5 UP – Processo Unificado

Este capítulo apresenta o Processo Unificado, um dos mais importantes padrões da indústria atual. Inicialmente, suas *características* principais (Seção 5.1) são explicadas e em seguida suas *quatro fases* originais (Seção 5.2). Na sequência, algumas implementações especificas do Processo Unificado são apresentadas, iniciando pelo *Processo Unificado Rational* ou *RUP* (Seção 5.3), a mais conhecida de todas, seguida pela apresentação de *AUP* (Seção 5.4) e *OpenUP* (Seção 5.5), que são implementações ágeis do Processo Unificado. *EUP* (Seção 5.6) é uma implementação que acrescenta aspectos empresariais que não constam na definição original do processo. *OUM* (Seção 5.7) é uma implementação para uso específico com ferramentas proprietárias. E, finalmente, *RUP-SE* (Seção 5.8), que é uma implementação orientada a sistemas complexos de grande porte.

O Processo Unificado (*UP* ou *Unified Process*) foi criado por três importantes luzes da orientação a objetos dos anos 90 (Jacobson, Booch, & Rumbaugh, 1999), sendo o resultado de mais de 30 anos de experiência acumulada em projetos, notações e processos.

O UP é o primeiro modelo de processo inteiramente adaptado ao uso com a *UML* (*Unified Modeling Language*), desenvolvida pelo mesmo grupo. Sua concepção foi baseada nas práticas de maior retorno de investimento (ROI) do mercado.

As atividades do UP são bem definidas, no seguinte sentido:

a) Elas têm uma descrição clara e precisa.
b) Apresentam responsáveis.
c) São definidos artefatos de entrada e saída.
d) São definidas dependências entre atividades.
e) Seguem um modelo de ciclo de vida bem definido.
f) Acompanha uma descrição sistemática de como executar cada atividade com o uso de ferramentas disponibilizadas (procedimentos).
g) Preconizam o uso da linguagem UML.

As atividades incluem *workflows*, que são grafos que descrevem as dependências entre diferentes atividades. *Workflows* são associados às disciplinas do Processo Unificado, que variam de implementação para implementação.

O RUP é considerado a principal implementação do Processo Unificado, pois foi definido pelos próprios criadores do UP. Mas existem outras implementações importantes, a maioria das quais varia a maneira como as diferentes disciplinas são definidas e organizadas. As implementações também podem variar a importância que dão a diferentes artefatos. As implementações ágeis do UP, por exemplo, simplificam as *disciplinas* e reduzem o número de artefatos esperados.

O número de implementações do UP é bastante grande, já que cada empresa pode implementar o modelo de acordo com suas necessidades. A Wikipédia[61] enumera várias implementações importantes, algumas das quais são citadas adiante neste capítulo.

5.1 Caracterização do Processo Unificado

O UP é mais do que um processo, é um *framework* extensível para concepção de processos, podendo ser adaptado às características específicas de diferentes empresas e projetos. As principais características do UP são:

a) É dirigido por casos de uso.
b) É centrado na arquitetura.
c) É iterativo e incremental.
d) É focado em riscos.

Essas características serão discutidas nas próximas subseções.

5.1.1 Dirigido por Casos de Uso

Um *caso de uso* é um processo compreendido do ponto de vista do usuário. Para o UP o conjunto de casos de uso deve definir e esgotar *toda a funcionalidade possível* do sistema. Wazlawick (2011, p. 35) apresenta várias técnicas para identificar casos de uso de boa granularidade.

Os casos de uso são úteis para várias atividades relacionadas ao desenvolvimento de um sistema. Entre elas:

a) *Definição e validação da arquitetura do sistema*. Classes e atributos usualmente são obtidos a partir dos textos dos casos de uso expandidos.
b) *Criação dos casos de teste*. Os casos de uso podem ser vistos como um roteiro para teste de sistema e de aceitação (Seção 13.2), onde as funcionalidades são testadas do ponto de vista do cliente.
c) *Planejamento das iterações*. Os casos de uso são priorizados e o esforço para desenvolvê-los é estimado de forma que cada iteração desenvolva certo numero deles (Seção 6.4).
d) *Base para a documentação do usuário*. Os casos de uso são descrições de fluxos normais de operação de um sistema bem como de fluxos alternativos representando o tratamento de possíveis exceções nos fluxos normais. Estas descrições são uma excelente base para iniciar o manual de operação do sistema, pois todas as funcionalidades possíveis estarão descritas aí de forma estruturada e completa.

Porém, a aplicação mais fundamendal do caso de uso no desenvolvimento de sistemas é a incorporação dos requisitos funcionais de uma forma organizada. Cada passo dos fluxos principal e alternativos de um caso de uso corresponde a uma função do sistema. Requisitos não funcionais podem ser anotados juntamente com os casos de uso ou seus passos, e requisitos suplementares são anotados em um documento a parte.

[61] en.wikipedia.org/wiki/IBM_Rational_Unified_Process (Acesso em 23/03/2010).

5.1.2 Centrado na Arquitetura

O UP preconiza que uma sólida arquitetura de sistema deve ser desenvolvida. As funcionalidades aprendidas com a elaboração dos diversos casos de uso devem ser integradas a esta arquitetura de forma incremental.

A arquitetura inicialmente pode ser compreendida como o conjunto de classes, possivelmente agrupadas em componentes, que realizam as operações definidas pelo sistema. A arquitetura é uma estrutura que provê funcionalidades. Os casos de uso são a descrição dos processos que realizam ou usam essas funcionalidades. A arquitetura então existe para que as funcionalidades sejam possíveis.

A arquitetura é basicamente um modelo que define a estrutura da informação, suas possíveis operações e sua organização em componentes ou mesmo em camadas.

Segundo UP, a cada ciclo iterativo deve-se incorporar à arquitetura existente as funcionalidades aprendidas com a análise de cada um dos casos de uso abordados no ciclo. Assim, fazendo-se a priorização dos casos de uso a partir dos mais críticos ou complexos para os mais triviais e simples, desenvolve-se em um primeiro momento todos os elementos de maior risco para a arquitetura, não ficando muitas surpresas para depois.

5.1.3 Iterativo e Incremental

Assim como nos métodos ágeis, o UP preconiza o desenvolvimento baseado em ciclos iterativos de duração fixa, onde a cada iteração a equipe incorpora à arquitetura as funcionalidades necessárias para realizar os casos de uso abordados no ciclo.

Cada ciclo iterativo produz um incremento no *design* do sistema, seja produzindo mais conhecimento sobre seus requisitos e arquitetura, seja produzindo código executável. Espera-se que em cada iteração todas as disciplinas previstas sejam executadas com maior ou menor intensidade (ver Figura 5-1). Então, assim como nos métodos ágeis, cada ciclo iterativo vai implicar em executar todas as atividades usuais de desenvolvimento de software.

A integração contínua reduz riscos, facilita os testes e melhora o aprendizado da equipe sobre o sistema, especialmente nos primeiros momentos, quando decisões críticas precisam ser tomadas com relativamente pouco conhecimento sobre o sistema em si.

Usualmente a fase de concepção, por ser curta, é executada em um único ciclo. Já as fases de elaboração, construção e transição, podem ser executadas a partir de uma série de ciclos iterativos. Porém, no caso de projetos muito grandes, mesmo a fase de concepção pode ser subdividida em ciclos nos quais se exploram diferentes características de um sistema.

5.1.4 Focado em Riscos

Em função das priorizações dos casos de uso mais críticos nos primeiros ciclos iterativos, pode-se dizer que o UP é focado em riscos, pois se estes casos de uso são os que apresentam maior risco de desenvolvimento, então devem ser tratados o quanto antes para que os riscos sejam resolvidos enquanto o custo de tratá-los ainda é baixo e o tempo disponível para acomodar surpresas ainda é relativamente grande.

Este tipo de abordagem (tratar primeiro os problemas mais difíceis) tem sido um valor incorporado em vários modelos de desenvolvimento modernos. Os requisitos ou casos de uso de maior risco são os mais imprevisíveis. Assim, estudá-los primeiramente, além de garantir um maior aprendizado sobre o sistema e as decisões arquiteturais mais importantes, também vai fazer com que riscos positivos ou negativos sejam dominados o mais cedo possível (um risco positivo é, por exemplo, o sistema ser mais simples do que inicialmente imaginado).

5.2 Fases do Processo Unificado

O Processo Unificado divide-se em quatro grandes fases:

a) Concepção (*inception*). Trata-se da elaboração de uma visão em abrangência do sistema. São levantados os principais requisitos, um modelo conceitual preliminar é construído, bem como são identificados os casos de uso de alto nível (Wazlawick, 2011, p. 35) que implementam a funcionalidade requerida pelo cliente. Nesta fase se calcula o esforço de desenvolvimento dos casos de uso e se constrói o plano de desenvolvimento composto por um conjunto de ciclos iterativos nos quais são acomodados os casos de uso. Pode haver alguma implementação e teste, caso seja necessário elaborar protótipos para redução de risco nesta fase.
b) Elaboração (*elaboration*). Nesta fase as iterações têm como objetivo predominantemente o detalhamento da análise, expandindo os casos de uso, para obter sua descrição detalhada e situações excepcionais (fluxos alternativos). O modelo conceitual preliminar será transformado em um modelo definitivo, cada vez mais refinado e sobre o qual são aplicados padrões de análise e uma descrição funcional poderá ser feita, bem como o *design* lógico e físico do sistema. Código também será gerado e testado, mas essas atividades não são as que vão ocupar a maior parte do ciclo iterativo, pois haverá proporcionalmente mais carga de trabalho em análise e *design* do que em codificação e teste.
c) Construção (*construction*). A fase de construção possui iterações onde os casos de uso mais complexos já foram tratados e a arquitetura estabilizada. Assim, as atividades de suas iterações consistem predominantemente na geração de código e testes do sistema. Com a automatização da geração de código, bem como com a introdução de modelos de desenvolvimento dirigidos a teste, pressupõe-se que um bom *design* possa dar origem rapidamente a código de alta qualidade.
d) Transição (*deployment*). A fase de transição consiste na implantação do sistema no ambiente final, com a realização de testes de operação. Também é feita a transferência de dados de possíveis sistemas antigos para o novo sistema e o treinamento de usuários, bem como outras adaptações necessárias que variam de caso a caso. Pode haver ainda alguma revisão de requisitos e geração de código nesta fase, mas não de forma significativa.

