# UNIVERSIDADE FEDERAL DE ALFENAS

## INSTITUTO DE CIÊNCIAS EXATAS

**Bacharelado em Ciência da Computação**

**Arthur Henrique Nagy Ghideli**


# Customizacão do OpenUP/Basic para o levantamento de requisitos com histórias de usuário


Alfenas, 27 de Março de 2013.

# UNIVERSIDADE FEDERAL DE ALFENAS

## INSTITUTO DE CIÊNCIAS EXATAS

**Bacharelado em Ciência da Computação**


# Customizacão do OpenUP/Basic para o levantamento de requisitos com histórias de usuário


***Arthur Henrique Nagy Ghideli***


Monografia apresentada ao curso de Ciência da Computação, da Universidade Federal de Alfenas, para a obtenção do grau de Bacharel em Ciência da Computação, sob orientação do Prof.º Rodrigo Martins Pagliares.

Alfenas, 27 de Março de 2013.

*Arthur Henrique Nagy Ghideli*

# Customizacão do OpenUP/Basic para o levantamento de requisitos com histórias de usuário

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

---

Prof. Humberto César Brandão

Universidade Federal de Alfenas

---

Prof. Douglas Donizeti De Castilho Braz

Universidade Federal de Alfenas

---

Prof. Rodrigo Martins Pagliares

Universidade Federal de Alfenas

Alfenas, 27 Março de 2013.

# RESUMO

Em um mundo altamente competitivo, a agilidade, eficiência e qualidade, no processo de desenvolvimento de software, tornam-se um grande diferencial. Técnicas que tornam o processo de produção de software mais ágil assumem um papel cada vez mais significativo, levando os fabricantes de software a buscarem por alternativas cada vez mais eficientes. Este trabalho apresenta o estudo sobre o EPF-Composer, histórias de usuário e o desenvolvimento da disciplina de requisitos focado em histórias do usuário utilizando o EPF-Composer. Por fim será apresentada a criação de um padrão de recurso para a disciplina de requisitos modelada.

**Palavras-chave:** técnicas ágeis, processo de desenvolvimento de software, histórias de usuário, EPF-Composer, padrão de recurso.

# Abstract

In a highly competitive world, the agility, efficiency and quality in software development process, become a major differentiator. Techniques that make the production process of software more agile play an increasingly significant role, leading software manufacturers to seek alternatives increasingly efficient. This paper presents a study on the EPF-Composer, user stories and the development of the discipline of requirements focused on user stories using the EPF-Composer. Finally, it will be seen the creation of a capability pattern for the requirements discipline modeled.



**Keywords:** agile techniques, software development process, user stories, EPF Composer, capability pattern.

# LISTA DE FIGURAS

Figura 1.0 - Distribuição de Sucesso e Fracassos em Projetos de Software....12
Figura 2.0 – Exemplo de uma história de usuário para um caso fictício....15
Figura 2.1 – Exemplo de testes de aceitação em um cartão de história....22
Figura 3.0 – O framework EPF-Composer....25
Figura 3.1 – Funcionamento simples de uma tarefa segundo o SPEM....27
Figura 3.2 – Organização do framework....28
Figura 3.3 – Página web resultante da publicação efetuada pelo EPF-Composer....29
Figura 4.0 – Fluxo de trabalho da disciplina de requisitos com histórias de usuário....31
Figura 4.1 – OpenUP/Basic, sem customizações no EPF-Composer....32
Figura 4.2 – Criação do método do plug-in “Histórias de usuário” e do método “Requisitos-US”....33
Figura 4.3 – Diagrama de atividades para a disciplina de requisitos....34
Figura 4.4 – Tarefas do método “Requisitos_US”....35
Figura 4.5 – Criando um padrão de recurso para as histórias de usuário....36
Figura 4.6 – Suprimindo a padrão de recurso corrente no ciclo de vida....37
Figura 4.7 – Criando uma atividade filha na iteração de fase....37
Figura 4.8 – Passo-a-passo para a adição do padrão de recurso de histórias de usuário....38
Figura 4.9 – Resultado esperado após a customização....38
Figura 4.10 – Adicionando uma perspectiva de visualização....39
Figura 4.11 – Criando a configuração de método para a publicação....40

# LISTA DE TABELAS

Tabela 1.0 - Pontos Críticos para o Desenvolvimento de Software.....12

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| XP | Extreme Programming |
| EPF-Composer | Eclipse Process Framework Composer |
| SME | Engenharia de métodos situacional |
| OpenUP/Basic | Open Unified Process |
| SCRUM | Processo iterativo para gerenciamento de projetos |
| CASE | Computer-Aided Software Engineering |
| CAME | Computer-Aided Method Engineering |

# SUMÁRIO


1 INTRODUÇÃO ..... 10

1.1 JUSTIFICATIVA ..... 11

1.2 PROBLEMATIZAÇÃO ..... 13

1.3 OBJETIVOS ..... 13

1.3.1 Gerais ..... 13

1.3.2 Específicos ..... 14

1.4 METODOLOGIA ..... 14

2 HISTÓRIAS DE USUÁRIO ..... 15

2.1 VISÃO GERAL ..... 15

2.2 ESCRITA DE HISTÓRIAS ..... 16

2.2.1 Independente ..... 17

2.2.2 Negociavel ..... 17

2.2.3 Valor para o cliente ..... 17

2.2.4 Estimável ..... 17

2.2.5 Tamanho ideal ..... 18

2.2.6 Testavel ..... 18

2.3MODELAR PAPÉIS DE USUÁRIO ..... 18

2.4 LEVANTAMENTO DE HISTÓRIAS ..... 19

2.5 SUBSTITUTOS DE USUÁRIOS ..... 20

2.6 TESTES DE ACEITAÇÃO ..... 21

2.7 GUIA PARA BOAS HISTÓRIAS ..... 22

2.8 CASOS DE USO VS. HISTÓRIAS DE USUÁRIOS ..... 24

3 CONTEXTUALIZAÇÃO ..... 25

3.1 MÉTODOS ..... 26

3.2 ECLIPSE PROCESS FRAMEWORK COMPOSER ..... 26

4 CUSTOMIZAÇÃO DO OPENUP/BASIC ..... 30

5 CONCLUSÕES E TRABALHOS FUTUROS ..... 41

3.1 MÉTODOS ..... 41

3.1 MÉTODOS ..... 42

REFERÊNCIAS ..... 43

# 1

# Introdução

*Este capítulo apresenta alguns detalhes pela razão o qual este trabalho foi desenvolvido, bem como seus objetivos, justificativa e metodologia para a realização deste.*

Nos dias de hoje, devido à evolução crescente dos programas de computadores, maneiras cada vez mais eficazes são trazidas pela Engenharia de Software, a fim de levantar cada vez mais informações sobre características a serem implementadas nos software. Verifica-se ainda a implantação de métodos cada vez mais eficientes que se encaixam nas tendências do mercado para dinamizar e facilitar a integração das equipes e a comunicação entre os compradores e os desenvolvedores.

Existem vários modelos de ciclo de vida de processo na Engenharia de Software. Podem ser citados como exemplos os modelos em cascata e espiral. Tais modelos são cada vez menos utilizados, pois a aplicabilidade destes métodos pode ser comprometida quando os requisitos estão emergindo e mudando rapidamente (COHEN et al., 2004). Sendo assim, levando em consideração o mercado de software atual, as empresas desenvolvedoras de software tendem a não utilizar esses métodos, de modo que o primeiro não se aplica à realidade do desenvolvimento por não considerar mudanças de requisitos e o segundo por ser utilizado principalmente para prototipação (SOMMERVILLE, 2003). Além disso, modelos baseados em cascata em geral não promovem uma boa comunicação entre os próprios membros da equipe, não existindo interação com o cliente durante o desenvolvimento. Com isso, corre-se o risco do software entregue não atender às reais necessidades dos usuários, uma vez que os requisitos já levantados podem ter sofrido mudanças ou adaptações.

Fica claro que esses modelos já não se encaixam mais nos ambientes de desenvolvimento atuais, pois cada vez mais as empresas vêm buscando aumentar a comunicação entre as equipes e a comunicação das equipes com os clientes, a fim de alcançar melhores resultados com esse relacionamento.

Neste sentido, surgiram abordagens menos tradicionais, que possuem premissas, dentre outras, “indivíduos e interação entre eles mais que processos e ferramentas” (Manifesto Ágil, 2001). Tais abordagens são conhecidas como métodos ágeis. Podem-se citar como exemplos neste contexto o conjunto de práticas definidas pelo XP (*Extreme programming)*, o modelo de processo ágil OpenUP/Basic, o método para planejamento e acompanhamento de projetos SCRUM, dentre outros.

Uma das maneiras tradicionais de se especificar requisitos é através do modelo de casos de uso, que é uma documentação detalhada dividida em cenários e fluxos, em que é feita toda a descrição de um requisito. Apesar de casos de uso serem uma técnica bastante difundida, ela prescreve um certo nível de detalhamento dos requisitos, gerando em muitos casos, documentação abrangente. Por outro lado, um dos princípios do manifesto ágil (Agile Manifesto, 2001) é de que software em funcionamento é mais importante que documentação abrangente, mostrando a necessidade de criação ou adaptação nas formas tradicionais de especificação de requisitos.

Neste sentido, este trabalho tem como objetivo estudar uma maneira alternativa de se especificar os requisitos de software, priorizando a comunicação entre a equipe e o cliente. Esta técnica é chamada de histórias de usuário, e procura facilitar a comunicação através do uso de uma linguagem em nível de negócio, se livrando de jargões técnicos, utilizando maneiras de representar os requisitos do cliente de maneira eficiente ao invés de ficar atrelado a uma documentação extensa.

## 1.1. Justificativa

Requisitos de software são especificações de serviços que um sistema deve fornecer, bem como restrições e conhecimentos necessários para produzi-lo (KOTONIA; SOMMERVILLE, 1998). Sendo assim, os requisitos tem que ser modelados e acompanhados ao longo de todo o processo de desenvolvimento de software, sendo que, um ponto principal é entender as características destes requisitos e a relação entre eles.

Requisitos de software são um problema de comunicação, pois aqueles que querem um software precisam se comunicar com aqueles que produzem e isto significa interagir com diferentes formas de pensar, trazendo perspectivas diferentes para a produção deste (COHN et al., 2004).

Uma pesquisa do *Standish Group* em 2009, representada pela Figura 1.0, revelou que dentre 8380 projetos de um grupo de 365 empresas, que a maioria, cerca de 44%, eram problemáticos, ou seja, não cobriam todas as funcionalidades, custo e prazos. Uma segunda parcela mostra que cerca de 32% dos projetos apresentaram sucesso em termos de prazo, custos e funcionalidades. Por fim, cerca de 24% dos projetos foram cancelados durante a sua fase de desenvolvimento.

