

- A **HistóriaAspecto Emissão Recibo** interagi com a história "*Saque"* depois da execução do cenário "*Execução da Transação Saque*" (**C18**); na história "*Depósito"* depois da execução do cenário "*Execução da Transação Deposito*" (**C35**); na história *"Transferêcia"* depois da execução do cenário "*Execução da Transação Transferência*" (**C25**); na história "*Consulta"* depois da execução do cenário "*Execução da Transação Consulta*" (**C30**);

- A **HistóriaAspecto InformativoTransação** interagi com a história "*Saque"* depois da execução do cenário "*Verificar se ATM possui dinheiro suficiente*" (**C17**), depois da execução do cenário "*Saldo suficiente para saque*" (**C19**), depois da execução dos cenários "*Cancelar saque*" (**C20** e **C21**); na história *"Depósito"* depois da execução dos cenários "*Cancelar depósito*" (**C36**, **C37**, **C38** e **C39**); na história *"Transferência"* depois da execução dos cenários "*Cancelar transferência*" (**C26**, **C27** e **C28**); na história *"Consulta"* depois da execução do cenário "*Cancelar consulta*" (**C31**);

- A **HistóriaAspecto Log** interagi com a história "*Saque"* depois da execução do cenário "*Execução da Transação Saque*" (**C18**); na história "*Depósito"* depois da execução do cenário "*Execução da Transação Depósito*" (**C35**); na história "*Transferência"* depois da execução do cenário "*Execução da Transação Transferência*" (**C25**); na história "*Consulta"* depois da execução do cenário "*Execução da Transação Consulta*" (**C30**);

- A **HistóriaAspecto Exceção** interagi com a história "*Transação*" depois do cenário "*Manipula Senha inválida*" (**C12**), do cenário "*Manipula Falha na Transação*" (**C13**) e do cenário "*Manipula Erro na Leitura do Cartão*" (**C14**).

Conforme estabelecido pela abordagem, a Figura 36 mostra a **Declaração de Composição do Aspecto** para a **HistóriaAspecto "**Sessão*",* indicando que ela interagi com as histórias "Saque", "Transferencia", "Consulta" e "Deposito" em todos os seus cenários (representado pelo o uso do asterisco) e que o seu **CenárioAspecto "C11"** é o cenário executado nas interações.

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ before Saque.*, Transferencia.*, Consulta.*, Deposito.* : C11 ]

Narrative:
In order to acessar minha conta bancária com cartão ATM
As a cliente
I want to ter acesso as operacões bancárias

Scenario: C06-O sistema deve solicitar a senha após leitura do cartão ATM de um cliente
Given sistema ligado e desocupado
When insiro um cartão legível
Then cartão é aceito
And o sistema solicita a entrada da senha

Scenario: C07-Espera-se que o sistema exiba o menu com as operações
Given o sistema está solicitando a entrada da senha
When insirir uma senha correta
Then o sistema exibe um menu com os tipos de operações
.
.
.
Scenario: C11-O sistema deve verificar sessão ativa para permitir nova transação
Given cliente iniciou uma sessão no ATM
When o tempo da sessão estiver ativo
Then a sessão está ativa e transações podem ser realizadas
```

**Figura 36 – Declaração de Composição da Aspecto para a HistóriaAspecto “Sessão” do exemplo ATM.**

Fonte: Autor

Além da declaração de composição do aspecto, a abordagem também estabele que cada história, nos pontos que recebem a interação com o aspecto (HistóriaAspecto), deve possuir a **Declaração *Aspect-Aware***. A Figura 37 representa a história “*Consulta*“ com suas **Declarações *Aspect-Aware*** nos pontos em que interage com com a **HistóriaAspecto “InformativoTransação”**, a **HistóriaAspecto “Log”**, a **HistóriaAspecto “Emissão Recibo”** e a **HistóriaAspecto “Sessão”**. As demais histórias com seus respectivos cenários, declarações de composição dos aspectos e declarações *aspect-aware* encontram-se no apêndice A deste trabalho.