Apesar de que as fases do Processo Unificado têm diferentes ênfases, espera-se que cada ciclo iterativo tome um conjunto de casos de uso e os desenvolva desde os requisitos até a implementação e integração de código final. Acontece que na fase de elaboração a equipe necessariamente trabalhará mais tempo em questões de análise e projeto do que em implementação e teste. Mas na fase de construção, os requisitos já terão sido em grande parte desvendados e o esforço recairá mais nas atividades de programação.

Uma das características do UP também é o fato de que a cada fase um macro-objetivo (*milestone*) é atingido. Ao final da fase de concepção o objetivo é ter entendido o escopo do projeto e planejado seu desenvolvimento. Ao final de fase de elaboração os requisitos devem ter sido extensivamente compreendidos e uma arquitetura estável definida. Ao final da fase de construção o sistema deve estar programado e testado. Ao final da fase de transição o software deve estar instalado e sendo usado pelos usuários finais (West, 2002).

5.2.1 Concepção

A fase de concepção no Processo Unificado não deve ser muito longa. De duas semanas a dois meses é o que se recomenda dependendo da dimensão relativa do projeto.

Nesta etapa são analisados os requisitos de projeto da melhor forma possível. Eles são analisados em abrangência, não em profundidade. É importante que o analista perceba claramente a diferença entre as necessidades lógicas e tecnológicas do cliente e os possíveis projetos de implementação que ele poderia fazer. A ideia é que o analista não polua a descrição dos requisitos com possibilidades tecnológicas de implementação que não foram expressamente requisitadas pelo cliente.

O UP espera que nesta fase sejam estabelecidos também os casos de uso. Há basicamente três formas de agir em relação a isso:

a) Obter casos de uso a partir de uma organização dos requisitos funcionais, onde cada grupo de requisitos poderá dar origem a um ou mais casos de uso.
b) Inicialmente proceder a uma análise de cenários, e posteriormente extrair deles os casos de uso, ou seja, os requisitos.
c) Trabalhar apenas com casos de uso, sendo eles a única expressão dos requisitos, e não havendo um outro documento de requisitos, exceto para os requisitos suplementares (Wazlawick, 2011, p. 26).

De posse de um conjunto significativo de casos de uso, a equipe deve proceder como nas fases iniciais de planejamento de *Scrum* ou XP, priorizando os casos de uso mais complexos e críticos em detrimento dos mais simples e triviais. Isso pode ser feito, inicialmente por uma análise bem simples:

a) Descubra os casos de uso que são meros relatórios, ou seja, que não alteram a informação armazenada, e classifique estes casos de uso como os mais simples de todos.
b) Verifique se existem casos de uso que seguem algum padrão conhecido, como CRUD (Wazlawick, 2011, p. 65), mestre-detalhe, etc. Estes casos de uso serão de média complexidade, pois embora possuam um comportamento bem conhecido e padronizado, podem esconder algumas surpresas nas suas regras de negócio, ou seja, inclusões, alterações e exclusões possivelmente só poderão ser feitas mediante determinadas condições (ver também Seção 7.5.9).
c) Os demais casos de uso serão considerados os mais complexos. É recomendável que estes ainda sejam organizados do mais importante para o menos importante em relação ao negócio da empresa. O analista deve se perguntar, qual dos processos é o mais crítico para o sucesso da empresa. Em função disso, fará a ordenação. Por

exemplo, uma venda possivelmente será mais importante do que uma compra, uma compra mais importante do que uma reserva, uma reserva mais importante do que um cancelamento de reserva, etc.

Na fase de concepção a equipe deve produzir estimativas de esforço para casos de uso, que serão explicadas melhor na Seção 7.5. A partir desse cálculo, deve-se fazer um planejamento de longo prazo, procurando acomodar os casos de uso, de acordo com sua prioridade nos diferentes ciclos ao longo do processo de desenvolvimento. Mas esse planejamento não deve ser muito detalhado.

Apenas o primeiro ciclo deve ter um planejamento mais detalhado, com suas atividades definidas de acordo com o processo em uso e os tempos, responsáveis e recursos para execução de cada atividade bem definidos. Os demais ciclos terão seu planejamento sendo detalhado apenas um pouco antes de iniciar.

A fase de concepção envolve também o estudo de viabilidade, pois ao seu final a equipe deve decidir se é viável prosseguir com o projeto, analisadas as questões tecnológicas, de orçamento e de cronograma.

O marco final (*milestone*) da fase de concepção é conhecido como *LCO*, ou *Lifecicle Objective Milestone* (marco do ciclo de vida).

5.2.2 Elaboração

A fase de elaboração consiste em um detalhamento da análise e realização do projeto para o sistema como um todo. A elaboração ocorre em ciclos, com partes de *design* sendo desenvolvidas e integradas ao longo de cada ciclo. Os principais objetivos desta fase, segundo Ambler e Constantine (2000) são:

a) Produzir uma arquitetura executável confiável para o sistema.
b) Desenvolver o modelo de requisitos até completar pelo menos 80% dele.
c) Desenvolver um projeto geral para a fase de construção.
d) Garantir que as ferramentas críticas, processos, padrões e regras estão disponíveis para a fase de construção.
e) Entender e eliminar os riscos de alta prioridade do projeto.

Esta fase permite analisar o domínio do problema de forma mais refinada, permitindo definir uma arquitetura mais adequada e sólida. Além disso, a priorização dos casos de uso mais complexos nesta fase permitirá eliminar ou mitigar os elementos de projeto que apresentam o maior risco.

Assim, embora o Processo Unificado também trabalhe com a perspectiva de acomodação de mudanças, ele procura minimizar seu impacto, mitigando riscos e elaborando uma arquitetura o mais próxima possível do necessário para que as funcionalidades requeridas possam ser desenvolvidas.

A fase de elaboração é caracterizada pela exploração dos casos de uso mais complexos. Estes casos de uso são os que vão necessitar de relativamente mais trabalho de análise do que de implementação, pois será necessário entender e modelar seu funcionamento. À medida que

estes casos de uso são estudados e desenvolvidos a arquitetura do sistema vai se formando. Espera-se que a arquitetura esteja estável no momento em que se passar a considerar apenas os casos de uso padronizados, como CRUD e relatórios. Estes casos de uso não devem impactar a arquitetura.

O marco final (*milestone*) da fase de elaboração é conhecido como *LCA*, ou *Lifecicle Architecture Milestone* (marco da arquitetura).

5.2.3 Construção

Na fase de construção, um produto completo e usável deve estar desenvolvido, testado e adequado para uso pelo usuário final. A fase de construção também é realizada em ciclos iterativos. O projeto é desenvolvido também de forma incremental, com novas funcionalidades sendo adicionadas ao sistema a cada ciclo.

Segundo Ambler e Constantine (2000a), os principais objetivos da fase de construção são:

a) Descrever os requisitos que ainda faltam.
b) Dar substância ao *design* do sistema.
c) Garantir que o sistema atenda às necessidades dos usuários e que ele se encaixa no contexto geral da organização.
d) Completar o desenvolvimento dos componentes e testá-los, incluindo tanto o software quanto a sua documentação.
e) Minimizar os custos de desenvolvimento pela otimização dos recursos.
f) Obter qualidade adequada tão rápido quanto possível.
g) Desenvolver versões úteis do sistema.

A fase de construção se caracteriza pela exploração dos casos de uso de baixa e média complexidade, ou seja, os casos de uso padronizados que não vão impactar na arquitetura. Como estes casos de uso são padronizados, o esforço de análise e *design* será menor nesta fase, ficando a maior parte do trabalho concentrada na implementação e teste dos componentes da arquitetura dedicados a estes casos de uso.

O marco final (*milestone*) da fase de construção é conhecido como *IOC*, ou *Initial Operational Capability Milestone* (marco da capacidade operacional inicial).

5.2.4 Transição

A fase de transição consiste da colocação do sistema em uso no ambiente final. São necessários testes de aceitação e operação, treinamento de usuários e transição de dados a partir de sistemas antigos, que podem ser capturados automaticamente ou digitados.

Nesta fase também poderá haver a execução do sistema em paralelo com sistemas legados para verificar sua adequação.

Após a conclusão da fase de transição o sistema entra em evolução, ou seja, depois de aceito e colocado em operação no ambiente final, ele passa a receber atualizações periódicas de forma a corrigir possíveis erros ou implantar novas funcionalidades necessárias (ver Seção 14).

O marco final (*milestone*) da fase de transição é conhecido como *PR*, ou *Product Release Milestone* (marco da entrega do produto).

5.3 RUP – *Rational Unified Process*

A mais detalhada e mais antiga implementação do UP é conhecida como RUP (Rational Unified Process[62]), criada por Booch, Rumbaugh e Jacobson através da empresa Rational, desde 2003 subsidiária da IBM. A versão RUP é tão importante que algumas vezes é confundida ou considerada sinônimo de UP.