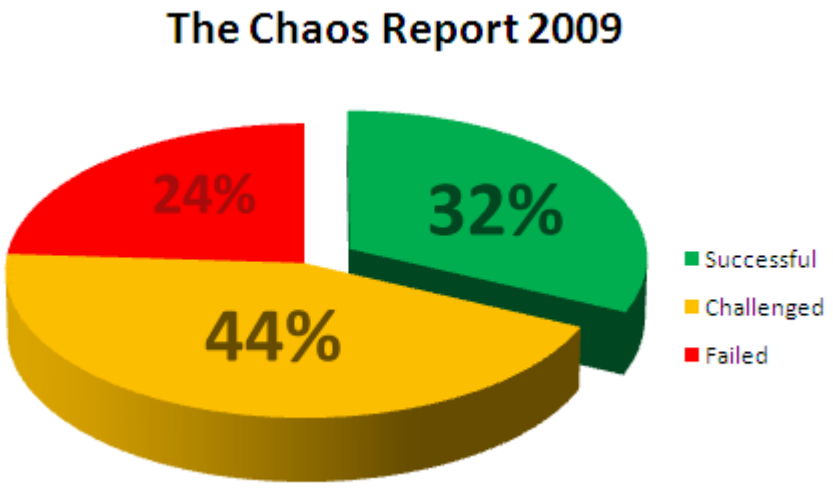


**Figura 1.0 -** Distribuição de sucesso e fracassos em projetos de software**. Fonte:** (The Standish Group, 2009).

Ainda na mesma pesquisa, o *Standish Group* revelou os fatores mais importantes para o desenvolvimento de softwares que podem ser vistos na Tabela 1.0. Dois dos fatores críticos são os "Requisitos Incompletos" e "Falta de Envolvimento do Usuário", mostrando a relevância dos requisitos e da comunicação no processo de desenvolvimento de software.

**Tabela 1.0 -** Pontos críticos para o desenvolvimento de software. **Fonte:** (The Standish Group, 2009).

| Fatores Críticos | % de Respostas |
|---|---|
| 1. Requisitos Incompletos | 13.1% |
| 2. Falta de Envolvimento do Usuário | 12.4% |
| 3. Falta de Recursos | 10.6% |
| 4. Expectativas Irreais | 9.9% |
| 5. Falta de Suporte Executivo | 9.3% |
| 6. Mudança de Requisitos & Especificações | 8.7% |
| 7. Falta de Planejamento | 8.1% |
| 8. Não Precisa por Mais Tempo | 7.5% |

Sendo assim, neste texto, a técnica de levantamento de requisitos ágeis conhecida como histórias de usuário será apresentada e estudada junto ao *Eclipse Processes Framework* (EPF-Composer). Posteriormente será criado, com o EPF-Composer, um padrão de recurso da

disciplina de requisitos focada em histórias de usuário que irá substituir a disciplina de requisitos padrão do OpenUp/Basic.

## 1.2. Problematização

Levando em conta que as empresas que se preocupam em fazer um bom uso das práticas de engenharia de software tendem a acompanhar a evolução das técnicas implementadas pela mesma. Muitas dessas empresas tendem a seguir métodos ágeis, acabando por deixar de utilizar conceitos antigos, que não se encaixam nos perfis de mercado atuais. Sendo assim, a modelagem dos métodos ágeis pode necessitar de uma ferramenta para implementar estes novos métodos.

Apesar de casos de uso ser uma forma muito difundida entre várias instituições, a busca por técnicas mais recentes é de fundamental importância para as instituições que desejam se manter em alta e com competividade no mercado. Neste trabalho será proposta e modelada uma técnica alternativa e que está dentro dos princípios do manifesto ágil. Esta técnica busca resolver antigos problemas do grupo perante a documentação até então utilizada, melhorando a forma como os requisitos são descritos e implementados.

## 1.3. Objetivos

Com a evolução da computação e dos sistemas, a engenharia de software busca cada vez mais buscar melhores resultados com o levantamento de requisitos. A técnica de histórias de usuário, sendo uma maneira alternativa à especificação tradicional por casos de uso, provê maneiras centradas em comunicação e documentação informal que facilitam e podem trazer, por fim, melhores resultados no levantamento de requisitos.

### 1.3.1. Gerais

Logo, o objetivo geral deste trabalho é modelar as histórias de usuário, utilizando o EPF-Composer e integra-la no OpenUP/Basic, efetuando assim a customização da disciplina de requisitos padrão utilizada, que é focada em casos de uso.

### 1.3.2. Específicos

Para que o objetivo geral do projeto possa ser alcançado, uma lista de objetivos específicos precisa ser atendida:

- Fazer o estudo sobre as histórias de usuário*;*
- Fazer o estudo do EPF-Composer;
- Criar um padrão de recurso para a disciplina de requisitos focada em histórias de usuário.
- Customizar o OpenUP/Basic, modelando as histórias de usuário com o EPF-Composer .
- Escrever de um artigo para a revista de engenharia de software com resultados do projeto.

## 1.4. Metodologia

A metodologia, segundo Ferreira (2002), é "o caminho pelo qual se atinge um objetivo". Assim, para que os objetivos sejam alcançados a metodologia que será utilizada neste trabalho será analítica, estudando a ferramenta (EPF-Composer) e o método (histórias de usuário) e assim construir a disciplina de requisitos focada em histórias de usuário.

De início será feito um estudo aprofundado sobre *as* histórias de usuário e o EPF-Composer, a fim de dominar os conceitos principais de cada um. Posteriormente a disciplina de requisitos será modelado no OpenUP/Basic, utilizando o *framework.*

A organização do trabalho apresenta primeiramente as histórias de usuário no Capítulo 2 e a ferramenta no Capítulo 3, a fim de introduzir os conceitos básicos necessários para a modelagem da disciplina de requisitos que será feita no Capítulo 4. Por fim, o Capítulo 5 apresentará as conclusões finais e trabalhos futuros.

# 2
# Histórias de Usuário

*Este capítulo apresenta uma revisão dos conceitos necessários sobre histórias de usuário, para um bom entendimento deste trabalho.*

## 2.1. Visão geral

Histórias de usuário são técnicas ágeis que descrevem funcionalidades de um sistema de software a ser desenvolvido. A técnica é dita ágil, pois pode ser facilmente adaptada para os métodos ágeis como o XP e o OpenUP, e também enfatiza a comunicação verbal ao invés da documentação (COHN, 2004). Sendo assim, as histórias de usuário buscam fornecer somente informações essenciais e indispensáveis para que os envolvidos no processo de desenvolvimento compreendam tudo que se passa ao longo do projeto, para gerar uma visão compartilhada do negócio, de forma simples, sem muitos detalhes facilitando a comunicação entre todos os envolvidos no projeto (SOARES; FERREIRA, 2011).

Inicialmente o usuário escreve uma história em um cartão de papel, que serve como lembrete para o mesmo, depois a historia é discutida para determinar alguns detalhes sobre ela, e no verso são escritos testes, chamados testes de aceitação, que servem para determinar se a história está completa ou não. Os testes devem ser escritos o mais cedo possível em uma iteração, pois mais expectativas e suposições da equipe do cliente são discutidas entre os desenvolvedores (COHN, 2004). A Figura 2.0, a seguir, ilustra um exemplo de um possível cenário para uma história.

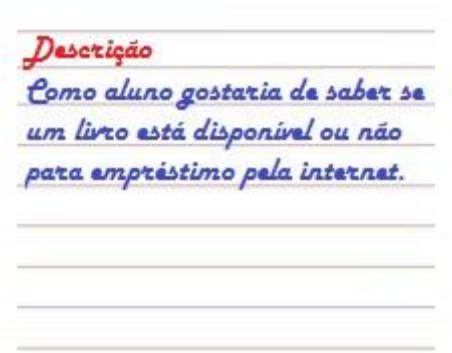


**Figura 2.0 –** Exemplo de uma história de usuário para um caso fictício.

O tamanho de uma história é a quantidade de funcionalidades que ela produz, e em casos de uma história ser muito grande ela poderá ser quebrada em histórias menores e mais detalhadas. As histórias não precisam ser documentadas, pois ao invés de documentá-las, é melhor que ela seja discutida entre a equipe do desenvolvedor e a equipe do cliente, isso a torna compreensível para ambas (DAVIES, 2001).

A equipe do cliente contém as pessoas que irão garantir que o software atenderá as expectativas dos usuários, sendo assim esta equipe possui testadores, gerente de produto, usuários reais e projetistas de interação (COHN, 2004).

Na técnica de histórias de usuário, os usuários estarão envolvidos por toda a duração do projeto, sendo assim os clientes e os usuários deveram ter uma função ativa no processo de escrita de histórias, pois as histórias deveram ser escritas em linguagem de negócio e a equipe do cliente é quem possui o papel de determinar o comportamento do *software* (COHN, 2004).

Outra importante característica das histórias de usuário é a entrega, que é composta por uma ou mais iterações. Para cada iteração é selecionado, dentre as historias, um conjunto menor de histórias criando um plano de iteração, ou seja, o plano de iteração é o que os programadores irão produzir em um período de tempo pré-estabelecido (20 dias, por exemplo). Existe também o plano de entrega que é uma linha de tempo que mostra a situação do projeto em relação às funcionalidades implementadas do projeto e as funcionalidades que o sistema deve ter ao fim do projeto.

As histórias são priorizadas pelo cliente, pois eles são as pessoas que possuem interesse direto no produto. No planejamento de uma entrega (iteração) serão selecionadas as histórias priorizadas que representam as funcionalidades que iram abranger uma grande quantidade de usuários, e também as historias que são designadas para um pequeno grupo de usuários e clientes importantes (DAVIES, 2001).

## 2.2. A escrita de histórias

Segundo Mike Cohn (2004), para a criação de uma boa história, são levados em conta seis atributos: independência, negociação, valor para o cliente, estimativa (tempo para a história se transformar em código em funcionamento), tamanho ideal para a história e testes. Essas seis características formam o acrônimo INVEST (*Independent, Negotiable, Valuable to users or customer, Estimable, Small, Testable).*

### 2.2.1. Independente

A independência entre as histórias é essencial para que a estimativa seja mais fácil e não ocorram problemas na hora de priorizar as histórias.

### 2.2.2. Negociável

As histórias são negociáveis, pois são escritas em cartões pequenos que contém uma descrição breve sobre o que devem implementar, e não são escritas como em contratos ou requisitos que o software deve implementar.

### 2.2.3. Valor para o cliente

As histórias devem ser escritas pelos clientes, pois são os maiores interessados no produto, além de serem os indivíduos que vão utilizar o sistema. Sendo a ordem de desenvolvimento das histórias serão escolhidas pelo clientes.

### 2.2.4. Estimável

As histórias geralmente são estimadas pelos desenvolvedores, mas em alguns casos particulares como falta de conhecimento do domínio, falta de conhecimento técnico acaba por dificultar na estimativa de uma história. O tamanho de uma história também influencia na estimativa, pois histórias de tamanho elevado são mais difíceis de serem compreendidas pelos desenvolvedores. A solução para obtenção de boas estimativas a partir histórias grandes, é quebra-las em histórias menores, facilitando assim à estimativa.

### 2.2.5. Tamanho Ideal

Deve-se ter atenção a escrever as histórias, pois elas precisam ser pequenas a ponto de poderem ser planejadas, sendo que uma vez que as histórias são grandes (épicos), elas podem contém múltiplas histórias. Em contra partida deve-se ter atenção com as histórias que são muito pequenas, pois elas não devem ser escritas, já que levariam mais tempo para serem

escritas do que desenvolvidas. Assim os épicos devem ser dividos em histórias menores para poderem ser planejadas, e as histórias muito pequenas devem ser combinadas com outras histórias de tamanho comum.

Existem também as histórias que devem ser estudadas pelos desenvolvedores para serem implementadas posteriormente. Essas histórias devem ser colocadas em iterações diferentes, e as histórias relacionadas a ela em iterações posteriores. Isso fará com que o nível de incerteza sobre a história não leve ao erro de planejamento de uma iteração, evitando que algumas histórias não sejam implementadas.