```
Narrative:
In order to realizar uma transação de consulta a minhas contas
As a cliente
I want to consultar minhas contas com sucesso

Scenario: C29-O sistema deve solicitar ao cliente a escolha de uma conta para consulta
Given a exibição do menu de tipos de transação !-- DECLARAÇÃO Aspect-Aware { AspectoSessao: before C11 }
When escolhida a transação consulta
Then o sistema exibe o menu com os tipos de conta para consulta

Scenario: C230-Espera-se que o sistema execute uma transação de consulta legítima corretamente
Given a exibição do menu com os tipos de conta para consulta { AspectoSessao: before C11 }
When uma conta é escolhida                !-- DECLARAÇÃO Aspect-Aware
Then o consulta é realizada com sucesso { AspectoLog: after C50; AspectoRecibo: after C51 }

Scenario: C31-Espera-se que o sistema cancele consulta antes de escolher uma conta
Given a exibição do menu com os tipos de conta { AspectoSessao: before C11 }
When escolher a opção cancelar                !-- DECLARAÇÃO Aspect-Aware
Then o cancelamento é realizado com sucesso { AspectoInformativoTransacao: after C52 }
And o sistema oferece ao cliente a opção de fazer outra transação
```

**Figura 37 – Declaração *Aspect-Aware* para a História “Consulta” do exemplo ATM.**

Fonte: Autor

## 4.3 Considerações sobre a aplicação da abordagem

O exemplo ilustrativo utilizado forneceu alguns artefatos que auxiliaram na criação das histórias com a notação *Gherkin* e na identificação dos interesses. No entanto, com a aplicação da abordagem foi possível identificar interesses transversais que não estavam evidenciados na documentação. Nesse processo de identificação e modularização dos interesses transversais, a construção da Tabela de Composição das Histórias (TCH) deve ser feita atentamente, visto que podem existir cenários que estão sendo citados em cenários de outras estórias e que, estariam compondo também esta história.

Em se tratando da Tabela de Interesses Transversais (TIT) esta pode ser útil para auxiliar na rastreabilidade de mudança feitas nas histórias, assim como, as declarações de composição das HistóriasAspectos (aspecto), podem contribuir para esta atividade. Logo, se algum comportamento da HistóriaAspecto for modificado, seria possível utilizando a TIT e a declaração de composição, identificar histórias e cenários que deveriam ser verificados para averiguar se os seus comportamentos não foram afetados com tal modificação.

A automatização dos interesses transversais, definidos textualmente, para código, seria uma atividade a ser implementada em trabalhos futuros, considerando que as **Declarações de Composição dos Aspectos** contidas nas **HistóriasAspectos** e as **Declarações *Aspect-Aware*** contidas nas demais histórias, podem ser usadas para auxiliar no possesso de tecelagem (*weaving*).

# 5 CONCLUSÕES

Neste capítulo é apresentado as conclusões deste trabalho: na seção 5.1 são colocadas as contribuições do trabalho, na seção 5.2 apresentada as limitações da pesquisa e na seção 5.3 as ações futuras.

## 5.1 Contribuições do trabalho

O estudo verificou a necessidade de uma maior brevidade no entendimento dos requisitos de um sistema e a forma como estão relacionados, para que se possa escrever testes que retratarem bem estes requisitos e suas relações.

Este trabalho estabelece uma abordagem com etapas bem definidas para auxiliar na identificação e modularização de interesses transversais para ser aplicado na escrita de testes com a notação *Gherkin*. Ressalta-se que a aplicação desta abordagem pode:

- Permitir um planejamento para evitar que as especificações de histórias/cenários transversais, quando transformadas em especificações executáveis (códigos), sejam mapeadas juntas, gerando especificações executáveis também transversais;
- Possibilitar o entendimento refinado dos comportamentos (cenários) de uma história e sua interferência em outras histórias, ou seja, das interações existentes;
- Facilitar a manutenção e a evolução das histórias/cenários (por meio da inserção e remoção) visto que os interesses transversais (*HistóriasAspectos*) estão adequadamente isolados, modularizados.

## 5.2 Limitações

Considerando que o objetivo deste trabalho é realizar a aplicação dos conceitos de orientação a aspecto para identificação e modularização de interesses transversais especificamente na escrita dos testes com a notação *Gherkin* no formato de histórias do usuário, ressalta-se que há limitações neste trabalho, sendo estas:

- A metodologia utilizada para realizar as pesquisas pode não ter sido eficiente o suficiente para retornar informações que sejam úteis sobre as características dos modelos de aplicação de orientação a aspecto na engenharia de requisitos;
- Aplicou-se a abordagem apenas em um exemplo demonstrativo, havendo a necessidade de ser aplicada estudos de casos mais robustos para averiguação de seus possíveis ganhos quanto a sua utilização;

- A utilização de métricas pode gerar informações importantes para medição e controle, que não foram levadas em consideração para medir a separação de interesses, e averiguar o impacto da identificação e modularização das histórias.

## 5.3 Trabalhos futuros

Como base nas limitações levantadas, nas necessidades percebidas durante a pesquisa e da aplicação da abordagem no exemplo demonstrativo, propõe-se as seguintes sugestões para trabalhos futuros:

- Aplicação da abordagem em estudos de casos maiores e a utilização de métricas de separação de interesses como *Concern Diffusion over Operations* (CDO) e *Concern Diffusion over Components* (CDC) para analisar a importância e o impacto das histórias transversais na escrita dos testes com a notação *Gherkin*;
- Desenvolvimento de uma ferramenta para automatizar a criação e alteração dos artefatos gerados pela abordagem na identificação dos interesses transversais: Tabela de Composição das Histórias (TCH) e Tabela de Interesses Transversais (TIT);
- Desenvolvimento de uma ferramenta para gerar testes de aceitação automatizados com *AspectJ* com base na proposta da abordagem para escrita dos testes com a notação *Gherkin*.

## 5.4 Considerações Finais

Esse trabalho, procurou usar os conceitos de AORE com a finalidade de definir etapas claras para a escrita de histórias do usuário com a notação *Gherkin*, visto que este formato de escrita dos testes tem se tornado comum com o uso da metodologia BDD, mas que não foi percebido durante as pesquisas a preocupação de identificar e modularizar interesses transversais neste momento.

Além disso, conforme proposto nessa abordagem, identificar e modularizar interesses transversais no momento da criação das histórias do usuário pode ser um ponto de partida para automatização de testes de aceitação orientados a aspecto escritos neste formato. Também pode ser considerado uma estratégia útil para proporcionar aos *stakeholders* um melhor entendimento dos comportamentos de cada funcionalidade criada, visto que as histórias estarão melhor modularizadas.

# REFERÊNCIAS

BECK, K. **Test-Driven Development By Example**. [S.l.]: Addison-Wesley Professional, 2002.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto Ágil**. 2001. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: Outubro de 2015.

BERNARDO, P. C. **Padrões de testes automatizados**. 2011. Dissertação (MESTRADO EM CIÊNCIA DA COMPUTAÇÃO) — UNIVERSIDADE DE SÃO PAULO.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML** . 3. ed. [S.l.]: Campus, 2015.

BILAL, H. A.; ILYAS, M.; TARIQ, Q.; HUMMAYUN, M. Requirements Validation Techniques: An Empirical Study . **International Journal of Computer Applications**, v. 148, n. 14, p. 0975 – 8887, Agosto 2016.

CEZERINO, A.; NASCIMENTO, F. P. UTILIZAÇÃO DA TÉCNICA DE DESENVOLVIMENTO ORIENTADO POR COMPORTAMENTO (BDD) NO LEVANTAMENTO DE REQUISITOS. **Revista Interdisciplinar Científica Aplicada**, v. 10, p. 40 – 51, 2016.

CHERNAK, Y. Implementing Aspect-Oriented Requirements Analysis for Investment Banking Applications. In: ____. **Aspect-Oriented Requirements Engineering**. 1. ed. [S.l.]: Springer, 2013. cap. 15, p. 289 – 316.

COMMUNITY, D. **What is Domain-Driven Design?** 2007. Disponível em: <http://dddcommunity.org/learning-ddd/what_is_ddd/>. Acesso em: Abril de 2016.

FASSBENDER, S.; HEISEL, M.; MEIS, R. A Problem-, Quality-, and Aspect-Oriented Requirements Engineering Method. In: ____. **Software Technologies**. [S.l.]: Springer, 2015. p. 291 – 310.

FELLER, N. J. **Geração de testes de aceitação a partir de modelos U2TP para sistemas web**. 2015. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul. Disponível em: <http://hdl.handle.net/10183/115214>.

JACOBSON, I.; NG, P. **Aspect-oriented Software Development with Use cases**. [S.l.]: Addison-Wesley, 2004.