Ainda antes de pertencer à IBM, a empresa Rational em 1997 adquiriu várias outras empresas: Verdix, Objectory, Requisite, SQA, Performance Awareness e Pure-Atria, e a partir da experiência acumulada destas empresas, estabeleceu algumas melhores práticas, que seriam a base filosófica para o novo modelo de processo mais tarde conhecido como RUP (Kruchten, 2003):

a) *Desenvolver iterativamente tendo o risco como principal fator de determinação de iterações*. É preferível conhecer todos os requisitos antes de iniciar o desenvolvimento propriamente dito, mas isso normalmente não é viável, de forma que o desenvolvimento iterativo orientado a redução de risco é bastante adequado.
b) *Gerenciar requisitos*. Deve-se manter controle sobre o grau de incorporação dos requisitos do cliente ao produto.
c) *Empregar uma arquitetura baseada em componentes*. Quebrar um sistema complexo em componentes não só é necessário, como inevitável. A organização permite diminuir o acoplamento, o que possibilita testes mais confiáveis e maior possibilidade de reuso.
d) *Modelar software e forma visual com diagramas*. UML é indicada como padrão de modelagem de diagramas.
e) *Verificar a qualidade de forma contínua*. O processo deve ser tão orientado a testes quanto possível. Se for o caso, utiliza-se técnicas de desenvolvimento orientados a testes, como TDD, ou *Test-Driven Development* (Beck, 2003).
f) *Controlar as mudanças*. A integração deve ser contínua e gerenciada adequadamente com o uso de um sistema de gerenciamento de configuração (Capítulo 10).

RUP é um produto que entre outras coisas inclui uma base de dados com *hiperlinks* incluindo vários artefatos e *templates* necessários para bem usar o modelo. Uma descrição em português do processo pode ser encontrada na Internet[63].

5.3.1 Os Blocos de Construção do RUP

RUP é baseado em um conjunto de elementos básicos (*building blocks*) identificados da seguinte forma:

a) *Quem*. Um *papel* (Seção 2.3.2) define um conjunto de habilidades necessário para realizar determinadas atividades.
b) *O quê*. O *produto do trabalho* (*work product*) define algo produzido por alguma atividade ou atividade, como diagramas, relatórios ou código funcionando, ou seja, os *artefatos* (Seção 2.3.1).

[62] www.rational.com/rup/
[63] www.wthreex.com/rup/portugues/index.htm

c) *Como*. Uma *atividade* (Seção 2.3) descreve uma unidade de trabalho atribuída a um papel que produz um determinado conjunto de artefatos.
d) *Quando*. Os *workflows* são grafos que definem as dependências entre as diferentes atividades.

As disciplinas RUP, como requisitos, análise e *design*, implementação etc., são os containers para os quatro elementos mencionados acima, ou seja, cada disciplina é definida a partir de um ou mais *workflows*, que definem dependências entre atividades, as quais são realizadas por pessoas representando papeis e produzindo ou transformando artefatos bem definidos.

5.3.1.1 Papeis

Segundo Kruchten (2003), um *papel* define um conjunto de comportamentos e responsabilidades para uma pessoa ou grupo de pessoas trabalhando como uma equipe. O comportamento é expresso através de um conjunto de atividades que este papel exerce, as quais devem ser coesas. Essas responsabilidades e atividades também são expressas em função dos artefatos que estes papeis criam ou alteram.

Papeis não são pessoas específicas nem cargos. Uma mesma pessoa pode exercer vários papeis em diferentes momentos num mesmo dia no mesmo projeto.

Os papeis são organizados em cinco categorias principais:

a) Papeis de analista.
b) Papeis de desenvolvedor.
c) Papeis de testador.
d) Papeis de gerente.
e) Outros papeis.

As subseções seguintes apresentarão alguns dos papeis mais comuns incluídos nestes cinco grupos.

5.3.1.1.1 Papeis de Analista

Papeis de analista estão relacionados principalmente ao contato com o futuro usuário ou cliente do sistema. As pessoas que representam estes papeis devem ser capazes de entender quais são as necessidades do sistema e criar descrições para estas necessidades que sejam compreensíveis para os *designers*, desenvolvedores e testadores. Alem de criar estas descrições, eles devem garantir sua qualidade, especialmente sua adequação às reais necessidades e conformidade com normas e padrões estabelecidos.

Os papeis de analista no RUP subdividem-se nos seguintes tipos:

a) *Analista de sistemas*. Ele lidera e coordena a análise de requisitos, definindo o escopo do sistema e seus casos de uso. O analista de sistemas deve ser uma pessoa com boa capacidade de comunicação e negociação, além de ser necessário que ele tenha conhecimento de negócios e tecnologia.
b) *Designer de negócio*. Ele faz a modelagem de negócio, usualmente utilizando diagramas de atividade, máquina de estados ou ainda BPMN. Ele atuará nas fases mais iniciais da análise com a produção do modelo de negócio ou ciclo de negócio, que vai

colocar o sistema a ser desenvolvido dentro da perspectiva mais ampla da organização. A partir das atividades de negócio identificadas os casos de uso serão posteriormente identificados.

c) *Revisor do modelo de negócios*. Ele deve revisar o modelo de negócio verificando se é consistente, coerente e não ambíguo. Deve ter conhecimento profundo do negócio e, se possível da tecnologia a ser usada para sua implementação (o papel poderá ser desempenhado por duas pessoas caso não se consiga uma única pessoa com estas características).
d) *Analista do processo de negócio*. Ele lidera e coordena a modelagem da organização como casos de uso de negócio (Jacobson, 1994). Esta atividade pode ser dispensada, especialmente se não houver necessidade de uma reengenharia de negócio a partir do projeto sendo desenvolvido.
e) *Especificador de requisitos*. Ele é responsável pelos requisitos que usualmente são representados como casos de uso, inicialmente de alto nível, e depois expandidos na sua forma essencial. Além dos requisitos funcionais, representados nos casos de uso, o especificador de requisitos deve também especificar os requisitos suplementares, ou seja, os aspectos não funcionais gerais do sistema.
f) *Revisor de requisitos*. Ele é responsável pela revisão minuciosa dos casos de uso e especificações suplementares verificando se são completos, coerentes e não ambíguos. Além disso, deve garantir que o documento de casos de uso esteja escrito em conformidade com os padrões adotados na empresa.
g) *Designer de interface com usuário*. Ele coordena as atividades de prototipação de interfaces e de *design* de interface com usuário. Ele é o responsável pelos requisitos de interface. Ele não implementa a interface, ficando sua responsabilidade restrita ao *design* visual e de usabilidade.

O RUP ainda considera que o *analista de teste* é um papel de analista. Porém, uma vez que existem papeis relacionados à disciplina de teste, pode-se considerar que sua atividade é mais fortemente relacionada a estes papeis e não aos de analista de sistemas. Dessa forma, ele foi classificado junto aos papeis de teste neste livro.

5.3.1.1.2 Papeis de Desenvolvedor

Os desenvolvedores são aqueles que efetivamente transformam os requisitos em produto. Os papeis relacionados ao desenvolvedor no RUP são os seguintes:

a) *Implementador.* Ele é responsável pela produção e teste de unidade do código fonte.
b) *Revisor de código.* Ele deve ser responsável por garantir a qualidade do código fonte implementado, verificando se este segue padrões estabelecidos e boas práticas de programação. O revisor deve ser um profundo conhecedor da linguagem de programação utilizada.
c) *Integrador.* Quando os implementadores liberam componentes já testados cabe ao integrador incluí-los em uma versão operacional do sistema para gerar uma nova versão (ou *build*). O integrador deverá elaborar e realizar os testes de integração caso essa atividade não tenha sido atribuída ao analista de teste.

d) *Arquiteto de software.* O arquiteto é responsável pelo *design* das camadas e/ou partições do sistema, ou seja, da sua estrutura em nível mais alto, incluindo o *design* de pacotes, componentes e sua distribuição em diferentes nodos de processamento.
e) *Revisor de arquitetura.* O revisor de arquitetura deve avaliar se o trabalho do arquiteto de software efetivamente leva a uma arquitetura sólida e estável. Visto que a arquitetura costuma conter as decisões mais cruciais para o sucesso de um sistema, este papel é importante para garantir que as decisões do arquiteto sejam as mais efetivas possíveis.
f) *Designer.* O *design*er toma decisões sobre o *design* das classes, seus atributos e associações, bem como de métodos a serem implementados.
g) *Revisor de design.* Ele deve revisar o *design* verificando se as melhores práticas e padrões foram adotadas.
h) *Designer de banco de dados.* Ele é responsável pelo *design* das tabelas e códigos associados ao banco de dados.
i) *Designer de cápsula.* Este papel só existe em projetos de sistemas de tempo real. Ele é responsável por assegurar que o sistema responda prontamente a eventos de acordo com os requisitos de tempo real.

O RUP inclui aqui também o papel de *design*er de teste, o qual, neste livro, foi agrupado aos papeis de testador.

5.3.1.1.3 Papeis de Testador

Embora no RUP originalmente exista apenas um papel específico para o testador, foram agrupados aqui também os papeis relacionados de analista, *design*er e gerente de teste.

a) *Designer de teste.* Ele é responsável por definir as técnicas e estratégias de teste a serem adotadas. Não deve ser confundido com o *analista* de teste que elabora os testes de sistema; o *design*er de teste define a estratégia de teste, mas não os testes reais.
b) *Analista de teste*. Ele é responsável pelo projeto e elaboração dos casos de teste a serem aplicados ao sistema. Usualmente o teste de unidade é feito pelo próprio programador e o teste de integração pelo integrador, sendo que o analista de teste costuma atuar principalmente no teste de sistema.
c) *Testador.* O testador é responsável pela realização efetiva dos testes. Ele verifica a melhor abordagem a ser utilizada de acordo com as estratégias definidas pelo *design*er de teste e analista de teste e implementa o teste além de executá-lo para verificar se o componente ou o sistema passam no teste. Ele deve registrar os resultados dos testes e no caso de componentes que não passem nos testes deve informar os respectivos responsáveis para providências de correção.
d) *Gerente de teste*. Ele coordena toda a atividade de teste do sistema.