### 2.2.6. Testável

As histórias devem ser testadas para determinar se elas foram desenvolvidas corretamente, sempre que possível, pois como o produto é desenvolvido de forma incremental, trechos funcionais podem deixar de funcionar em um incremento.

## 2.3. Modelar os papéis de usuários

Na maioria dos casos as histórias são escritas somente por um ponto de vista, isso pode ocasionar a falta de histórias para usuários que não se encaixam na parte geral do sistema. Por esse motivo as histórias de usuário possuem o papel do usuário, que trata as histórias por várias perspectivas, a fim de atender o maior numero de usuários possíveis (DAVIES, 2001).

Segundo Ferreira e Soares (2010), utilizar papéis de usuário ajuda a melhorar a descrição de uma história levando a história a ser desenvolvida com mais qualidade levando em conta as necessidades do perfil do usuário que irá utilizá-la.

Para identificar os possíveis papéis existem quatro técnicas que são usadas em sequência, sendo *brainstorm* de um conjunto inicial de papéis de usuário a primeira delas, reunindo o maior numero de desenvolvedores e clientes possíveis. No *brainstorm*, cada participante pega um cartão, escreve o nome do papel no cartão, deixa à vista de todos e realiza esses passos novamente, escrevendo o maior numero de papéis possíveis (COHN, 2004).

Uma vez reunido a amostra inicial, os cartões são movidos para uma mesa ou quadro de forma que a posição destes cartões mostra a relação entre eles. A medida que a relação

entre os papéis aumenta, a sobreposição dos cartões aumenta também, sendo que quanto maior a similaridade entre duas histórias, maior será a sobreposição de um cartão sobre o outro. Esta sequência é chamada de organização inicial do conjunto (COHN, 2004).

Depois que os cartões foram agrupados, eles devem ser consolidados, levando em consideração o quão sobreposto os cartões estão. Cartões muito sobrepostos possuem características tão semelhantes que podem se tornar um só, somente adicionando a pequena diferença que fazia com que fossem diferentes. Alguns papéis são simplesmente separados, pois possuem características únicas. Após o termino da consolidação os cartões não estarão sobrepostos e serão entendidos com clareza, assim eles poderão ser refinados definindo atributos para cada um (COHN, 2004).

Uma técnica adicional para os papéis são os personas, que é de fato, uma representação imaginária de um papel, personificando estes papéis com fotos e contos, como se fossem pessoas reais, simplesmente para aumentar o relacionamento do papel com a equipe. Utilizar personas facilita a percepção de detalhes que não seriam percebidos se a história fosse atribuída pelo cliente (SOARES; FERREIRA, 2010).

Outra técnica é o uso de personagens extremos, que são casos de usuários finais que podem levar à falha de uma história por não ser considerado aquele uso peculiar deste usuário. Considerar esses personagens extremos pode aumentar a satisfação do usuário com o sistema (COHN, 2004).

## 2.4. O levantamento de histórias

Na coleta de histórias, as primeiras iterações irão possuir histórias de grande importância, deixando passar as histórias (requisitos) menos importantes, mas à medida que as iterações forem aumentando as histórias que ficaram para trás serão selecionadas. Assim, à medida que o projeto avança, ele é cada vez mais refinado, possuindo requisitos cada vez mais direcionados aos pequenos detalhes (COHN, 2004). Assim, certos requisitos que não foram implementados em dado ponto do projeto, não foram simplesmente por não serem tão importantes quanto os outros.

Para Mike Cohn (2004), quatro técnicas tradicionais para levantamento de requisitos podem ser utilizadas também para levantamento de histórias, são elas, entrevistas com usuários, questionários, observação e os workshops de escrita de histórias.

- As entrevistas são perguntas bem direcionadas aos usuários, para levantar histórias coerentes. Fica a cargo de o entrevistador direcionar as perguntas, pois os usuários podem não saber realmente explicar o que querem de imediato.
- Os questionários são usados para definir as prioridades das histórias ,quando o software que está a ser desenvolvido for abranger uma grande quantidade de usuários. Na primeira iteração não é feita a coleta, assim a coleta de histórias e as informações mais amplas deverão ser coletadas após a primeira iteração.
- A observação consiste em observar as pessoas trabalhando com o software para levantar informações tais como facilidades e dificuldades dos usuários. Sempre que possível é altamente recomendado seguir esta técnica.
- O workshop de escrita de histórias é uma das melhores técnicas levantamento de histórias. Ela consiste em reunir usuários, desenvolvedores, clientes e outras partes que podem contribuir com a criação das histórias em uma reunião onde cada individuo irá escrever o máximo de histórias possível. Após, ainda na reunião, é feito um protótipo de baixa fidelidade, que é a organização lógica das histórias. Como o protótipo não é um artefato de longo prazo no desenvolvimento ele devará ser jogado fora após alguns dias, já que serão feitos mais workshops com foco em papéis ou personas diferentes (COHN, 2004).

## 2.5. Substitutos de usuários

Uma vez que as histórias são escritas através de uma entidade específica, como um usuário, ou cliente, por exemplo, este tipo de escrita é chamada de substituto de usuários (*user proxies*), que nada mais é do que escrever histórias por diferentes perspectivas (COHN, 2004).

Existem vários tipos de substitutos que podem escrever histórias, como um gerente de desenvolvimento, vendedores, especialistas do domínio, o grupo de marketing, ex-usuários, clientes, formadores ou suporte técnico, analistas de negócio ou sistemas. Estas perspectivas denotam, para cada uma, os próprios interesses para o software a ser implementado, portanto não há uma escolha certa ou errada sobre qual tipo de perspectiva (*user proxy*) a ser adotada. O mais indicado é que a equipe do cliente contenha mais de um tipo de substituto de usuário, para que o software atenda uma variedade maior de usuários (COHN, 2004).

Uma equipe com mais de uma perspectiva tende a ser uma equipe boa, pois as deficiências de uma parte são supridas por outra, até que a equipe fique balanceada. Existem quatro passos para a criação da equipe do cliente (COHN, 2004):

1- Lembrar que, usuários reais vencem qualquer perspectiva;

2- Adicionar usuários reais. Caso o software contenha mais de tipo de usuário, adicione a maior variabilidade de tipos de usuário possível;

3- Identifique um responsável na equipe do cliente, que normalmente é o gerente do produto;

4- Determine os fatores críticos para o sucesso do projeto;

## 2.6. Testes de aceitação

Ao se tratar de histórias e suas implementações, os testes de aceitação para as histórias de usuário são de fundamental importância para expressar os diversos detalhes resultantes entre a comunicação entre o cliente e os desenvolvedores. Assim, escrever testes de aceitação antes da implementação facilita o processo de desenvolvimento, pois esses testes vão levantar grandes quantidades de informações, que serão lembradas constantemente pelos desenvolvedores, evitando que ao longo do processo esses detalhes sejam esquecidos (COHN, 2004). Outro detalhe sobre a escrita de testes de aceitação, é que devem ser escritas pelo cliente, pois o sistema em questão será desenvolvido para satisfazer as ideias do(s) cliente(s). Quando necessário o cliente e o programador podem criar testes em conjunto, mas o cliente é quem possui o papel de criar os testes de maneira minuciosa para saber quando a história foi implementada corretamente. Testes devem ser escritos a medida em que valor (mais detalhes) e clareza são adicionados ao projeto (BECK; FOWLER, 2004). A figura a seguir ilustra um possível teste de aceitação para a história da Figura 2.1.

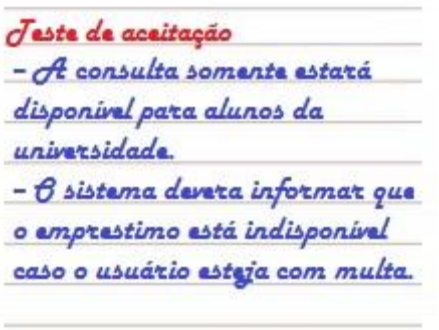


**Figura 2.1 –** Exemplo de testes de aceitação em um cartão de história.

Ao final de cada iteração o cliente e a equipe de desenvolvedores, irão se reunir, e com base nas histórias escritas e implementadas realizarão a verificação quanto ao funcionamento da mesma dentro do contexto de testes, estabelecido no cartão.

No desenvolvimento de um software o tempo é algo crucial, e testes levam tempo. Portanto usar um Framework para testes integrados pode diminuir o tempo gasto com testes em cada iteração. Existem vários tipos de testes e as equipes do cliente e do desenvolvedor devem trabalhar em conjunto para garantir que os tipos certos de testes estão sendo rodados no projeto (COHN, 2004). Alguns tipos de testes importantes segundo Mike Cohn (2004) são:

- Testes de interface, que analisam o comportamento das interfaces.
- Testes de usabilidade, que analisam se o software pode ser facilmente utilizado.
- Testes de performance, que analisam o quão bem o software irá rodar em diferentes cargas de sistema.
- Testes de estresse, que sujeitam o software a valores extremos para determinado domínio.

## 2.7. Guia para boas histórias

De acordo com Soares e Ferreira (2010) não há uma maneira ideal para garantir uma boa divisão de histórias exata e concisa, mas é recomendado levar em conta o tamanho das tarefas, bem como o valor (importância para o projeto) da história para o todo, sendo assim as histórias que não devem ultrapassar cinco dias, ou uma iteração para ser desenvolvida. As histórias que podem ser desenvolvidas por mais de um desenvolvedor devem ser divididas em mais histórias e, caso uma história seja difícil de estimar por conta de sua complexidade, ela deve ser separada de outras histórias, assim o grau de dependência desta história para outras será diminuída.

Visto a grande relevância da escrita das histórias Mike Cohn (2004) descreve alguns pontos que quando seguidos podem fornecer histórias exatas e concisas, como:

- Começar com histórias principais, ou seja aquelas que vão atribuir as funções primordiais do produto;
- “Fatiar o bolo”, já que grandes as histórias grandes (épicos) que ao serem divididas perdem o sentido, sendo assim, ao dividir histórias grandes é importante ficar atento se ela será útil mesmo após ser dividida (PICHLER, 2010);

- Escrever histórias fechadas, que são aquelas que descrevem funcionalidades importantes para o sistema, assim quando elas forem finalizadas o usuário terá a sensação de que algo foi completo e que o projeto está evoluindo;
- Colocar constantes nos cartões para histórias que possuem algo que precisa ser obedecido ao invés de implementado, como por exemplo: o sistema terá que suportar mais de 30 usuários conectados diretamente;
- Histórias planejáveis, ou seja, aquelas que caibam em uma iteração e não sejam grandes a ponto de perder precisão;
- Mantenha o projeto sem interface gráfica até chegar ao fim, pois ao incluir as interfaces os integrantes do projeto podem não saber diferenciar o que são requisitos e o que são soluções de especificação;
- Algumas coisas não são histórias, pois apesar da técnica de histórias de usuários ser uma técnica muito abrangente ela pode não expressar alguns requisitos tão bem como outros formatos, assim, se for preciso, utilize outras técnica para suprir essa deficiência;
- Inclua papéis de usuários nas histórias, pois isso trará a sensação real de que determinada história está sendo implementada para um usuário que precisa utilizar está história;
- Escreva para um usuário, em grande parte dos casos, não fará diferença escrever para um ou para muitos, mas em alguns casos as histórias podem apresentar diferenças significativas para o sucesso do projeto por serem mais especificas, direcionadas a uma classe de usuários específica;
- Escreva as histórias em voz alta para facilitar a compreensão;
- O Cliente escreve as histórias, como foi dito anteriormente, a responsabilidade está com os clientes, pois eles possuem a visão do sistema;
- Não numere os cartões de história, pois isso vai isso vai trazer a falsa sensação de rastreamento, ou seja, um cartão que possui um numero poderia trazer a ideia que este numero determina sua importância ou ordem de implementação, o que não é valido para as histórias de usuário;
- Não esqueça o propósito, pois a real função do cartão é atuar como lembrete para discutir determinada característica;

Estes são os principais conceitos sobre as histórias de usuário. Sendo assim, no próximo capítulo será apresentada a ferramenta de modelagem de processo utilizada e as principais definições teóricas sobre ela.