JBEHAVE. **JBehave**. 2015. Disponível em: <http://jbehave.org/>. Acesso em: 12/2015.

KICZALES, G. Aspect-Oriented Programming. **ACM Comput. Surv.**, v. 28, n. 4es, p. 154 –, 1997. Disponível em: <http://doi.acm.org/10.1145/242224.242420>.

KICZALES, G.; MEZINI, M. Aspect-Oriented Programming and Modular Reasoning. In: ANAIS DO ICSE'05, 2005. [S.l.], 2005. p. 49 – 58.

KUDRYASHOV, K.; STEAD, A.; NORTH, D. **Behaviour Driven Development**: The beginner's guide to BDD. 2015. Disponível em: <https://inviqa.com/bdd-guide>. Acesso em: Abril de 2016.

KUMAR, S. **A Measurement Framework for Aspect Oriented Software Testability**. 1. ed. [S.l.]: LAP LAMBERT, 2016.

LADDAD, R. **Aspectj in action**: enterprise AOP with spring applications. 2. ed. [S.l.]: Manning Publications Co, 2010.

MAJCHRZAK, T. A.; SIMON, A. Using spring Roo for the test-driven development of Web applications. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 2012. [S.l.], 2012.

MITCHELL, J. L.; BLACK, R. **Advanced Software Testing**. 2. ed. [S.l.]: Rocky Nook, 2015. v. 3. ISBN 978-1-937538-64-4.

MOREIRA, A.; CHITCHYAN, R.; ARAÚJO, J.; RASHID, A. **Aspect-oriented requirements engineering**. 1. ed. [S.l.]: Springer, 2013.

NAIK, K.; TRIPATHY, P. **Software testing and quality assurance** / **Theory and Pratice**. 1. ed. [S.l.]: Wiley, 2008.

NEGARA, N.; STROULIA, E. Automated Acceptance Testing of JavaScript Web Applications. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 2012. [S.l.], 2012. p. 318 – 322.

NORTH, D. **INTRODUCING BDD**. 2006. Disponível em: <https://dannorth.net/introducing-bdd/>. Acesso em: Abril de 2016.

NORTH, D. **BDDWiki**: Introduction. 2014. Editado pela última vez em 2014-09-29. Disponível em: <http://behaviourdriven.org/>. Acesso em: Abril de 2016.

OKOLNYCHYI, A.; FÖGEN, K. A Study of Tools for Behavior-Driven Development. In: FULL-SCALE SOFTWARE ENGINEERING - CURRENT TRENDS IN RELEASE ENGINEERING, 2016. 2016. Disponível em: <https://www2.swc.rwth-aachen.de/docs/teaching/seminar2016/FsSE_CTRelEng_2016.pdf>.

PRESSMAN, R. S. **Engenharia de Software**. 7. ed. [S.l.]: Bookman, 2011.

PUGH, K. **Lean-agile acceptance test-driven development**. [S.l.]: Pearson Education, 2010.

RAMOS, R.; ARAÚJO, J.; MOREIRA, A.; CASTRO, J.; ALENCAR, F.; PENTEADO, R. **Um Padrão para Requisitos Duplicados**. [S.l.]: Supporting Organizations, 2007.

RASHID, A.; MOREIRA, A.; ARAÚJO, J. Modularisation and composition of aspectual requirements. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 2003. 2003. Disponível em: <http://dblp.uni-trier.de/db/conf/aosd/aosd2003.html>.

RASHID, A.; SAWYER, P.; MOREIRA, A.; ARAUJO, J. Early aspects: A model for aspect-oriented requirements engineering. In: REQUIREMENTS ENGINEERING, 2002. PROCEEDINGS. IEEE JOINT INTERNATIONAL CONFERENCE ON, 2002. **IEEE**. [S.l.], 2002. p. 199 – 202.

RESTIVO, A. M. de O. **Incremental Modular Testing in Aspect Oriented Programing**. 2015. 212 p. Tese (Faculdade de Engenharia Informatica) — Faculdade de Engenharia da Universidade do Porto.

RUMPE, B. Model-based testing of object-oriented systems. In: ____. **Formal Methods for Components and Objects**. [S.l.]: Springer Berlin Heidelberg, 2003. p. 380 – 402.

SHESHASAAYEE, A.; JOSE, R. A Theoretical Framework for the Maintainability Model of Aspect Oriented Systems. **Procedia Computer Science**, v. 62, p. 505 – 512, Agosto 2015.

SILVA, L. de P. **A engenharia de requisitos orientada a aspectos**: a abordagem DAORE. 2007. 233 p. Dissertação (Programa de Pós- Graduação em Ciência da Computação) — Universidade Estadual de Maringá.

SMART, J. F. **BDD in Action**: Behavior-Driven Development for the whole software lifecycle. [S.l.]: Manning, 2015. ISBN 9781617291654.

SOLÍS, C.; WANG, X. A Study of the Characteristics of Behaviour Driven Development. **IEEE**, Oulu (Finlândia) - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011.

SOMMERVILLE, I. **Software Engineering - 9th edition**. [S.l.]: Pearson, 2011.

TEKINERDOğAN, B.; MOREIRA, A.; ARAÚJO, J.; CLEMENTS, P. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. In: ASPECTS, E. (Ed.). [S.l.], 2005.

# APÊNDICE - Especificações das histórias do usuário ATM

A seguir as especificações das histórias do exemplo demonstrativo Automated Teller Machine (ATM) com a abordagem aplicada.