5.3.1.1.4 Papeis de Gerente

Os papeis de gerente estão relacionados especialmente às atividades de planejamento, controle e organização do projeto. Os papeis de gerente no RUP são os seguintes:

a) *Engenheiro de processo*. Ele é o responsável pela aplicação do processo de desenvolvimento, devendo configurar e ajustar o processo de acordo com as necessidades da equipe e dos projetos.
b) *Gerente de projeto*. Ele é o responsável por um ou mais projetos específicos, devendo planejar as atividades e alocar os recursos físicos e humanos, bem como acompanhar o projeto garantindo que prazos e orçamentos sejam cumpridos e tomando decisões de correção de rumo quando necessário.
c) *Gerente de controle de mudança*. Ele é o responsável pelo controle das requisições de mudança, seja do cliente, seja da própria equipe de desenvolvimento, e acompanha o atendimento a elas.
d) *Gerente de configuração*. Ele planeja e disponibiliza o ambiente para que os desenvolvedores e integradores possam realizar suas atividades. Ele garante que todos tenham acesso aos artefatos necessários.
e) *Gerente de implantação*. Ele planeja e acompanha a implantação do sistema junto aos clientes e usuários.
f) *Revisor do projeto*. Ele é responsável pela revisão dos planos e avaliações do projeto ao longo do seu desenvolvimento.

Além destes papeis conta-se ainda o gerente de teste, agrupado junto aos papeis de testador.

5.3.1.1.5 Outros Papeis

Outros papeis do RUP, não classificados nos tipos anteriores, são definidos:

a) *Interessados* (ou *envolvidos*). São todos aqueles afetados pelo sistema ou seu desenvolvimento.
b) *Desenvolvedor de curso.* É o responsável pela criação de material de treinamento para usuários.
c) *Artista gráfico.* É o responsável pela criação da arte final do produto, de sua embalagem e de outros artefatos correlatos.
d) *Especialista em ferramentas.* Ele dá suporte aos desenvolvedores pela seleção, instalação e treinamento no uso das ferramentas de apoio ao desenvolvimento e gerenciamento dos projetos.
e) *Administrador do sistema.* Ele mantém o ambiente de desenvolvimento, cuidando do hardware, versões de software, rede, etc.
f) *Redator técnico.* Ele é o responsável pela redação final tanto dos manuais quanto de partes do sistema orientadas a texto.

Como o RUP é adaptável, novos papeis podem ser necessários e adicionados a um projeto ou processo específico de empresa.

5.3.1.2 Atividades

Atividades são unidades de trabalho executadas por um indivíduo que exerce um papel dentro do processo (Kruchten, 2003). Toda atividade deve produzir um resultado palpável em termos de criação ou alteração consistente de artefatos como modelos, elementos (classes, atores, código etc.) ou planos. Em RUP, toda atividade é atribuída a um papel específico.

Assim como nos métodos ágeis, uma atividade não deve ser muito curta (poucas horas) nem muito longa (vários dias). Sugere-se pensar em atividades que possam ser realizadas em períodos de 1 a 3 dias, porque esta duração facilita o acompanhamento.

A mesma atividade pode ser repetida sobre o mesmo artefato, o que inclusive é normal ao longo dos ciclos iterativos, mas não é repetida necessariamente pela mesma pessoa, embora possa ser o mesmo papel. Por exemplo, a arquitetura do sistema pode ser refinada e revisada diversas vezes ao longo dos ciclos iterativos

5.3.1.2.1 Passos de Atividades

Atividades são normalmente descritas em termos de passos. Kruchten (2003) identifica três tipos de passos:

a) Passos de *pensamento*: a pessoa exercendo o papel compreende a natureza da atividade, obtém e examina os artefatos de entrada e formula a saída.
b) Passos de *realização*: a pessoa exercendo o papel cria ou atualiza artefatos.
c) Passos de *revisão*: a pessoa exercendo o papel inspeciona os resultados em função de algum critério.

Assim como passos de casos de uso expandidos (Wazlawick, 2011, p. 43), passos de atividade não são necessariamente executados sempre. Deve existir um conjunto de passos que consista no fluxo principal da atividade, sendo que, neste caso, todos são obrigatórios, e podem existir passos que pertencem a fluxos alternativos, sejam variantes ou tratadores de exceção.

5.3.1.3 Artefatos

Um artefato pode ser um diagrama, modelo, elemento de modelo, texto, código fonte, código executável etc., ou seja, qualquer tipo de produto criado ao longo do processo de desenvolvimento de software.

Assim, artefatos podem ser por sua vez compostos por outros artefatos, como por exemplo, uma classe contida no modelo conceitual. Como os artefatos mudam com o passar do tempo, pode ser interessante submetê-los a um controle de versões. Porém, o controle de versão normalmente é exercido no artefato de mais alto nível e não em elementos individuais.

Os artefatos são as entradas para as atividades e também são suas saídas. Normalmente o objetivo de uma atividade é criar artefatos ou promover alterações consistentes e verificáveis em artefatos.

Artefatos de saída ainda podem ser classificados em *entregas*, que são artefatos entregues ao cliente.

Artefatos (especialmente os de texto) também podem ser definidos por *templates*, isto é, modelos de documentos que dão uma forma geral ao artefato.

O artefato, do ponto de vista do Processo Unificado não deve ser entendido simplesmente como um documento acabado no estilo dos relatórios de revisão final dos marcos do Modelo Cascata. Os artefatos no UP, e consequentemente no RUP são documentos dinâmicos que podem ser alterados a qualquer momento e, por isso, são mantidos sob controle de versão.

O RUP não encoraja a produção de papel. Sugere-se que todos os artefatos sejam mantidos e gerenciados por uma ferramenta adequada, de forma que sua localização e versão corrente sempre sejam conhecidas e acessíveis por quem de direito.

Ao contrário dos métodos ágeis, que incentivam a posse coletiva, RUP sugere que cada artefato tenha um dono ou responsável. Embora outros tenham acesso, apenas o responsável pode alterar um artefato, ou permitir a outros o direito de fazer alterações.

5.3.1.3.1 Relatórios

Um relatório é uma visão gerada para um artefato ou conjunto de artefatos com o propósito de servir para revisão.

Ao contrário dos artefatos, os relatórios não são sujeitos a controle de versão, porque devem poder ser gerados a qualquer tempo (de preferência automaticamente) de acordo com a versão atual dos artefatos.

5.3.1.3.2 Grupos de Artefatos

Os artefatos no RUP estão organizados em grupos de acordo com as disciplinas (Seção 5.3.2) às quais se relacionam. Basicamente são os seguintes tipos:

a) *Artefatos de gerenciamento*, que são todos os artefatos relacionados ao planejamento e execução do projeto em si. Exemplos: plano de desenvolvimento de software, caso de negócio, plano de iteração, avaliação de iteração, avaliação de *status*, plano de resolução de problemas, plano de gerenciamento de riscos, lista de riscos, ordem de trabalho, plano de aceitação do produto, plano de métricas, plano de garantia de qualidade, lista de problemas, registro de revisão e métricas de projeto.
b) *Artefatos de gerenciamento de configuração e mudança*, que apresentam informações relacionadas à disciplina de gerenciamento de configuração e mudança. Exemplos: registro de auditoria de configuração, plano de gerenciamento de configuração, repositório do projeto, espaço de trabalho de desenvolvimento, espaço de trabalho de integração e solicitação de mudança.
c) *Artefatos de ambiente*, que são artefatos usados ao longo do processo de desenvolvimento de forma a garantir a consistência dos demais artefatos. Exemplos: caso de desenvolvimento, avaliação da organização de desenvolvimento, *template*s específicos do projeto, guia de modelagem de negócio, guia de *design*, guia de programação, guia de modelagem de casos de uso, guia de interface de usuário, guia de teste, manual de guia de estilo, guia de ferramentas, ferramentas e infraestrutura de desenvolvimento.
d) *Artefatos de modelo de negócio*, os quais capturam e apresentam o modelo de negócio da empresa, ou seja, o contexto no qual o sistema deverá funcionar. Exemplos: glossário de negócios, regras de negócios, modelo de casos de uso de negócios, modelo de objetos de negócios, avaliação da organização alvo, visão do negócio, arquitetura de negócio e especificação suplementar de negócio.
e) *Artefatos de requisitos*, que são todos os artefatos relacionados à descrição do sistema a ser desenvolvido. Exemplos: plano de gerenciamento de requisitos, solicitações dos interessados, glossário, visão, modelo de casos de uso, especificações suplementares, atributos de requisitos, protótipos de interfaces e cenários de casos de uso.

f) *Artefatos de design*, que capturam e apresentam a solução tecnológica para atender aos requisitos dos interessados. Exemplos: prova de conceito arquitetural, arquitetura do software, modelo de análise, modelo de *design*, modelo de dados e modelo de implantação.
g) *Artefatos de implementação*, que capturam e realizam a solução tecnológica em um conjunto de artefatos executáveis. Exemplos: componentes, versão do sistema (*build*), plano de integração da versão, modelo de implementação.
h) *Artefatos de teste*, que são os planos e produtos das atividades de teste. Exemplos: plano de teste, sumário de avaliação de teste, *script* de teste, *log* de teste, lista de ideias de teste, casos de teste, modelo de análise de carga de trabalho, dados de teste, resultados de teste, arquitetura para automatização de teste, especificação da interface de teste, configuração do ambiente de teste, conjunto de testes, guia de teste, classe de teste e componente de teste.
i) *Artefatos de implantação*, que consistem na informação final entregue ao cliente, bem como outros artefatos ligados à transição do sistema. Exemplos: plano de implantação, lista de materiais, notas de *release*, produto, artefatos de instalação, materiais de treinamento, unidade de implantação, arte final do produto e material de suporte para o usuário.