No próximo capítulo, primeiramente será feito uma contextualização entre o EPF-Composer a computação auxiliada por engenharia de métodos (CAME), seguido de uma breve apresentação sobre as características básicas do SPEM e a organização dessas características dentro do EPF-Composer.

## 2.8. Casos de uso Vs. Histórias de usuários

Já que o OpenUP/Basic é um processo ágil, como a disciplina de requisitos implementada por ele (casos de uso) pode não ser ágil?

Os casos de uso é uma técnica utilizada para levantamento de requisitos envolve grande documentação, isso pode agregar problemas de completeza, ambiguidade e falta de clareza dos requisitos. Com isso, os desenvolvedores poderão perder tempo para compreender os casos de uso escritos (BITTNER et. al., 2003).

Outro fator, é que os casos de uso é um artefato de abstração conceitual e não de implementação, por isso telas e protótipos devem estar associados ao caso de uso e não o contrário (BITTNER et. al., 2003).

Já as histórias de usuário não envolvem grande documentação e envolvem o usuário no processo de levantamento de requisitos ativamente durante todo o processo de desenvolvimento de software (COHN, 2004).

Devido a essas diferenças, foi criado os “Casos de Uso 2.0”, que oferece o levantamento de requisitos utilizando histórias de usuário, tornando-se ideal para projetos que utilizam metodologias ágeis (JACOBSON, 2011).

# 3
# Contextualização

*Neste capítulo será apresentada a ferramenta utilizada para a customização das histórias de usuário e os conceitos teóricos necessários para a compreensão de seu funcionamento.*

De maneira análoga a computação auxiliada por engenharia de software (CASE), que visa facilitar a engenharia de software através de ferramentas especializadas, a computação auxiliada por engenharia de métodos (CAME) se esforça para fornecer um amplo suporte às atividades utilizadas pelos engenheiros de métodos (NIKNAFS; RAMSIN, 2008).

Engenharia de métodos pode partir de um método já existente, ou pode ser desenvolvida do zero, dependendo dos requisitos do projeto. Considerando isso, quando um método precisa ser desenvolvido para um projeto especifico a fim de aperfeiçoar o processo de desenvolvimento, pode ser dito que foi aplicada a Engenharia de Métodos Situacional (SME)(NIKNAFS; RAMSIN, 2008).

Por fim, um ambiente CAME é composto por uma série de ferramentas correlacionadas com o objetivo de facilitar, de forma ideal, toda a SME (NIKNAFS; RAMSIN, 2008). A Figura 3.0 a seguir, mostra uma imagem simples do framework.

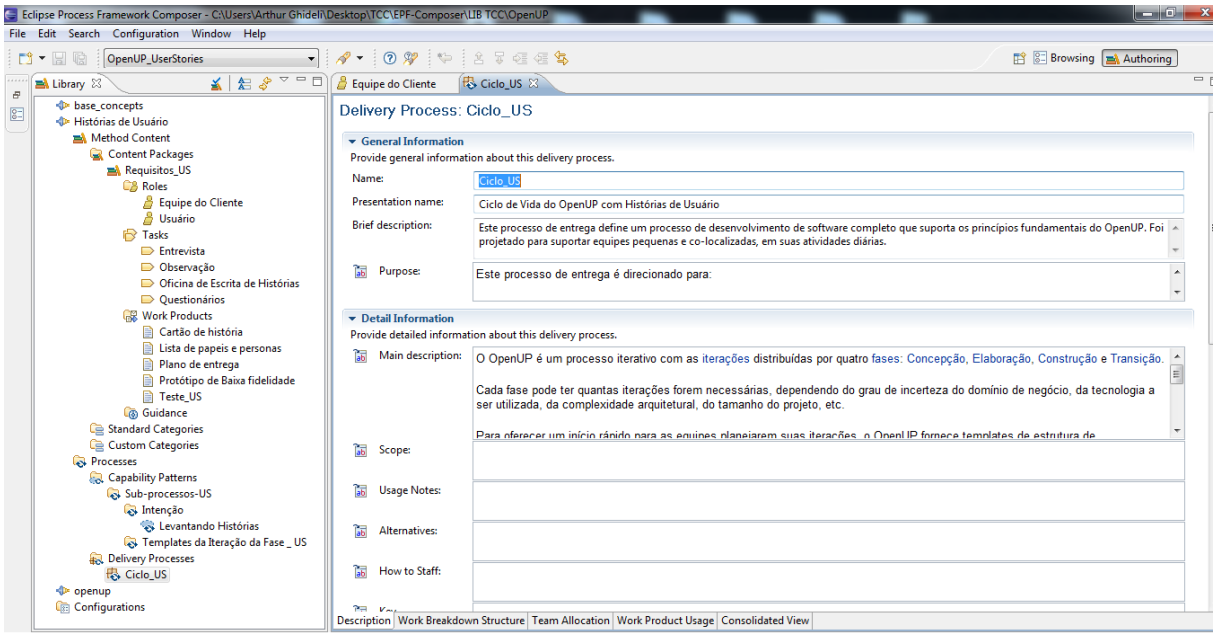


**Figura 3.0 –** O framework EPF-Composer.

## 3.1. Métodos

Um método é composto de duas partes, sendo a primeira o produto, que captura os conhecimentos relacionados com o mesmo. E a parte de processo, que inclui atividades relacionadas aos aspectos do método. Com essa divisão o processo pode ser dividido em dois fragmentos:

1. Fragmentos do produto: compostos por artefatos como modelos, diagramas e documentos.
2. Fragmentos do processo: compostos por atividades, fases e tarefas.

## 3.2. O OpenUP/Basic e o Eclipse Process Framework Composer

O OpenUP/Basic é um processo de desenvolvimento, que foi desenvolvido pela IBM com o intuito de criar uma versão ágil do processo unificado, conhecido como RUP (*Rational Unified Process*). Sendo assim, o OpenUP/Basic preserva as características básicas do processo unificado, que possui o desenvolvimento iterativo e incremental, gerenciamento de requisitos, gerenciamento de projetos e gerenciamento de riscos. Neste sentido o OpenUP/Basic fornece as melhores práticas de engenharia de software, sendo uma parte do Eclipse Process Framework (EPF), que é um framework de processo de livre, desenvolvido dentro da fundação eclipse (SANTOS, 2009).

O EPF-Composer é uma plataforma de ferramentas para engenheiros de métodos e lideres de projetos. Estes são os responsáveis por implementar e dar manutenção aos processos para organizações de desenvolvimento ou projetos individuais (EPF, 2007).

O framework é baseado no meta-modelo SPEM, que é uma linguagem de modelagem, ou seja, um conjunto de construtores e regras para definir e modelar processos de software (OMG, 2008). Neste sentido é preciso, para compreender inicialmente como o EPF-Composer funciona ficar familiarizado com produtos de trabalho, atividades, papeis, disciplinas, orientações, tarefas e padrões de recurso (OMG, 2008).

1. Produtos de trabalho são descrições de algo que contém informações sobre o produto ou é são entidades físicas (documentos) produzidas ou usadas por atividades do processo;

2. Atividades descrevem determinadas tarefas que um papel realiza dentro de um processo;
3. Papeis descrevem responsabilidades e competências que um determinado indivíduo tem dentro do processo;
4. Disciplina é um agrupamento coerente de elementos do processo (artefatos, papéis, atividades) cujas atividades são organizadas segundo algum ponto de vista ou tema comum;
5. Orientação é um elemento que ajuda a instruir outros elementos em sua realização;
6. Tarefas são ações realizadas por um determinado papel que podem ou não produzir um artefato (produto de trabalho);
7. Padrões de recurso expressam e transmitem conhecimento do processo, como uma prática, por exemplo, de levantar requisitos. Desta forma eles, são dentre outras palavras, blocos de construções com boas práticas de engenharia de software, que podem ser reutilizados em diversos projetos.

A Figura 3.1, mostra um diagrama de uma atividade que possui atores (*software architect, system analyst, Project manger)*, uma tarefa (*prioritize use cases)* e produtos de trabalho (*risk list, requirements atributes).* O(s) ator(es) realiza(m) a(s) tarefa(s) que pode(m) possuir produto(s) de trabalho como entrada, e produz(em) produto(s) de trabalho como saída ao fim de sua execução.

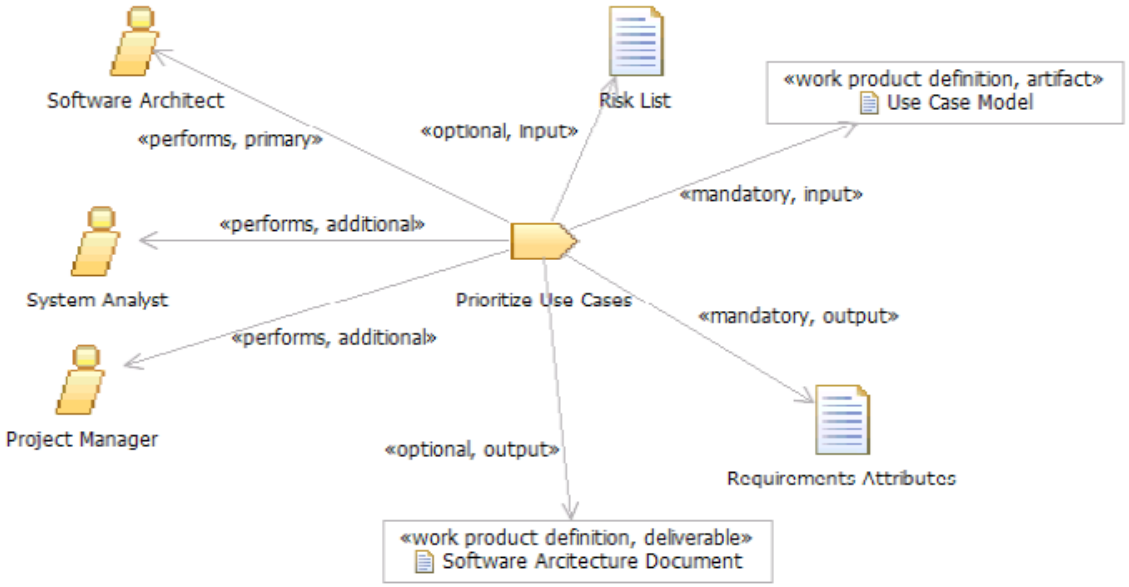


**Figura 3.1 –** Funcionamento simples de uma tarefa segundo o SPEM. **Fonte:** (OMG, 2008).

A Figura 3.2, a seguir, mostra alguns dos itens citados anteriormente e suas respectivas localizações no EPF-Composer.