```
Narrative:
In order to ligar uma máquina ATM
As a operador da máquina ATM
I want to inicializar o sistema da máquina adequadamente

Scenario: C01-Deve inicializar sistema na máquina ATM
Given o sistema desligado
When ativar a chave de ligar
Then sistema solicita montante inicial em dinheiro

Scenario: C02-Espera validar o montante inicial em dinheiro
Given sistema solicitando uma quantia em dinheiro
When inserido um valor legítimo
Then sistema liga

Scenario: C03-Deve testar primeira conexão com banco
Given sistema ligado com sucesso
When realizada uma transação teste válida
Then sistema deve informar conexão estabelecida
```

**Figura 38 – História Inicializar Sistema ATM**

Fonte: Autor

```
Narrative:
In order to desligar uma máquina ATM
As a operador da máquina ATM
I want to desligar o sistema da máquina adequadamente

Scenario: C04-Deve realizar o desligamento do sistema
Given sistema ligado e não atendendo um cliente
When ativar a chave desligar
Then o sistema desliga
And a conexão do banco com o ATM não deve mais existir

Scenario: C05-Espera encerrada a conexão com o Banco quando o sistema for desligado
Given o ativamento da chave desligar
When sistema desligar
Then a conexão do banco com o ATM não deve mais existir
```

**Figura 39 – História Desligar Sistema ATM**

Fonte: Autor

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ before Saque.*, Transferencia.*, Consulta.*, Deposito.* : C11 ]

Narrative:
In order to acessar minha conta bancária com cartão ATM
As a cliente
I want to ter acesso as operacões bancárias

Scenario: C06-O sistema deve solicitar a senha após leitura do cartão ATM de um cliente
Given sistema ligado e desocupado
When inserir um cartão legível
Then cartão é aceito
And o sistema solicita a entrada da senha

Scenario: C07-Espera-se que o sistema exiba o menu com as operações
Given o sistema está solicitando a entrada da senha
When inserir uma senha correta
Then o sistema exibe um menu com os tipos de operações

Scenario: C08-O sistema deve permitir ao cliente realizar uma operação
Given o sistema está exibindo o menu com os tipos de operações
When executar uma operação
Then o sistema pergunta se o cliente quer outra operação

Scenario: C09-O sistema deve permitir múltiplas operações em uma sessão
Given o sistema está perguntando se o cliente quer outra operação
When responder sim
Then o sistema exibe um menu com os tipos de operação

Scenario: C10-O sistema deve finalizar a sessão
Given o sistema está perguntando se o cliente quer outra operação
When responder não
Then O sistema ejeta o cartão
And está pronto para iniciar uma nova sessão

Scenario: C11-O sistema deve verificar sessão ativa para permitir nova transação
Given cliente iniciou uma sessão no ATM
When o tempo da sessão estiver ativo
Then a sessão está ativa e transações podem ser realizadas
```

**Figura 40 – HistóriaAspecto Sessão ATM**

Fonte: Autor

```
Narrative:
In order to da execução de uma transação
As a cliente
I want to conseguir realizar minhas transacões bancárias

Scenario: C12-Espera-se que o sistema não permita transação com PIN inválido
Given um cartão legível inserido
When um PIN incorreto for inserido e uma transação for solicitada
Then a transação não é permitida { AspectoExcecao: after C40 }

Scenario: C13-Espera-se que o sistema manipule corretamente um erro durante uma transação
Given um cartão legível inserido
When ocorrer um erro durante a transação
Then a transação deve ser encerrada { AspectoExcecao: after C48 }

Scenario: C14-Espera-se que o sistema não permita transação com cartão inlegível
Given um cartão inserido
When o cartão for inlegível e uma transação for solicitada
Then a transação não é permitida { AspectoExcecao: after C49 }
```