Os artefatos não são produzidos de forma exclusiva em uma ou outra fase do RUP, já que este é um processo iterativo. O que se espera é que a maioria dos artefatos sejam iniciados na fase de concepção e refinados ao longo do projeto, sendo finalizados na fase de implantação. Porém, espera-se que os artefatos iniciais, como requisitos e *design* sejam finalizados mais cedo do que os artefatos finais como os de implementação e implantação.

5.3.1.4 Workflows

As atividades a serem executadas dentro de cada disciplina são definidas a partir de grafos direcionados chamados *workflows*. Um *workflow* define um conjunto de atividades e um conjunto de papéis responsáveis por cada atividade. Além disso, o *workflow* indica as dependências entre as diferentes atividades, ou seja, quais atividades dependem logicamente de quais outras atividades para poderem ser executadas. Essa dependência pode se dar em diferentes níveis de intensidade, porém, sendo algumas absolutamente necessárias e outras meramente sugeridas.

O RUP define três tipos de *workflow*:

a) *Workflow núcleo* (*core*), que define a forma geral de condução de uma dada disciplina.
b) *Workflow detalhe*, que apresenta um refinamento do *workflow* núcleo, indicando atividades em um nível mais detalhado, bem como artefatos de entrada e saída de cada atividade.
c) *Planos de iteração*, que consistem em uma instanciação do processo para uma iteração específica. Embora o RUP tenha uma descrição geral dos *workflows* para cada atividade, elas usualmente ocorrem de forma diferente em projetos diferentes e mesmo em ciclos diferentes dentro do mesmo projeto. Assim, o plano de iteração consiste em especificar atividades concretas a serem realizadas de fato dentro de uma iteração planejada.

5.3.1.5 Outros Elementos

Além dos elementos mencionados anteriormente, o RUP ainda apresenta outros elementos que auxiliam a aplicação do processo a um projeto. Esses elementos são:

a) *Procedimentos* (*guidelines*), as atividades são apresentadas nos *workflows* de forma mnemônica, mais para servir de lembrança do que para orientar. Mas os procedimentos mostram um detalhamento dessas atividades de forma que não apenas pessoas acostumadas ao processo, mas também novatos, possam saber o que devem fazer. No RUP, procedimentos basicamente são regras, recomendações e heurísticas sobre como executar a atividade. Elas devem focar também na descrição dos atributos de qualidade dos artefatos a serem feitos, como por exemplo, o que é um bom caso de uso, uma classe coesa, etc.
b) *Templates*, que são modelos ou protótipos de artefatos. Eles podem ser usados para criar os respectivos artefatos. Usualmente devem estar disponíveis na ferramenta usada para criar e gerenciar o artefato. Exemplos de *template*s são os modelos de documento do Microsoft Word e os próprios *workflows*, especificados em uma ferramenta apropriada, que podem ser usados como modelos para a criação dos planos de iteração.
c) *Mentores de ferramenta*, que consistem na descrição detalhada de como realizar uma atividade em uma ferramenta específica. Enquanto as descrições das atividades e até mesmo os procedimentos no RUP devem ser razoavelmente independentes de tecnologia, para que possam ser interpretados em diferentes ferramentas, os mentores devem ser preferencialmente construídos como um tutorial na própria ferramenta, mostrando como usá-la.

5.3.2 Disciplinas

Como foi visto, o UP preconiza que diferentes disciplinas sejam definidas. Cada disciplina descreve uma possível abordagem ao problema de desenvolver e gerenciar o desenvolvimento de um sistema. As disciplinas do UP englobam diferentes atividades e papeis relacionados por área de especialidade e suas implementaçõesdo variam o número e descrição das disicplinas.

Particularmente, o RUP conta com seis disciplinas de projeto e três disciplinas de suporte. As disciplinas de projeto são:

a) Modelagem de negócio.
b) Requisitos.
c) Análise e *design*.
d) Implementação.
e) Teste.
f) Implantação.

E as disciplinas de suporte são:

a) Gerenciamento de mudança e configuração.
b) Gerenciamento de projeto.
c) Ambiente.

Cada uma destas disciplinas aparece com uma ênfase diferente ao longo das fases e dos ciclos iterativos no RUP. A Figura 5-1 apresenta graficamente as ênfases ao longo do ciclo de vida RUP.

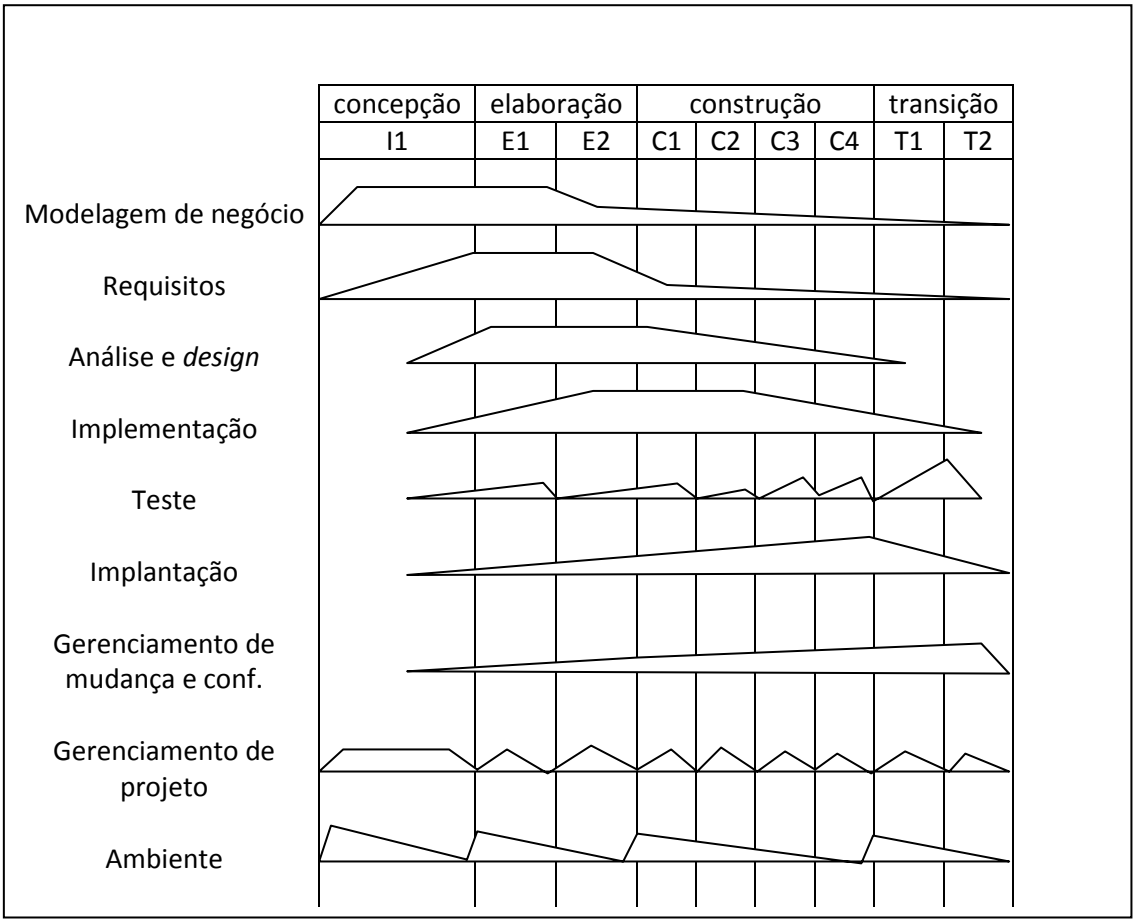

Figura 5-1: Diferentes ênfases de cada disciplina nas diferentes fases do RUP[64].

A Figura 5-1 é também chamada de "Arquitetura do RUP". Embora os nomes das disciplinas até lembrem as fases do modelo cascata, elas não são executadas de forma sequencial, como no modelo cascata visto que o RUP é um processo iterativo: suas quatro fases são sequenciais, mas as disciplinas são executadas de forma paralela ou sequencial, dependendo do caso, dentro de cada iteração individual.

5.3.2.1 Gerenciamento de Projeto

Segundo Kruchten (2003), gerenciar um projeto consiste em balancear objetivos que competem entre si, gerenciar riscos e superar restrições, com o objetivo de obter um produto que atenda às necessidades dos clientes (os que pagam pelo desenvolvimento) e dos usuários finais.

A disciplina de gerenciamento de projeto tem como objetivos indicar como planejar o projeto como um todo (Seção 6.4), como planejar cada iteração individual (Seção 6.5), como gerenciar os riscos do projeto (Capítulo 8) e como monitorar o progresso (Capítulo 9). Entretanto, RUP não trata os seguintes aspectos:

[64] Fonte: Ambler (2005). www.ambysoft.com/*downloads*/managersIntroToRUP.pdf

a) Gerenciamento de pessoas, incluindo contratação e treinamento.
b) Gerenciamento de orçamento.
c) Gerenciamento de contratos.

Tais aspectos são cobertos por uma implementação de RUP, a extensão EUP (Seção 5.6).

A disciplina de gerenciamento de projeto produz planos. O planejamento no RUP ocorre em dois níveis: plano de fase (ou projeto) e planos de iteração.