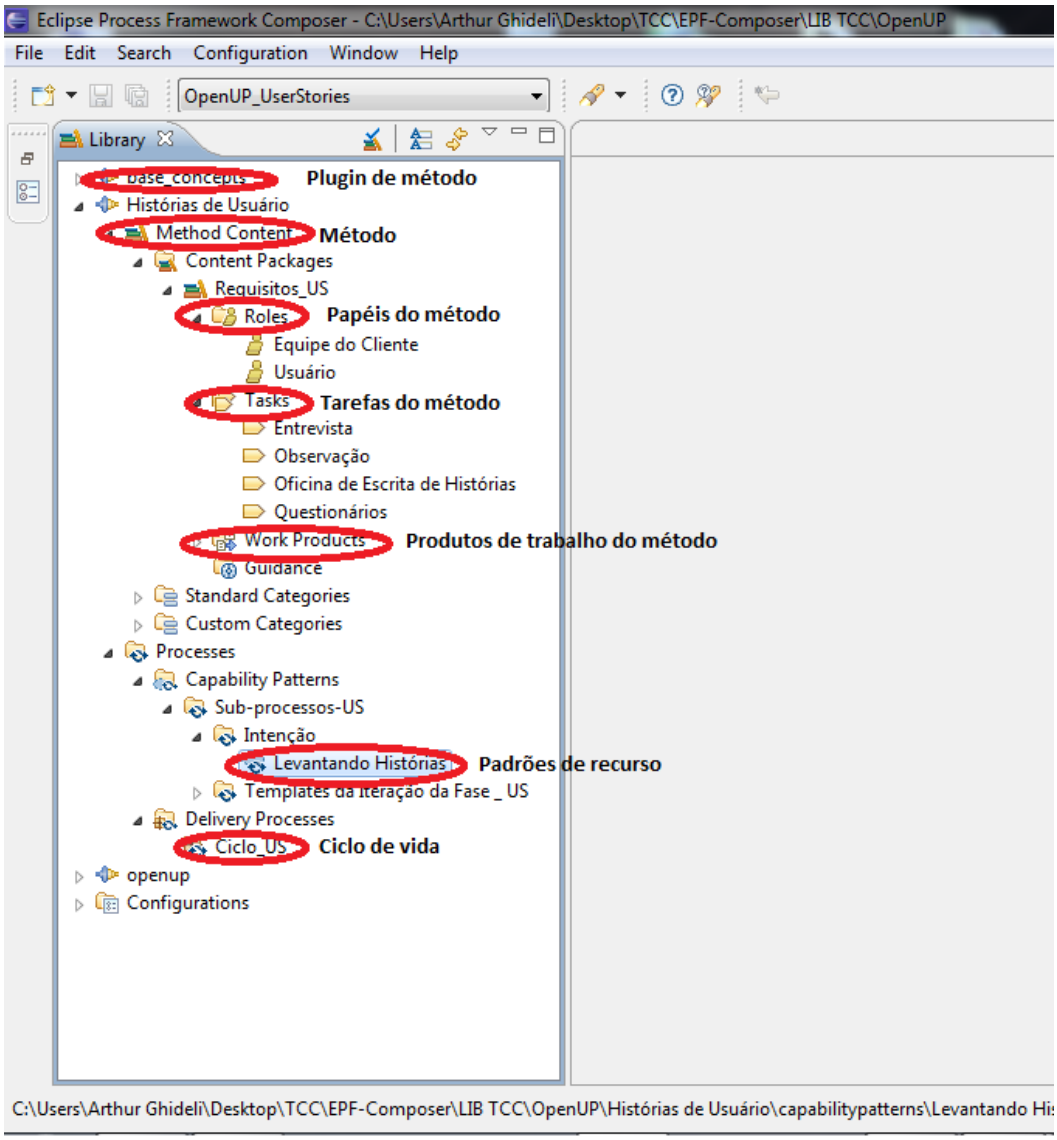


**Figura 3.2** – Organização do framework.

Por fim, o EPF-Composer é essencialmente um aplicativo de gerenciamento de conteúdo e publicação, apresentando como resultado um site com a orientação dos métodos e processos, criados ou customizados, que podem ser utilizados pela equipe do projeto sempre que necessário (EPF, 2007).

A Figura 3.3, a seguir, mostra uma imagem de uma página web publicada pelo EPF-Composer.

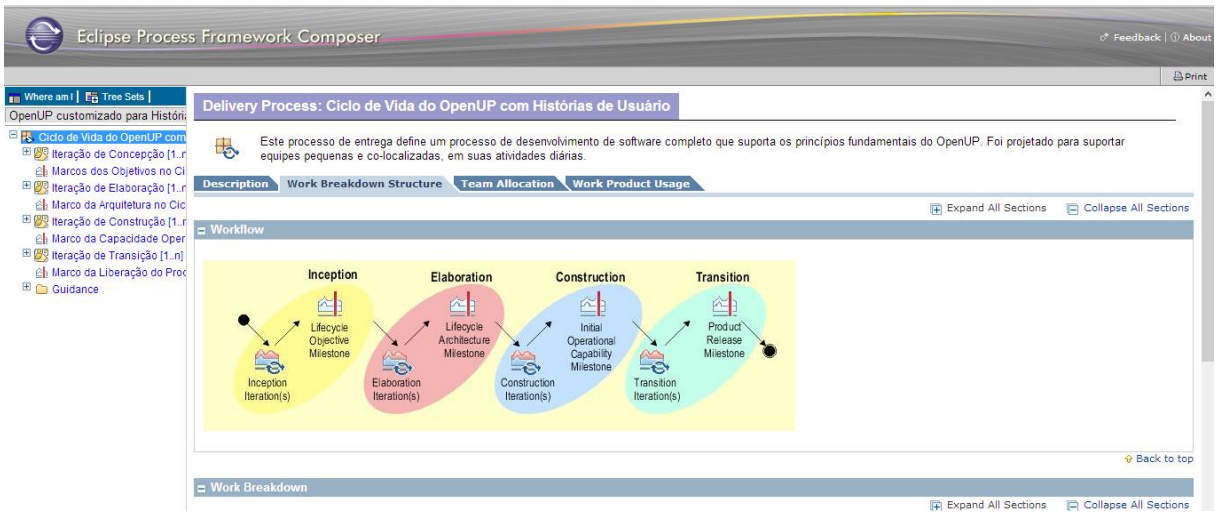


**Figura 3.3 –** Página web resultante da publicação efetuada pelo EPF-Composer.

Os usuários podem publicar um ciclo de vida a partir do menu “Publish”, produzindo uma página web com os dados produzidos nos métodos.

Agora que foram introduzidos os conhecimentos necessários sobre as histórias de usuário e o EPF-Composer, no próximo capítulo será mostrada a customização do OpenUp/Basic.

# 4
# Customização do OpenUP/Basic

*Este capítulo apresenta a criação da disciplina de requisitos focada em histórias de usuário, a criação do padrão de recurso para a disciplina e a customização do OpenUP/Basic para suporte da mesma.*

Para customizar métodos é necessário, primeiramente, obter um raciocínio pertinente ao método que será customizado, bem como a sua organização. Como, nesta monografia, será customizada a disciplina de requisitos, todo o fluxo de trabalho da disciplina será remodelado. Como as histórias de usuário foram estudadas previamente, o fluxo de trabalho (*workflow)* da disciplina ficará semelhante ao da Figura 4.0. Primeiramente é selecionado uma ou mais abordagens que serão utilizadas para a obtenção de histórias (questionários, workshops, entrevistas, observação), produzindo uma lista de abordagens. Feito isso, é realizado o levantamento de histórias, que produz como saída os cartões de histórias. Por ultimo, o código é testado com a realização dos testes de aceitação, produzindo um código testado.

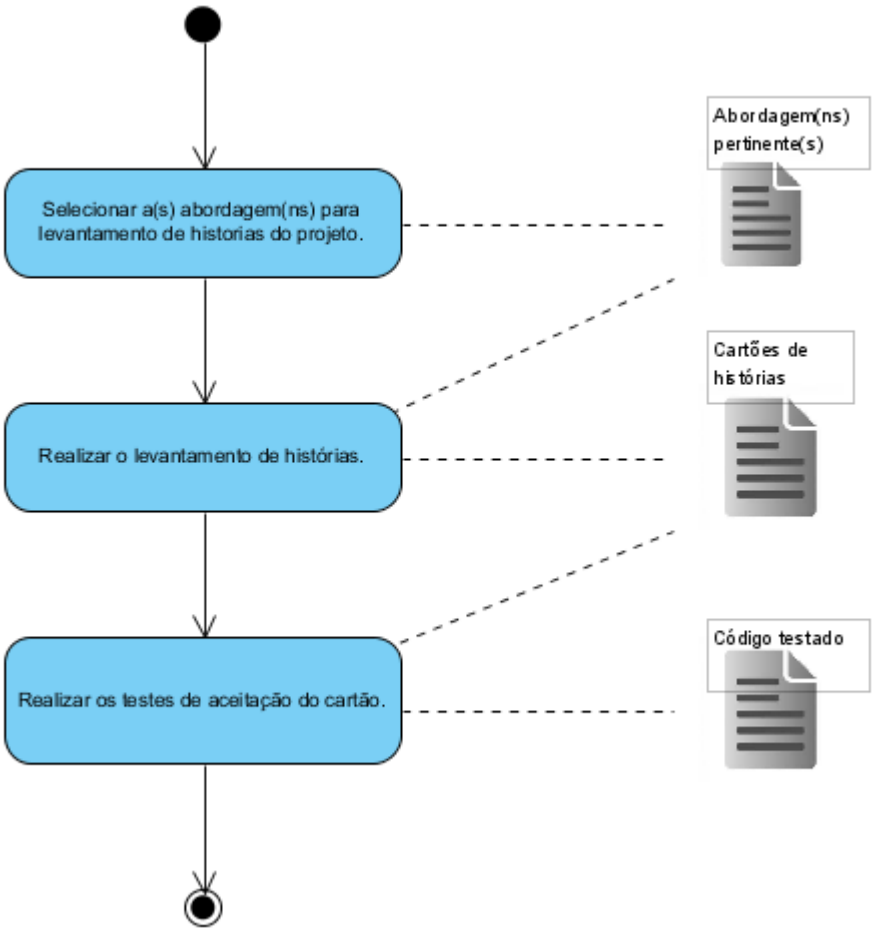


**Figura 4.0 –** Fluxo de trabalho da disciplina de requisitos com histórias de usuário.

Tendo o raciocínio solidificado o próximo passo é obter uma biblioteca com os métodos que serão customizados e abrí-los na ferramenta onde o mesmo serão customizados, no caso, o EPF-Composer. A Figura 4.1, a seguir, mostra o OpenUp/Basic, em seu estado comum, sem nenhuma modificação ou customização.