**Figura 41 – História Transação ATM**

Fonte: Autor

```
Narrative:
In order to realizar uma transação de saque
As a cliente
I want to realizar saque com sucesso

Scenario: C15-O sistema deve pedir ao cliente para escolher em qual de suas contas será o saque
Given o menu com tipos de transações{ AspectoSessao: before C11 }
When escolhida a transação saque
Then o sistema exibe um menu com os tipos de contas disponíveis

Scenario: C16-O sistema deve solicitar ao cliente os valores para o saque
Given o menu com os tipos de contas { AspectoSessao: before C11 }
When uma conta é escolhida
Then o sistema exibe um menu de possíveis quantidades de retirada

Scenario: C17-O sistema deve verificar se ATM possui dinheiro suficiente para saque
Given o sistema está solicitando uma quantia para saque{ AspectoSessao: before C11 }
When o valor informado é maior que o ATM disponibiliza para saque
Then o sistema pede ao cliente para escolher uma quantidade diferente { AspectoInformativoTransacao: after C54 }

Scenario: C18-Espera-se que o sistema execute uma transação de saque legítima corretamente
Given o menu de valores para saque{ AspectoSessao: before C11 }
When escolhido um valor do menu
And o valor não seja maior que saldo da conta
Then o sistema permite o saque { AspectoLog: after C50; AspectoRecibo: after C51}

Scenario: C19-O sistema deve verificar saldo da conta para permitir saque
Given o sistema está solicitando uma quantia de retirada{ AspectoSessao: before C11 }
When escolhido um valor disponível no ATM, mas que é maior do que o saldo da conta
Then o sistema oferece ao cliente a opção de fazer outra transação{ AspectoInformativoTransacao: after C55 }

Scenario: C20-Espera-se que um saque possa ser cancelado pelo cliente antes da escolha da conta
Given o menu com os tipos de contas para saque { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação{ AspectoInformativoTransacao: after C52 }

Scenario: C21-Espera-se que saque possa ser cancelado pelo cliente antes de escolha do valor
Given o menu de valores para saque { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }
```

**Figura 42 – História Saque ATM**

Fonte: Autor

```
Narrative:
In order to realizar uma transação de transferência entre contas
As a cliente
I want to realizar uma transferência com sucesso

Scenario: C22-O sistema deve solicitar ao cliente que escolha a conta de retirada para transferência
Given o menu de tipos de transação{ AspectoSessao: before C11 }
When escolhida a transação transferência
Then o sistema exibe o menu para escolha da conta de retirada para transferência

Scenario: C23-O sistema deve exibir menu para escolha da conta que receberá a transferência
Given o menu para escolha da conta de retirada para transferência { AspectoSessao: before C11 }
When a conta de retirada para transferência for escolhida
Then o sistema exibe o menu para escolha da conta que receberá a transferência

Scenario: C24-O sistema deve solicitar ao cliente o valor para transferência
Given o menu para escolha da conta que receberá a transferência { AspectoSessao: before C11 }
When a conta que receberá a transferência for escolhida
Then O sistema solicita o valor para transferência

Scenario: C25-Espera-se que o sistema execute uma transação de transferência legítima corretamente
Given o sistema solicite o valor para transferência{ AspectoSessao: before C11 }
When um valor legítimo for inserido
Then O sistema permite a transferência { AspectoLog: after C50; AspectoRecibo: after C51}

Scenario: C26-Espera-se que o sistema cancele transferência no momento da escolha da conta de retirada para transferência
Given o menu para escolha da conta de retirada para transferência{ AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação{ AspectoInformativoTransacao: after C52 }

Scenario: C27-Espera-se que o sistema cancele transferência no momento da escolha da conta que receberá transferência
Given o menu menu para escolha da conta que receberá a transferência{ AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }

Scenario: C28-Espera-se que o sistema cancele transferência antes da inserção do valor que será transferido
Given o sistema solicite o valor para transferência{ AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }
```