O *plano de fase* ou *plano de projeto* procura delinear o tempo total de desenvolvimento, duração e esforço das fases, número de ciclos e objetivos gerais de cada ciclo. Ele é controlado e mensurado pelos seguintes componentes:

a) O *plano de medição* (*measurement plan*), que estabelece as métricas a serem usadas para definir o andamento do projeto ao longo de sua execução (Seção 9.5).
b) O *plano de gerenciamento de riscos*, que detalha como os riscos serão tratados ao longo do projeto, criando atividades de mitigação e monitoramento de riscos e atribuindo responsabilidades quando necessário.
c) A *lista de riscos*, que é uma lista ordenada dos riscos conhecidos e ainda não resolvidos, em ordem decrescente de importância, juntamente com planos de mitigação e contingência quando for o caso.
d) O *plano de resolução de problemas*, que descreve como problemas identificados ao longo do projeto devem ser reportados, analisados e resolvidos (Seção 9.4.2).
e) O *plano de aceitação de produto*, que descreve como o produto será avaliado pelo cliente para verificar se satisfaz as necessidades. O plano deve incluir a identificação dos testes de aceitação.

O plano de fase não precisa ser longo (Seção 6.4). Usualmente poucas páginas são suficientes para a maioria dos projetos. Ele deve fazer referência ao documento de visão geral do sistema para esclarecimento sobre os objetivos do sistema e seu contexto.

Os *planos de iteração* (Seção 6.5) são planos mais detalhados do que o plano de fase. Cada plano de iteração inclui um cronograma de atividades atribuídas a responsáveis, com recursos alocados, prazos e dependências.

Usualmente há sempre um plano de iteração sendo seguido enquanto o da iteração seguinte já vai sendo delineado, para ser finalizado ao final do ciclo corrente.

Para definir um plano de iteração sugere-se observar os seguintes elementos:

a) O *plano da fase* corrente incluído no plano de projeto.
b) Informação sobre o *status do projeto* (por exemplo, atrasado, em dia, com grandes riscos em aberto, etc.).
c) Uma *lista de casos de uso* que pelo plano da fase devem ser finalizados na iteração corrente.
d) Uma *lista dos riscos* que devem ser tratados na iteração corrente.
e) Uma *lista das modificações* que devem ser incorporadas ao produto na iteração corrente (defeitos e erros encontrados ou mudanças de funcionalidade).

Todas as listas acima devem ser ordenadas do mais importante para o menos importante, de forma que se a iteração for muito mais trabalhosa do que esperado os itens menos importantes possam ser deixados para iterações posteriores.

O plano de iteração usualmente é apresentado como um diagrama Gantt (Seção 6.5.6) e deve indicar, para cada membro da equipe as atividades em que estará envolvido, seu início e final. Os aspectos de planejamento de projeto, medição e gerenciamento de riscos serão abordados respectivamente nos capítulos 6, 7 e 8.

De acordo com RUP, o principal papel da disciplina de gerenciamento de projeto é o de gerente de projeto, o qual é responsável pelos seguintes artefatos:

a) Caso de Negócio (*business case*).
b) Plano de desenvolvimento de software, incluindo:
    a. Plano de aceitação de produto.
    b. Plano de gerenciamento de riscos.
    c. Lista de riscos.
    d. Plano de resolução de problemas.
    e. Plano de medição.
c) Plano da iteração.
d) Avaliação da iteração.
e) Ordem de serviço.
f) Avaliação de *status*.
g) Medidas de projeto.

Além dele, outro papel importante é o de revisor de projeto, conforme será visto no detalhamento do *workflow* abaixo. O *workflow* da disciplina de gerenciamento de projeto é definido conforme a Figura 5-2.

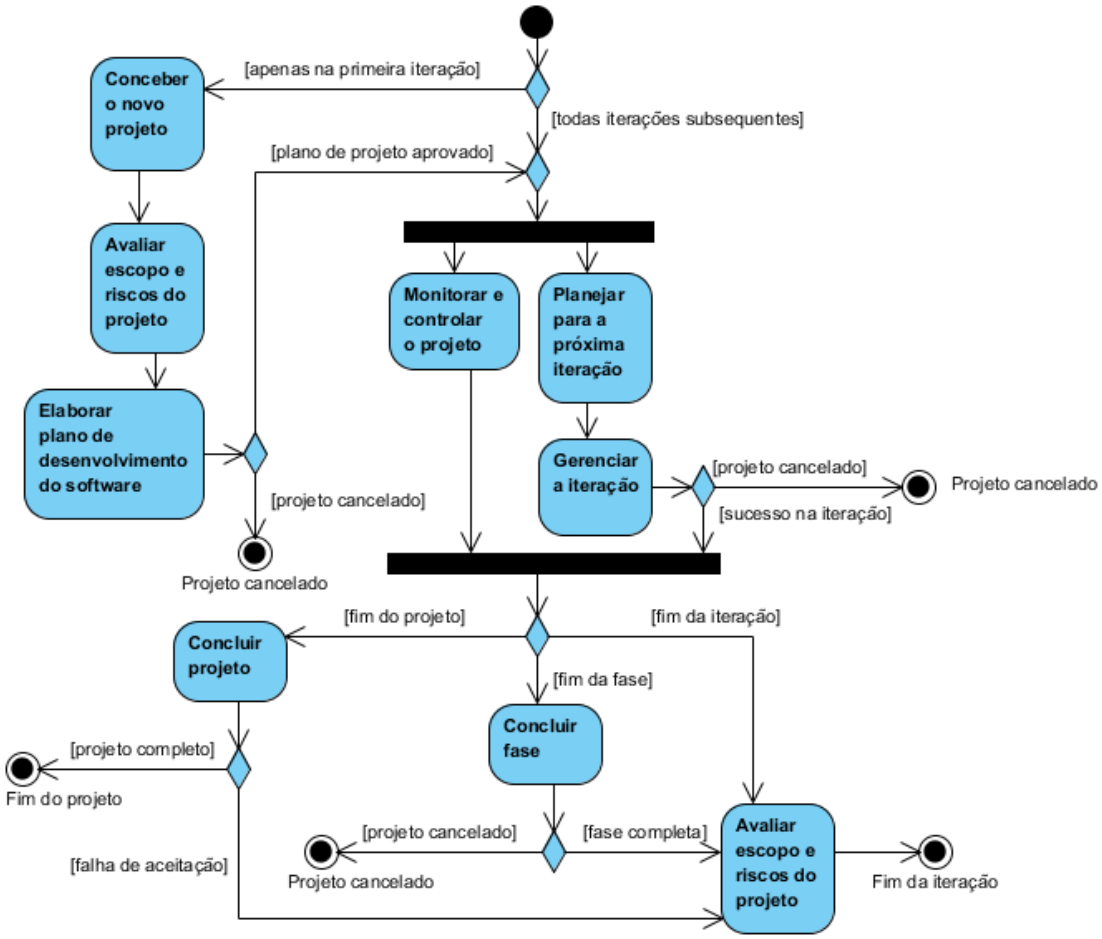

Figura 5-2: *Workflow* da disciplina gerenciamento de projeto[65].

Este *workflow* é decomposto em uma série de detalhamentos, ou seja, cada uma das atividades é descrita por atividades ainda mais elementares conforme a Tabela 5-1.

Tabela 5-1: Detalhamento do *workflow* para a disciplina de gerenciamento de projeto.

Atividade/Objetivo	Subatividades	Responsável
Conceber o novo projeto. O propósito desta atividade é levar o projeto do ponto em que ele é apenas uma ideia para um ponto onde sua viabilidade possa ser avaliada.	Identificar e avaliar riscos	Gerente de projeto
	Desenvolver o caso de negócio	
	Iniciar o projeto	
	Revisar a aprovação do projeto	Revisor de projeto
Avaliar escopo e riscos do projeto. Seu objetivo é reavaliar as características do projeto e os riscos associados. Isto é feito no início do projeto e ao final de cada iteração.	Identificar e avaliar riscos	Gerente de projeto
	Desenvolver o caso de negócio	
Elaborar plano de desenvolvimento do software. O objetivo desta atividade é desenvolver os componentes do plano de projeto e garantir que sejam formalmente revisados e validados. O maior esforço ocorre na primeira vez que a atividade é executada. Depois, a medida que os ciclos avançam o trabalho resume-se a revisar e atualizar os planos.	Desenvolver plano de medição	Gerente de projeto
	Desenvolver plano de gerenciamento de risco	
	Desenvolver plano de aceitação do produto	
	Desenvolver plano de resolução de problemas	
	Desenvolver organização de projeto e pessoal	
	Definir processos de controle e monitoramento	
	Planejar fases e iterações	
	Compilar o plano de desenvolvimento do software	
	Revisar planejamento do projeto	Revisor de projeto

[65] Adaptado de: Kruchten (2003).