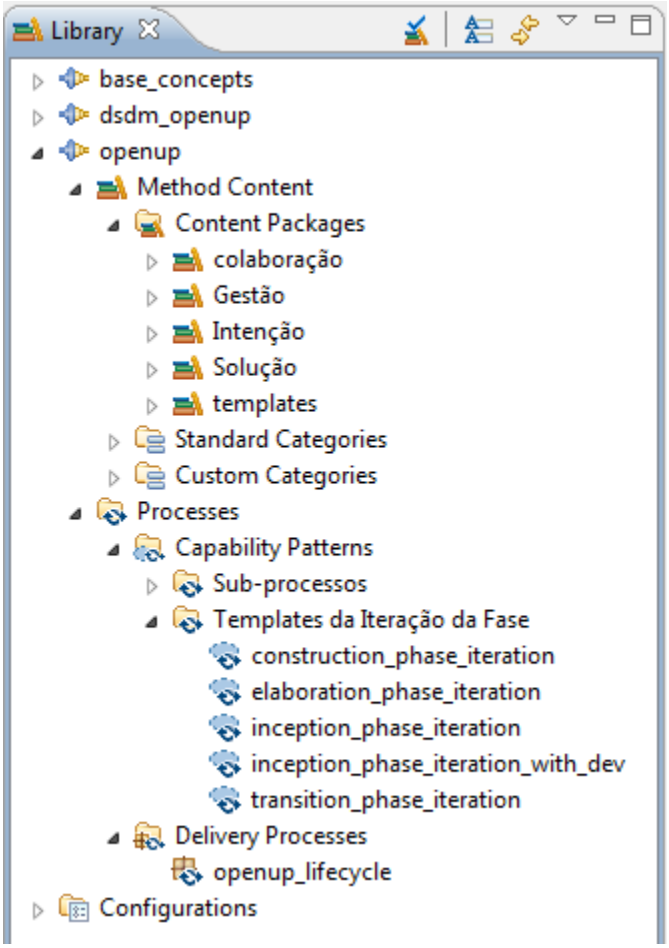


**Figura 4.1 –** OpenUP/Basic, sem customizações no EPF-Composer.

Pode-se notar que o método possui diversos conteúdos dentro do *Method Content*, como colaboração, gestão, intenção, solução e templates. É possível também visualizar os templates de fase na Figura 4.1, onde o conteúdo do "*Method Content"* está sendo aplicado em diversos "*Capability Patterns"* (padrões de recurso).

Tendo essa primeira parte em conhecimento, para o começo da customização de métodos, primeiramente é necessário a criação de um conjunto onde o novo conteúdo será criado. Sendo assim, na Figura 4.2, foi criado o conjunto (method plugin) nomeado de "Histórias de usuário", onde foi criado um conteúdo de método, nomeado intuitivamente de "Requisitos-US".

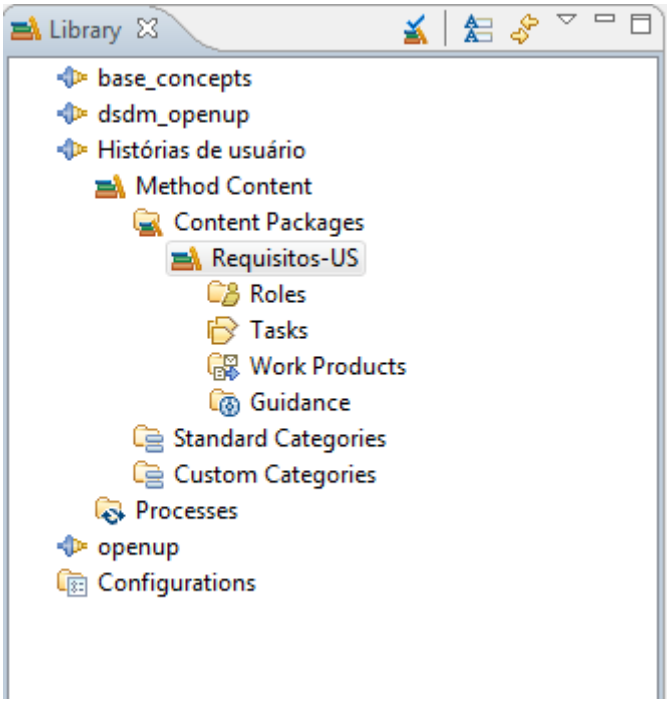


**Figura 4.2 –** Criação do método do plug-in "Histórias de usuário" e do método "Requisitos-US".

O próximo passo de qualquer customização que um usuário poderá fazer estará dependente do entendimento do método a ser customizado. A Figura 4.3, mostra o diagrama atividades para a disciplina de levantamento de requisitos focado em histórias de usuários. Pode-se observar que ela possui quatro diferentes tarefas (abordagens) que podem ser utilizadas para a obtenção de história, onde cada uma delas possui produtos de trabalho de saída, respectivamente.

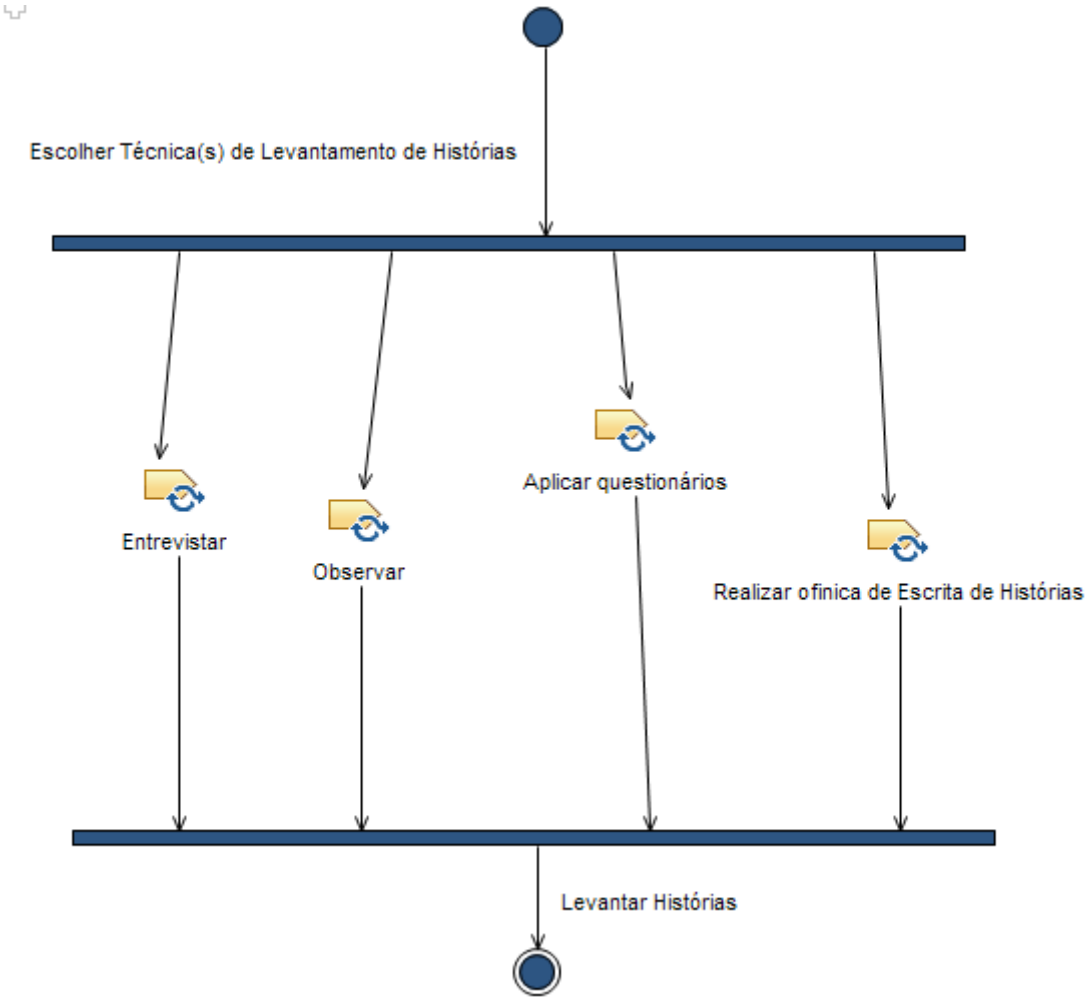


**Figura 4.3 –** Diagrama de atividades para a disciplina de requisitos.

Como pode ser notado na Figura 4.3, todas as tarefas de levantamento de histórias podem ocorrer, simultaneamente ou não, dependendo das necessidades do projeto. Esta figura é um padrão de recurso para a disciplina de requisitos com a abordagem por histórias de usuário.

É preciso ter em mente diversos detalhes sobre a modelagem, bem como os atores e produtos de trabalho para cada tarefa que será modelada. Como pode ser observado na Figura 4.3 as tarefas foram criadas, dentro do pacote "*tasks*" do "Requisitos_US", que é ilustrado na Figura 4.4.

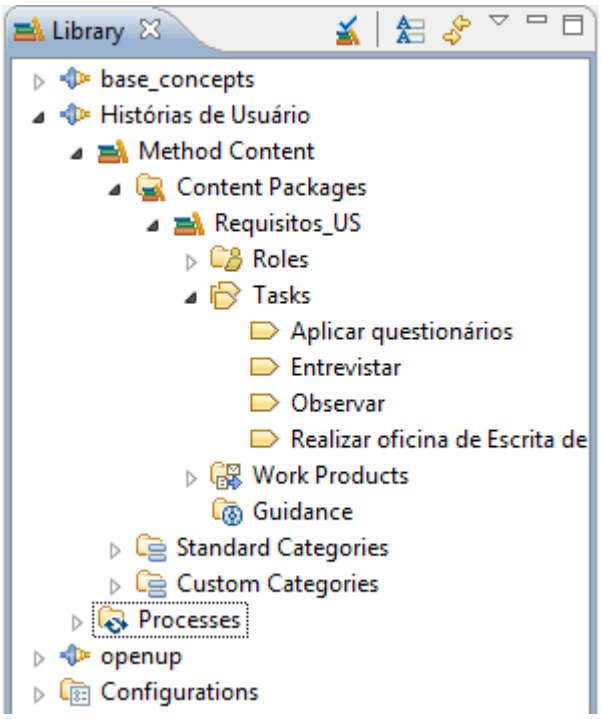


**Figura 4.4 –** Tarefas do método "Requisitos_US".

Uma vez criadas as tarefas para o levantamento de histórias bem como a sua organização (em termos de entrada em saída e descrição), podemos criar um padrão de recurso que poderá ser utilizado em outros métodos quando necessário simplesmente importando o plug-in.

A Figura 4.5, mostra passo a passo, como criar um padrão de recurso reutilizável para outros projetos.

- 1º Passo mostra que uma pasta para o para padrão de recurso precisa ser criada, como por exemplo, "Sub-processos-US".
- 2º Passo mostra a criação de um padrão de recurso novo
- 3º Passo mostra que as tarefas (observar, aplicar questionários, entrevistar, realizar oficina de escrita de histórias) foram incluídas ao padrão de recurso, simplesmente clicando com o botão direito em "Levantando Histórias" e adicionando as tarefas que desejas do "Method Content".