**Figura 43 – História Transferência ATM**

Fonte: Autor

```
Narrative:
In order to realizar uma transação de consulta a minhas contas
As a cliente
I want to consultar minhas contas com sucesso

Scenario: C29-O sistema deve solicitar ao cliente a escolha de uma conta para consulta
Given a exibição do menu de tipos de transação { AspectoSessao: before C11 }
When escolhida a transação consulta
Then o sistema exibe o menu com os tipos de conta para consulta

Scenario: C230-Espera-se que o sistema execute uma transação de consulta legítima corretamente
Given a exibição do menu com os tipos de conta para consulta { AspectoSessao: before C11 }
When uma conta é escolhida
Then o consulta é realizada com sucesso { AspectoLog: after C50; AspectoRecibo: after C51 }

Scenario: C31-Espera-se que o sistema cancele consulta antes de escolher uma conta
Given a exibição do menu com os tipos de conta { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o cancelamento é realizado com sucesso { AspectoInformativoTransacao: after C52 }
And o sistema oferece ao cliente a opção de fazer outra transação
```

**Figura 44 – História Consulta ATM**
Fonte: Autor

```
Narrative:
In order to realizar uma transação depósito em uma conta
As a cliente
I want to realizar o depósito com sucesso

Scenario: C32-O sistema deve solicitar ao cliente que escolha em qual de suas contas será o depósito
Given o menu de tipos de transação { AspectoSessao: before C11 }
When a transação depósito for escolhida
Then O sistema exibe um menu com os tipos de conta

Scenario: C33-O sistema deve solicitar ao cliente que informe o valor para depósito
Given o menu com os tipos de conta { AspectoSessao: before C11 }
When uma conta for escolhida
Then o sistema solicita ao cliente o valor para depósito

Scenario: C34-O sistema deve solicitar ao cliente que insira um envelope
Given o sistema solicita para o cliente digitar um valor para depósito { AspectoSessao: before C11 }
When inserido um valor legítimo
Then o sistema solicita que o cliente insira um envelope

Scenario: C35-Espera-se que o sistema execute uma transação de depósito legítima corretamente
Given o sistema está solicitando que o cliente insira um envelope { AspectoSessao: before C11 }
When inserido um evelope
Then o sistema permite o depósito { AspectoLog: after C50; AspectoRecibo: after C51}
```

**Figura 45 – Primeira parte História Depósito ATM**
Fonte: Autor

```
Scenario: C36-Espera-se que o sistema cancele o depósito no momento da escolha da conta
Given o menu de tipos de contas { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }

Scenario: C37-Espera-se que o sistema cancele o depósito no momento da escolha do valor para depósito
Given o sistema está solicitando que o cliente digite um valor para depósito { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }

Scenario: C38-Espera-se que o sistema cancele o depósito antes de inserir o envelope
Given o sistema está solicitando que o cliente insira um envelope { AspectoSessao: before C11 }
When escolher a opção cancelar
Then o sistema oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }

Scenario: C39-Espera-se que o sistema cancele depósito automaticamente
Given o sistema está solicitando que o cliente insira um envelope { AspectoSessao: before C11 }
When atingir o tempo limite de espera para inserir o envolepe
Then o sistema cancela o depósito
And oferece ao cliente a opção de fazer outra transação { AspectoInformativoTransacao: after C52 }
```

**Figura 46 – Segunda parte História Depósito ATM**
Fonte: Autor

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ after Transacao.C12 : C40;
  after Transacao.C13 : C48;
  after Transacao.C14 : C49;
]
Narrative:
In order to representar as exceções ocorridas
As a cliente
I want to usar ATM sem problemas

Scenario: C40-Espera-se que sistema permita reinserção do PIN
Given cliente inserir PIN incorreto
When tentar realizar uma transação
Then a transação não é permitida
And solicitado reinserção do PIN

Scenario: C41-Espera-se que sistema permita transação após reinserção de PIN correto
Given solicitação do sistema para o cliente reinserir PIN
When inserido o PIN correto
Then a transação é concluída com sucesso

Scenario: C42-Espera-se que sistema permita a execução de nova transação após reinserção de PIN correto
Given reinserção correta do PIN pelo cliente
When cliente executar outra transação
Then a nova transação é concluída com sucesso

Scenario: C43-Espera-se que o sistema permita uma segunda reinserção de PIN
Given solicitação do sistema para o cliente reinserir PIN
When inserido o PIN incorreto
Then uma mensagem adequada é exibida
And o sistema solicita reinserção do PIN pela segunda vez