Planejar para a próxima iteração. Seu objetivo é criar um plano da iteração detalhado para a iteração seguinte	Desenvolver plano de iteração	Gerente de projeto
	Desenvolver caso de negócio	
	Elaborar plano de desenvolvimento do software	
	Revisar plano de iteração	Revisor de projeto
Monitorar e controlar o projeto. Trata do dia a dia do gerente de projeto, incluindo o gerenciamento das mudanças solicitadas, monitoramento do progresso e dos riscos ativos, relatório de *status* para a Project Review Authority (PRA), e lidar com problemas de acordo com o plano de resolução de problemas.	Escalonar e atribuir atividades	Gerente de projeto
	Monitorar *status* do projeto	
	Tratar exceções e problemas	
	Relatar *status*	
	Revisão para a autoridade de revisão de projeto (PRA)	Revisor de projeto
Gerenciar a iteração. Detalha como iniciar, terminar e revisar uma iteração. Trata da obtenção dos recursos necessários, alocação de trabalho e avaliação dos resultados da iteração.	Obter pessoal	Gerente de projeto
	Iniciar iteração	
	Avaliar iteração	
	Revisão de critérios de avaliação de iteração	Revisor de projeto
	Revisão de aceitação de iteração	
Concluir fase. A fase é fechada quando o gerente de projeto garante que: todos os principais problemas das iterações anteriores foram resolvidos, o *status* de todos os artefatos é conhecido, os artefatos requeridos foram entregues aos interessados, todos eventuais problemas de implantação foram resolvidos e todas as questões financeiras foram resolvidas, caso o contrato corrente esteja terminando.	Preparar para conclusão de fase	Gerente de projeto
	Revisão de macro-objetivo (milestone) de fase	Revisor de projeto
Concluir projeto. O projeto é preparado para seu término e a posse dos ativos é repassada ao cliente.	Preparar para conclusão do projeto	Gerente de projeto
	Revisão de aceitação de projeto	Revisor de projeto

As subatividades de identificação e avaliação de riscos e desenvolvimento do caso de negócio aparecem repetidas nas atividades de concepção de novo projeto e avaliação de escopo e riscos do projeto, porque quando são executadas durante a avaliação de escopo e riscos elas o serão na forma de uma revisão, realizada após o início e aprovação do projeto, e também ao final das iterações.

A disciplina de planejamento de projeto, e especificamente as atividades de construção de plano de projeto e plano de iteração, serão detalhadas no Capítulo 6.

5.3.2.2 Modelagem de Negócio

A modelagem de negócio consiste em estudar e compreender a empresa e seus processos, visto que o sistema a ser desenvolvido não será um produto isolado na empresa, mas parte orgânica de seu funcionamento. Os objetivos desta disciplina são:

a) Entender a estrutura e a dinâmica da organização alvo na qual o software será utilizado.
b) Entender os problemas atuais na organização alvo e identificar potenciais melhorias que podem ser produzidas com o software.
c) Certificar que clientes, usuários e desenvolvedores tenham um conhecimento comum da organização alvo.
d) Derivar os requisitos para suportar estas melhorias.

Um dos pontos críticos para o sucesso de um projeto de desenvolvimento de software é o seu financiamento. Para que o cliente se mantenha interessado no desenvolvimento de um produto, ele deve estar sempre convencido de que o investimento no produto representará uma vantagem para ele e não uma despesa inútil. A compreensão do modelo de negócio é

fundamental para que a equipe de desenvolvimento permaneça sintonizada com esta compreensão e mantenha o cliente interessado.

Outro aspecto importante da modelagem de negócio é aproximar as equipes de engenharia de negócio e engenharia de software, de forma que os reais problemas e necessidades da organização sejam entendidos, analisados e solucionados com tecnologia da informação.

Esta disciplina, porém, nem sempre é necessária. Em alguns casos, quando o objetivo do projeto é simplesmente adicionar algumas funcionalidade em um sistema que não abrange um número significativo de pessoas, a modelagem de negócio pode ser desnecessária. Ela é mais importante quando mudanças significativas de comportamento serão introduzidas para um grupo de pessoas. Neste caso, a compreensão do negócio e as consequências da instalação de um novo sistema devem ser estudadas e conhecidas.

A modelagem de negócio pode ter ainda diferentes graus de ênfase em função das necessidades do projeto no qual a equipe vai se engajar. Kruchten (2003) identifica seis cenários de complexidade crescente em termos de necessidade de modelagem de negócio:

a) *Organograma*. Pode-se querer apenas construir um organograma da organização para saber quais são os setores e responsabilidades. Neste caso a modelagem de negócio usualmente acontece apenas na fase de concepção.
b) *Modelagem de domínio*. Se o objetivo for construir aplicações para gerenciar e apresentar informação, então usualmente faz-se a modelagem de negócio juntamente com a modelagem do domínio, ou seja, é um modelo de informação estático onde os *workflows* da empresa não são considerados. A modelagem de domínio usualmente é feita nas fases de concepção e elaboração.
c) *Uma empresa, vários sistemas*. Pode-se estar a ponto de desenvolver toda uma família de sistemas para uma empresa. Neste caso, a modelagem de negócio vai tratar não apenas de um sistema, mas de vários projetos e poderá inclusive ser tratada como um projeto a parte. Ela ajudará a descobrir os requisitos dos sistemas individuais, bem como determinar uma arquitetura comum para a família de sistemas.
d) *Modelo de negócio genérico*. Se o objetivo for a construção de um ou mais aplicativos que sirvam a um grupo de empresas, então a modelagem de negócio poderá ser útil para ajudar a alinhar as empresas a uma visão de negócio, ou, se isso não for possível, obter uma visão de negócio onde as especificidades das empresas seja visível.
e) *Novo negócio*. Se uma organização resolve iniciar um novo negócio e todo um conjunto de sistemas de informação deve ser desenvolvido para dar suporte a ele, então um esforço significativo de modelagem de negócio deve ser realizado. Neste caso, o objetivo da modelagem de negócio não é apenas encontrar requisitos, mas verificar a viabilidade efetiva do novo negócio. Assim, a modelagem de negócio nesta situação usualmente é um projeto a parte.
f) *Renovação*. Se uma organização resolve renovar completamente seu modo de fazer negócio, então a modelagem de negócio será um projeto a parte executado em várias etapas: visão do novo negócio, engenharia reversa do negócio existente, engenharia direta do novo negócio e instalação do novo negócio.

Os principais papeis RUP para modelagem de negócio são:

a) *Analista de processo de negócio*. Lidera e coordena a visão de negócio, definindo os atores externos e processos a serem modelados.
b) *Designer de negócio*. Detalha a visão de negócio da organização, determinando como os atores colaboram e usam objetos de negócio para realizar objetivos de negócio.
c) *Interessados (stakeholders)*. Representam várias partes da organização que podem prover informações ou revisão.
d) *Revisor de negócio*. Revisa os artefatos resultantes.

Os principais artefatos desta disciplina são:

a) *Documento de visão de negócio.* Define os objetivos e metas do esforço de modelagem de negócio.
b) *Modelo de casos de uso de negócio.* Um modelo das funções da empresa usado para identificar papéis e entregas (*deliverables*) da organização.
c) *Modelo de análise de negócio.* Um modelo de objetos que descreve a realização dos casos de uso de negócio.

Ao contrário do modelo de caso de uso do software cujos atores são os usuários do sistema, o modelo de caso de uso de negócio modela a empresa inteira como um sistema e os atores são as pessoas e organizações que se relacionam (fazem negócios) com ela. Da mesma forma, os objetos de negócio não são necessariamente instâncias de classes de um sistema, mas departamentos, pessoas e sistemas inteiros que operam dentro da empresa (Kroll & Kruchten, 2003).

Outros artefatos que também podem ser construídos são:

a) *Avaliação da organização alvo*. Descrição do *status* corrente da organização na qual o sistema será instalado.
b) *Regras de negócio*. Declaração de políticas e condições que devem ser satisfeitas.
c) *Especificações suplementares de negócio*. Documento que apresenta definições do negócio não contempladas no modelo de caso de uso ou modelo de objetos.
d) *Glossário de negócio*. Definição de termos técnicos relevantes ao negócio.
e) *Documento de arquitetura de negócio*. Uma visão de aspectos arquiteturais do negócio a partir de vários pontos de vista. Deve ser usado apenas quando decisões sobre mudanças no negócio devem ser feitas ou quando o negócio necessita ser descrito a terceiros.

Jacobson (1994) apresenta detalhadamente técnicas de modelagem de negócio com casos de uso. Via de regra, em modelos de negócio, atores são entidades externas à empresa alvo, casos de uso são os processos internos da empresa que normalmente afetam estes atores externos e os subsistemas podem ser setores ou departamentos da empresa. Usam-se as mesmas técnicas de modelagem de caso de uso para sistemas informatizados, apenas o nível de abstração é maior porque a empresa como um todo é considerada um sistema.

O *workflow* da disciplina de modelagem de negócio é apresentado na Figura 5-3.