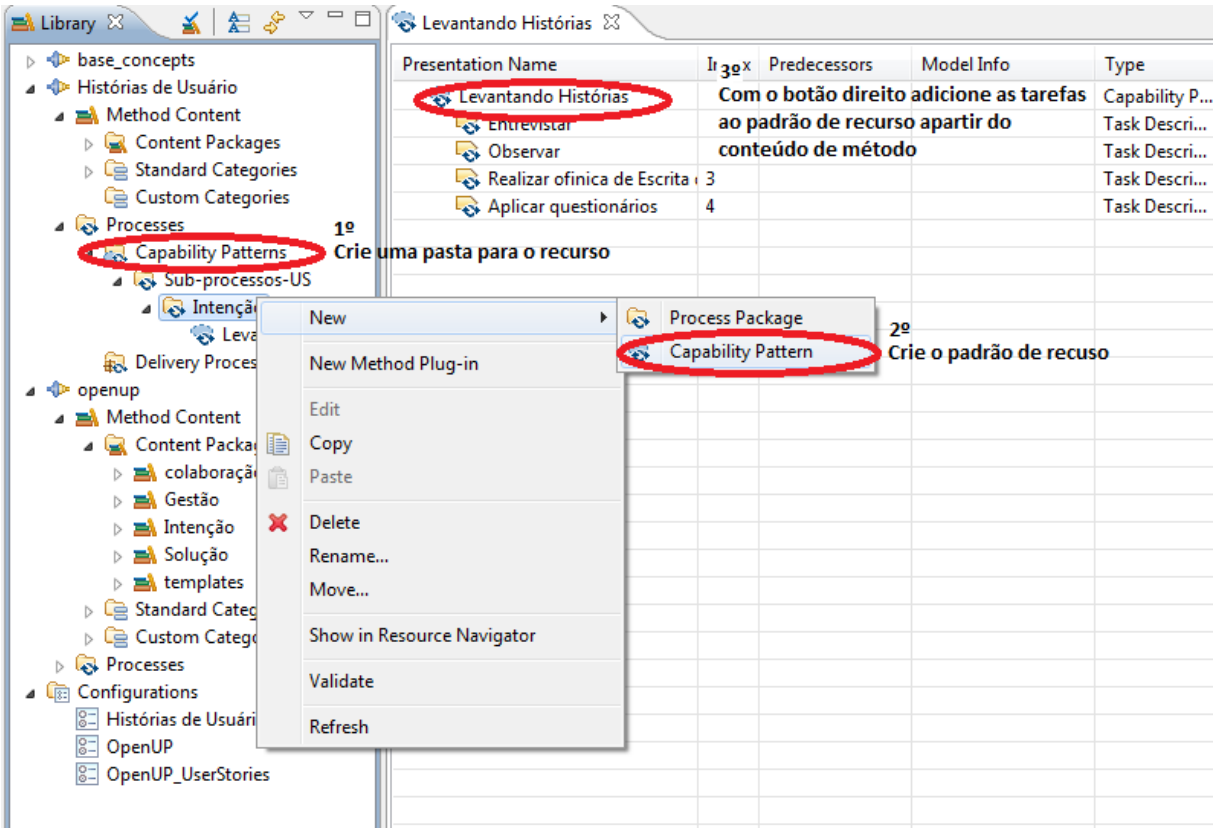


**Figura 4.5 –** Criando um padrão de recurso para as histórias de usuário.

Feito isso, teremos o novo padrão de recurso da disciplina de requisitos com histórias de usuário, que poderá ser reutilizado em qualquer projeto. Este padrão pode ser exportado a partir do menu “File -> Export -> Method plug-ins”. Para importar o plug-in para o projeto, basta ir no menu “File -> Import -> Method plug-ins”.

Como a criação do método na biblioteca foi a partir do zero e não referenciamos a biblioteca do Open/Up e nenhum de seus conteúdos, pode ser compreendido que o método de levantar histórias de usuário é um plug-in independente, por enquanto. Sendo assim será assumido que o método foi importado para a biblioteca do OpenUp para a customização. Agora que o plug-in de histórias de usuário foi incluído na biblioteca do EPF-Composer em conjunto com a biblioteca do OpenUP/Basic, resta apenas customizar o ciclo de vida do OpenUp/Basic para suportar a nova disciplina de requisitos, criando uma nova configuração de publicação e efetuando a publicação.

Para isto, o ciclo de vida do OpenUp/Basic deve ser editado. Para garantir o funcionamento das publicações feitas anteriormente, é recomendado copiar o ciclo de vida do OpenUp, e colá-lo com outro nome, assim sua edição não irá influenciar em partes que estão estáveis ou em uso.

Após criar o novo ciclo de vida é preciso customizá-lo. Alguns passos devem ser seguidos para que a customização funcione perfeitamente, como é mostrado nas Figuras 4.6, 4.7, 4.8.

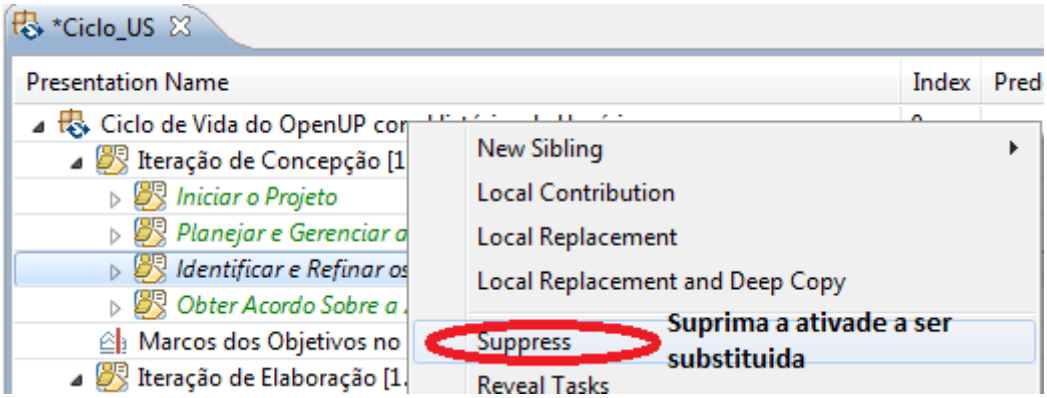


**Figura 4.6** – Suprimindo a padrão de recurso corrente no ciclo de vida.

Como pode ser observado, neste passo, a atividade "Identificar e refinar requisitos" foi suprimida. Feito isso as tarefas pertinentes à aquela atividade serão excluídas do ciclo de vida e não irão mais aparecer nas publicações.

O segundo passo será criar uma atividade filha da fase de iteração e nomeá-la, como mostra a Figura 4.7.

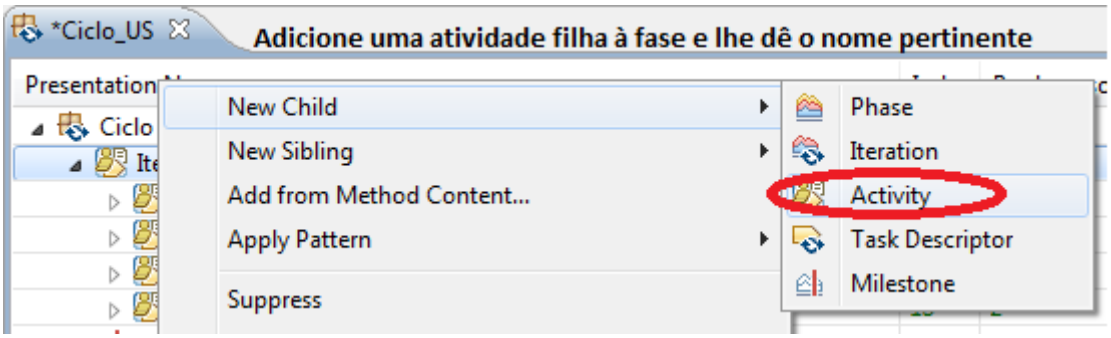


**Figura 4.7** – Criando uma atividade filha na iteração de fase.

Por ultimo, o padrão de recurso deve ser adicionado àquela atividade, a fim de fazer parte do novo ciclo de vida. Para isso a Figura 4.8, a seguir, mostra passo-a-passo como deve ser feito a inclusão do recurso no ciclo de vida.

O primeiro passo mostra que é preciso abrir a barra de propriedades no canto inferior direito para poder modificar o tipo de variabilidade da atividade no segundo passo. Por fim, o terceiro passo mostra a adição no novo padrão no ciclo de vida.

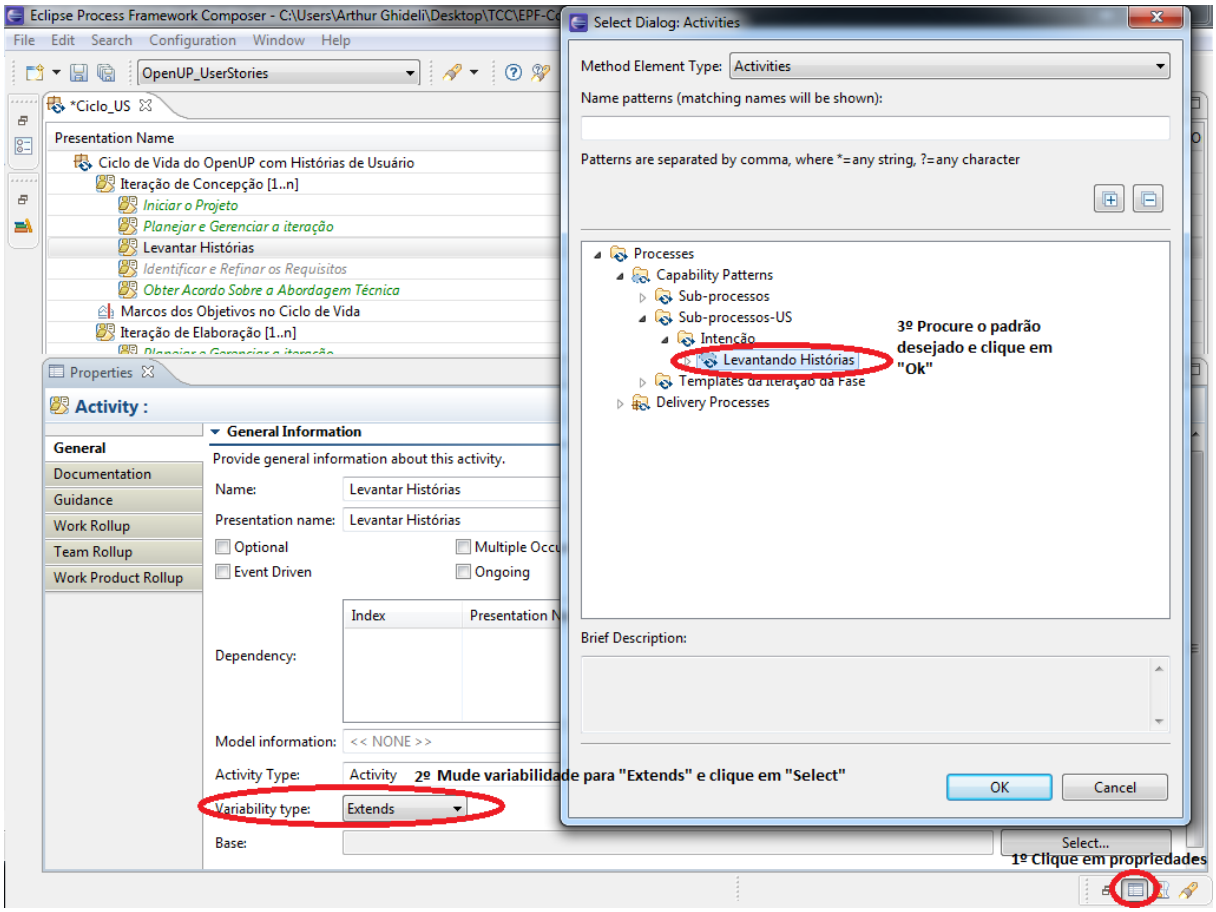


**Figura 4.8** – Passo-a-passo para a adição do padrão de recurso de histórias de usuário.

Feito isso o resultado esperado é a inclusão do padrão de recurso no ciclo de vida e a adição de todas as tarefas pertinentes a ele, como mostra a Figura 4.9.