Scenario: C44-Espera-se que sistema permita transação após a segunda reinserção de PIN correto
Given o sistema solicita reinserção do PIN pela segunda vez
When inserido o PIN correto
Then a transação é concluída com sucesso
```

**Figura 47 – Primeria parte HistóriaAspecto Exceção ATM**

Fonte: Autor

```
Narrative:
In order to representar as exceções ocorridas
As a cliente
I want to usar ATM sem problemas

Scenario: C45-Espera-se que o sistema permita uma terceira reinserção de PIN
Given o sistema solicita reinserção do PIN pela segunda vez
When inserido o PIN incorreto
Then uma mensagem adequada é exibida
And o sistema solicita reinserção do PIN pela última vez

Scenario: C46-Espera-se que sistema permita transação após a terceira reinserção de PIN correto
Given o sistema solicita reinserção do PIN pela terceira vez
When inserido o PIN correto
Then a transação é concluída com sucesso

Scenario: C47-O sistema deve reter cartão do cliente caso ocorra 3 reinserções incorretas do PIN
Given o sistema solicita reinserção do PIN pela terceira vez
When inserido o PIN incorreto
Then uma mensagem adequada é exibida, cartão é retido pela máquina e a sessão termina

Scenario: C48-O sistema deve tratar falha na transação que não seja por senha inválida
Given a execução de uma transação
When ocorrer uma falha qualquer que não senha inválida
Then uma explicacao é exibida
And o sistema perguta ao cliente se deseja outra transação

Scenario: C49-Espera-se que o sistema recuse um cartão ilegível
Given sistema ligado e desocupado
When insiro um cartão ilegível
Then cartão é rejeitado,uma tela de erro é exibida e pode ser iniciada nova sessão
```

**Figura 48 – Segunda parte HistóriaAspecto Exceção ATM**
Fonte: Autor

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ after Saque.C18, Transferencia.C25, Consulta.C30, Deposito.C35 : C50 ]

Narrative:
In order to armazenar informações sobre a execução das transações dos clientes
As a gerente do banco
I want to que seja gerado um log das transações executadas pelos clientes

Scenario: C50-Espera-se que o sistema gere o log das transações
Given o cliente realize uma transação
When o sistema concluir a execução da transação
Then o sistema deve gerar log para transação
```

**Figura 49 – HistóriaAspecto Log ATM**
Fonte: Autor

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ after Saque.C18, Transferencia.C25, Consulta.C30, Deposito.C35 : C51 ]

Narrative:
In order to receber um comprovante das transações executadas
As a cliente
I want to após a execução com sucesso das transações receber um recibo

Scenario: C51-Espera-se que o sistema forneça recibo das transações
Given o cliente realize uma transação com sucesso
When o sistema concluir a execução da transação
Then o sistema deve gerar o recibo da transação
```

**Figura 50 – HistóriaAspecto Emissão Recibo ATM**
Fonte: Autor

```
!--DECLARAÇÃO DE COMPOSIÇÃO DO ASPECTO
[ after Saque.{20,21}, Transferencia.{C26,C27,28}, Consulta.C31, Deposito.{C36,37,38,39}: C52;
  after Saque.C17 : C54;
  after Saque.C19 : C55;
 ]

Narrative:
In order to receber informações sobre a execução das transações
As a cliente
I want to ser informado do sucesso ou insucesso da execução das transações

Scenario: C52-O sistema deve informa ao cliente o cancelamento de uma transação
Given a execução de uma transação
When o cliente solicitar o cancelamento da transação
Then o sistema deve informa na tela que a transação foi cancelada

Scenario: C53-O sistema deve informa ao cliente o sucesso de uma transação
Given o cliente realize uma transação
When o sistema concluir a execução da transação com sucesso
Then o sistema deve informa na tela que a transação foi realizada com sucesso

Scenario: C54-O sistema deve informa ao cliente a impossibilidade do saque no ATM
Given o cliente informa o valor para saque
When o valor informado for maior que o ATM disponibiliza para saque
Then o sistema deve informa na tela que não possui dinheiro suficiente para saque
And informar o valor máximo disponível para saque

Scenario: C55-O sistema deve informa ao cliente a impossibilidade do saque por saldo insuficiente
Given o cliente informa o valor para saque
When escolhido um valor disponível no ATM, mas que é maior do que o saldo da conta
Then o sistema deve informa na tela que o cliente não possui saldo suficiente para saque
And informar o valor máximo disponível para saque
```

**Figura 51 – HistóriaAspecto Informativo Transação ATM**

Fonte: Autor