*Ciclo_US

| Presentation Name | Index | Predecessors | Model Info | Type |
|---|---|---|---|---|
| Ciclo de Vida do OpenUP com Histórias de Usuário | 0 | | | Delivery Pro... |
| Iteração de Concepção [1..n] | 1 | | extends 'inception_... | Activity |
| Iniciar o Projeto | 2 | | extends 'initiate_pr... | Activity |
| Planejar e Gerenciar a iteração | 5 | | extends 'plan_man... | Activity |
| Levantar Histórias | 9 | | extends 'Levantando Histórias, Histó | |
| Entrevistar | 10 | | | Task Descri... |
| Observar | 11 | | | Task Descri... |
| Realizar ofinica de Escrita de Histórias | 12 | | | Task Descri... |
| Aplicar questionários | 13 | | | Task Descri... |
| Identificar e Refinar os Requisitos | 14 | 2 | extends 'identify_a... | Activity |
| Obter Acordo Sobre a Abordagem Técnica | 18 | 2 | extends 'agree_tech... | Activity |
| Marcos dos Objetivos no Ciclo de Vida | 20 | 1 | | Milestone |

**Figura 4.9** – Resultado esperado após a customização.

Existem outras maneiras de efetuar uma customização de um ciclo de vida, que são um pouco mais complexas, mas são mais coerentes, como a customização do padrão de recurso diretamente, e não no ciclo de vida. Assim cada padrão de recurso modificado teria que ser copiado e modificado.

Agora que o ciclo está pronto, portanto basta adicioná-lo em uma perspectiva de visualização, para que ele possa aparecer na publicação. A Figura 4.10 mostra que o ciclo de vida customizado anteriormente foi adicionado à "*Custom Categories*". Para que o novo ciclo de vida possa ser publicado com êxito, a Figura 4.10 mostrou que foi criado a categoria "OpenUP – US" em "*Custom Categories*" e dentro dela foi designado o ciclo de vida "Ciclo_US", simplesmente clicando em "Assign" e escolhendo o ciclo de vida customizado.

**Figura 4.10 –** Adicionando uma perspectiva de visualização.

Outros conteúdos poderiam ser adicionados à árvore de visualização, como guias, papéis, disciplinas, mas para facilitar o entendimento e a visualização do que foi feito, basta incluir o ciclo de vida.

Por fim, devemos criar uma configuração para fazer a publicação. Para isso basta criar uma nova configuração, adicionar todos os plug-ins que são referenciados e adicionar a categoria "OpenUP – US" no botão "Add view", localizado na aba "Views".

A Figura 4.11, mostra como foi feita a criação da configuração. O primeiro passo foi criado simplesmente a configuração. O segundo passo mostra aonde os plug-ins deverão ser adicionados. O ultimo passo mostra que na aba "Views" a categoria "OpenUP – US" foi adicionado à publicação.

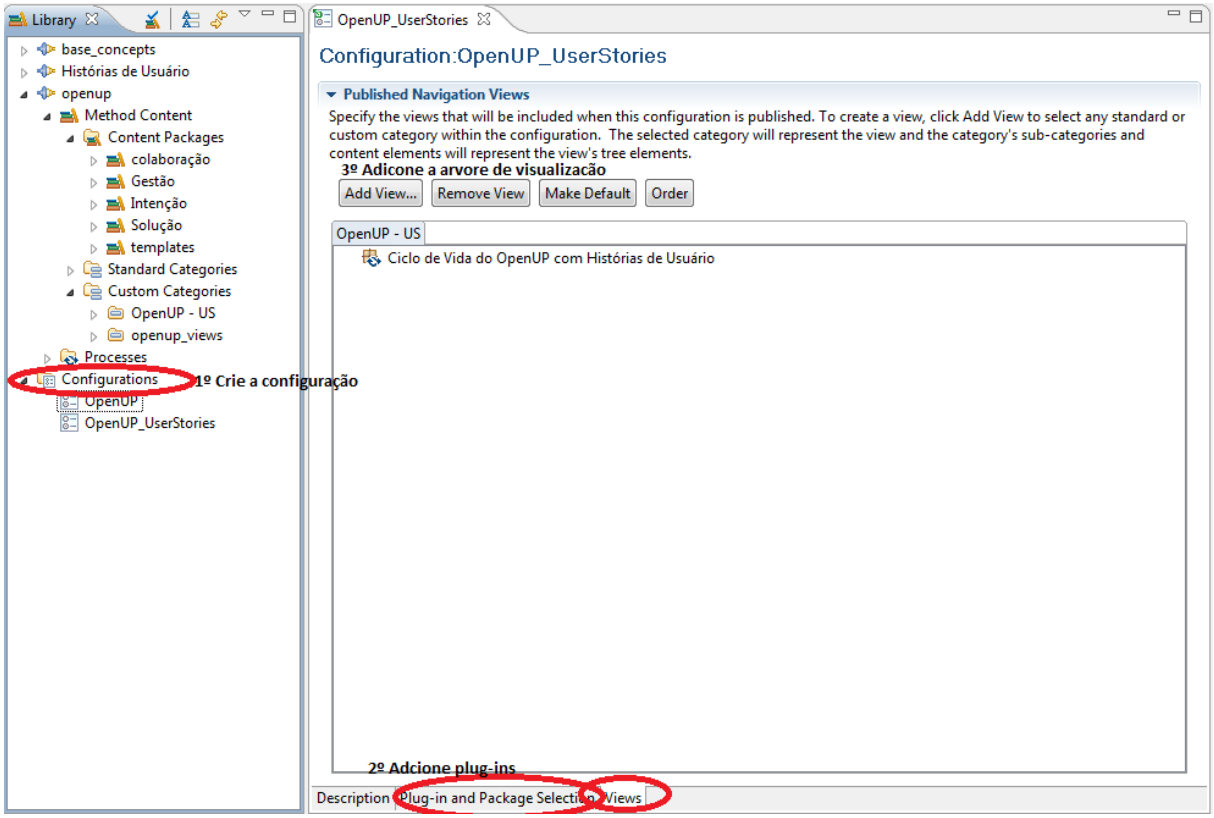


**Figura 4.11 –** Criando a configuração de método para a publicação.

Finalmente, basta publicar a configuração criada no ultimo passo no menu “Publish”. Note que existem diversas opções que podem ser selecionadas, desde um simples padrão de recurso a configurações inteiras.

Esta sessão demonstrou como é feita a modelagem de métodos e a customização de um ciclo de vida, mostrando como o raciocínio começa com o estudo do fluxo da disciplina e o entendimento da mesma e de suas atividades. Uma vez que o entendimento foi solidificado foi possível modelar todo o processo com a ferramenta de métodos, o EPF-Composer e obter um novo método customizado.

# 5 Conclusões e Trabalhos Futuros

*Este capítulo apresenta as conclusões relacionadas a este trabalho juntamente aos trabalhos futuros.*

## 5.1. Conclusões

A partir do estudo das histórias de usuário, bem como o seu entendimento, foi possível compreender o fluxo das atividades, produtos e tarefas para a criação de um processo de levantamento de requisitos não tradicional. Este processo, por ser ágil, foi encaixado perfeitamente na disciplina de requisitos do OpenUP/Basic, modificando o seu ciclo de vida para suportar a nova técnica.

Tendo em vista que as histórias de usuário podem ser facilmente adaptadas ao OpenUp/Basic, foi criado um padrão de recurso, com o auxílio do EPF-Composer, para possibilitar que qualquer indivíduo possa adicionar as histórias de usuário a um ciclo de vida, ao invés de criar as histórias de usuário diretamente no OpenUP/Basic.

Este padrão, apesar de simplificado, disponibiliza um novo conjunto de opções para o levantamento de requisitos que pode ou não ser seguido por empresas desenvolvedoras de software.

É notável a facilidade que a ferramenta CAME proporcionou ao criar um método do zero, como foi o caso das histórias de usuário. O EPF-Composer induz o engenheiro de métodos a utilizar boas práticas de engenharia de software e fornece suporte a imagens, textos, anexos para a criação do mesmo, mostrando-se uma ferramenta que atende a grande parte das possíveis necessidades dos engenheiros de métodos.

## 5.2. Trabalhos futuros

O presente trabalho buscou customizar de maneira eficaz o ciclo de vida do OpenUp/Basic e criar um padrão de recurso para as histórias do usuário. Porém existem alguns pontos que podem ser estudados futuramente, sendo eles:

- ✓ Criar novos padrões de recurso;
- ✓ Verificar a adaptação do novo método em um contexto real de trabalho;
- ✓ Publicar um artigo que mostra como criar padrões de recurso utilizando o EPF-Composer.

Este trabalho verificou que é possível criar a técnica de levantamento de requisitos ágil, histórias de usuário, bem como seu padrão de recurso em poucos passos utilizando o EPF-Composer. Devido a este estudo engenheiros de métodos terão acesso ao padrão de recurso de histórias de usuário, e poderão utilizá-lo livremente em projetos, o que não estava disponível até então.

# REFERÊNCIAS

AGILE MANIFESTO. **Manifesto for Agile Software Development.** 2001.
Disponível em: <http://agilemanifesto.org/>.
Acessado em Fevereiro de 2013.

BITTNER, K et. al. **Use Case Modeling.** Massachusetts, Addison-Wesley (2003).

BECK, K. e FOWLER, M. **Planning Extreme Programming.** Addison Wesley (2000).

COHEN, D., LINDVALL, M., e COSTA, P. **Na Introduction to Agile Methods. In Advances in Computers** (p. 1 – 66). New York: Elsevier Science (2004).

COHN, M. **User Stories Aplied.** Massachusetts, Addison Wesley (2004).

DAVIES, R. **The Power of Stories.** XP 2001. Sardinia (2001).
Disponível em: <http://www.agilexp.com/presentations/PowerOfStories.pdf>.
Acessado em Abril de 2013.

ENGENHARIA DE SOFTWARE. **Introdução à Engenharia de Requisitos**. (2008).
Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=8034>.
Acessado em Setembro de 2012.

FERREIRA, S. **As Pesquisas Denominadas "Estado da Arte".** (2002).
Disponível em: <http://www.scielo.br/pdf/es/v23n79/10857.pdf>.
Acessado em Setembro de 2012.

JACOBSON, I., SPENCE, I., e BITTNER, K. **Use Case 2.0 – The Guide to Succeeding with Use Cases.** International (2011).

KOTONIO, G. e SOMMERVILLE, I. **Rquirements Enginering: Processes and Techniques.** Chichester, England (1998).

NIKNAFS, A. e RAMSIN, R. **Computer-Aided Method Engineering: An Analysis of Existing Environment.** Springer, Berlin Heidelberg (2008).

OMG. **Software & Systems Process Engineering Meta-Model Specification.** (2008). Disponível em: <http://www.omg.org/spec/SPEM/2.0/PDF/>.
Acessado em Fevereiro de 2013.

SANTOS, S. **OpenUP: Um processo ágil.** (2009) Disponível em: <http://www.ibm.com/developerworks/br/rational/local/open_up/index.html>.
Acessado em Março de 2013.

SOARES, I. e FERREIRA, L. **Melhorando a Comunicação com Histórias do Usuário**. Java Magazine, n. 86, p. 81-91, Agosto (2010).
Disponível em:<http://www.devmedia.com.br/post-18785-Melhorando-a-comunicacao-com-Estorias-do-Usuario-Java-Magazine-86.html>.
Acessado em Novembro de 2011.

PICHLER, R. **Agile Product Management with Scrum: Creating Products that Custumers Love** (p. 53). Addison-Wesley, Estados Unidos (2010).

SOMMERVILLE, I. **Engenharia de Software**. Pearson Education do Brasil, São Paulo (2003).

The Standish Group. **Chaos.** (2009).
Disponível em: <http://www.projectsmart.co.uk/docs/chaos-report.pdf>.
Acessado em Setembro de 2011.