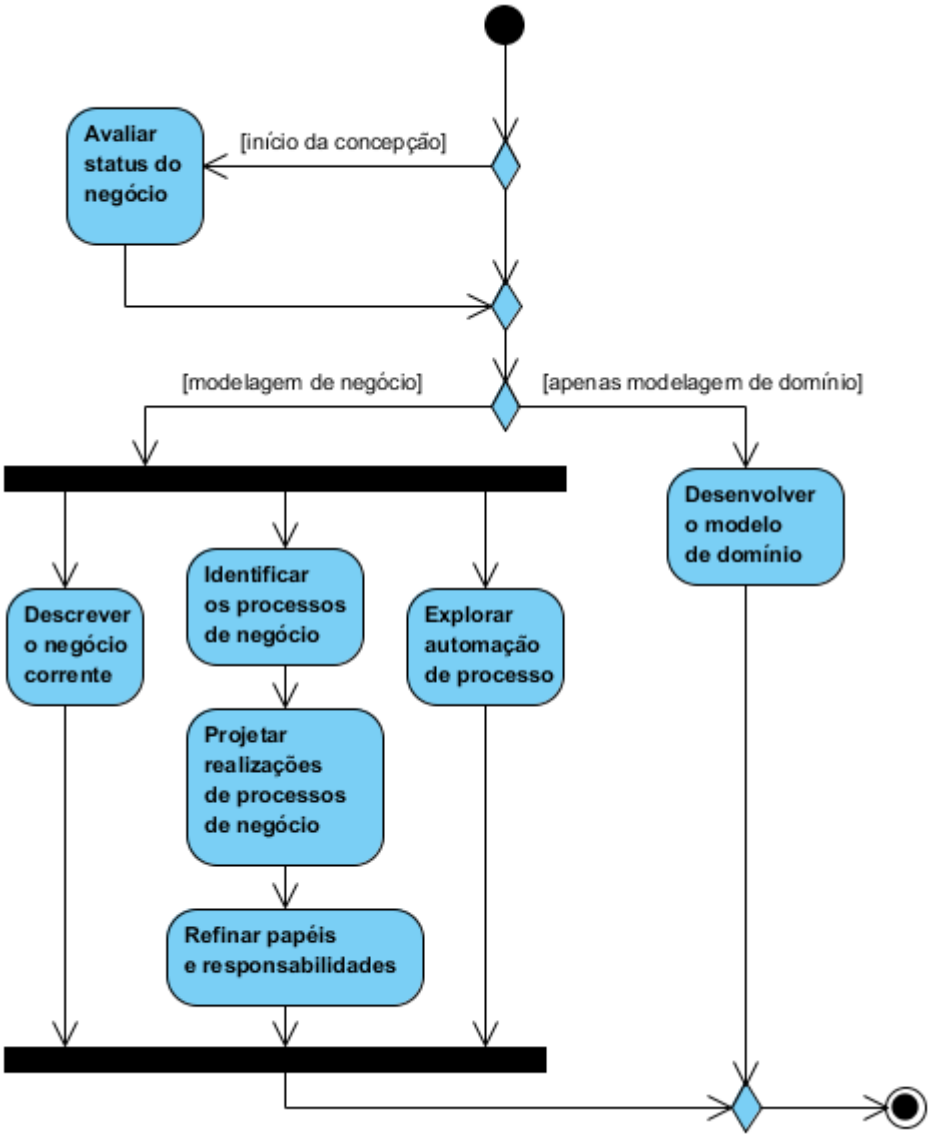

Figura 5-3: *Workflow* da disciplina de modelagem de negócio.

O modelo de casos de uso de negócio pode ser usado para gerar o modelo de casos de uso do software. Objetos de negócio ativos possivelmente serão atores ou sistemas-atores no modelo de casos de uso do software e os processos de negócio possivelmente serão quebrados em processos individuais que serão identificados como casos de uso do software. Exemplos podem ser encontrados em Kroll e Kruchten (2003). A vantagem de iniciar com o modelo de negócio, na maioria das vezes, é conseguir perceber mais claramente o sistema a ser desenvolvido no contexto geral da empresa.

5.3.2.3 Requisitos

Requisitos são a expressão mais detalhada sobre aquilo que o usuário ou cliente precisa em termos de um produto de software. A disciplina de requisitos RUP, entretanto, trata não apenas dos requisitos (que são necessidades expressas pelo cliente), como também do *design* da interface do sistema (que é uma possível solução para as necessidades apresentadas).

Os objetivos desta disciplina são (Kruchten, 2003):

a) Estabelecer e manter concordância com o cliente e outros interessados sobre o que o sistema deve fazer, incluindo o porquê.
b) Fornecer aos desenvolvedores uma melhor compreensão sobre os requisitos do sistema.
c) Delimitar escopo do sistema, ou seja, o que pertence e o que não pertence ao sistema.
d) Prover uma base para o planejamento técnico das iterações.
e) Prover uma base para a estimação de custo e tempo de desenvolvimento.
f) Definir uma interface com usuário focada em suas necessidades e objetivos.

O *workflow* RUP para a disciplina inclui cinco papeis:

a) *Analista de sistemas*. Lidera e coordena a eliciação de requisitos e a modelagem de casos de uso, especificando os limites do sistema.
b) *Especificador de requisitos* ou *especificador de casos de uso*. Responsável pelo detalhamento ou expansão dos requisitos, na forma de casos de uso.
c) *Arquiteto.* Deve assegurar que uma arquitetura consistente poderá ser obtida a partir dos casos de uso.
d) *Revisor de requisitos*. É composto por vários tipos de pessoas que revisam os requisitos, seja para garantir que estejam de acordo com normas e padrões, seja para garantir que estejam de acordo com as necessidades dos interessados.

Os três principais artefatos da disciplina de requisitos são os seguintes:

a) *Visão geral de sistema*. É um documento de alto nível indicando especialmente aos financiadores do projeto qual o seu escopo e vantagens. O documento pode ter uma visão do que será e não será incluído, capacidades operacionais desejadas, perfis de usuários e interfaces com sistemas externos quando for o caso.
b) *Modelo de casos de uso*. Os casos de uso incorporam os requisitos funcionais e boa parte dos não funcionais. Eles servem como uma definição de funcionalidades desejadas e são usados ao longo de todo processo de desenvolvimento.
c) *Especificações suplementares*. Consiste nos requisitos suplementares, isto é, as restrições gerais que se aplicam ao sistema como um todo e não apenas a um ou outro caso de uso.

Complementarmente, a disciplina de requisitos também pode produzir:

a) *Um glossário*, contendo os termos técnicos do jargão da aplicação.
b) *Storyboards*, que servem como base para o *design* da interface com o usuário.

O *workflow* da disciplina de requisitos é mostrado no diagrama de atividades UML da Figura 5-4.

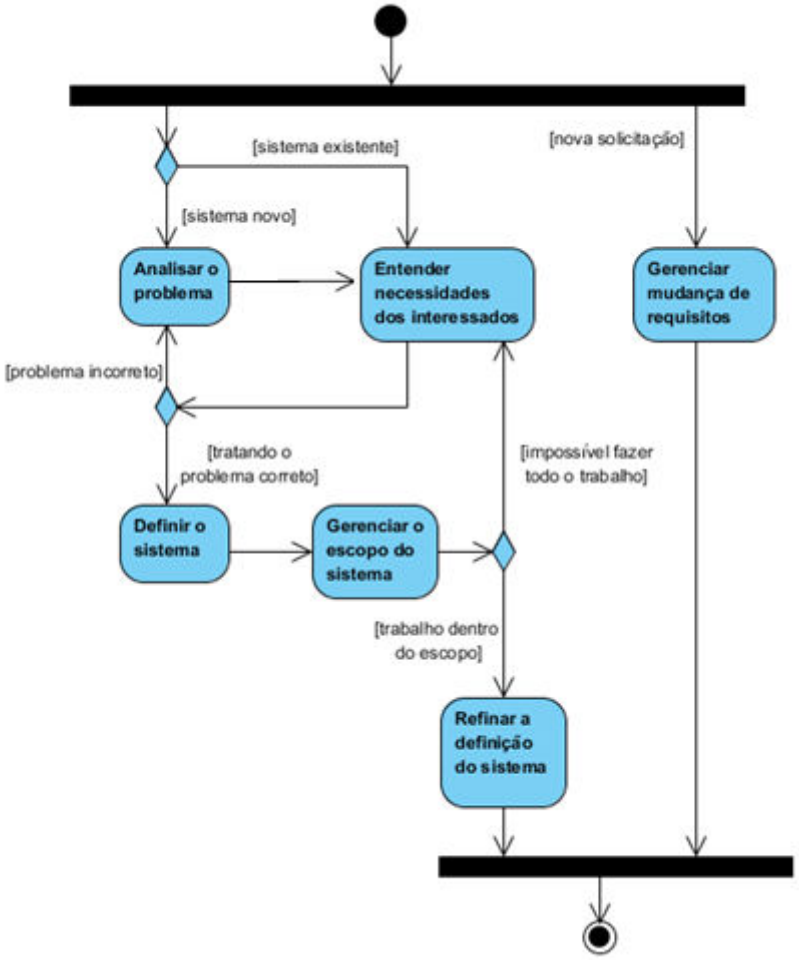

Figura 5-4: *Workflow* da disciplina de requisitos.

As atividades do *workflow* podem ser assim caracterizadas:

a) *Analisar o problema*. Deve-se obter concordância dos interessados sobre o enunciado do problema que se está tentando resolver. Deve-se também identificar o próprios interessados e os limites e restrições do sistema.
b) *Entender necessidades dos interessados*. Usando variadas técnicas de eliciação, obter os requisitos e as reais necessidades dos interessados.
c) *Definir os sistema.* Baseando-se nas necessidades dos interessados, estabelecer um conjunto de funcionalidades que se pode considerar para entrega. Determinar os critérios para priorização das necessidades mais importantes e identificar atores e casos de uso para cada funcionalidade base do sistema.
d) *Gerenciar o escopo do sistema.* Coletar informações importantes dos interessados e mantê-las na forma de atributos de requisitos, identificando, por exemplo, a importância do requisito, sua urgência, impacto no negócio, etc.
e) *Refinar a definição do sistema.* Usando o modelo de casos de uso, refinar a descrição dos requisitos funcionais e obter os não funcionais associados.
f) *Gerenciar mudança de requisitos.* Usar atributos de requisitos e rastreabilidade para avaliar o impacto da mudança de requisitos. Usar uma autoridade central para

controlar as mudanças nos requisitos. Manter entendimento com o cliente e definir expectativas realistas sobre o que poderá ser feito.

5.3.2.4 Análise e Design

Análise implica no estudo do problema de forma mais aprofundada. Requisitos são detalhados e incluídos em modelos de análise como o modelo conceitual, de interação e funcional. Já o *design* consiste em apresentar uma possível solução tecnológica para o modelo de análise. Os modelos de *design* podem ser o modelo dinâmico, de interface, de persistência, além de outros. Wazlawick (2011) apresenta muitas informações sobre como realizar esta disciplina do RUP.

Para o RUP a disciplina de análise e *design* tem como característica principal a modelagem, ou seja, a transformação dos requisitos em modelos úteis para a geração de código. Boa parte da análise considera apenas a funcionalidade, mas não as restrições, especialmente as tecnológicas. Ou seja, a análise concentra-se no sistema ideal, independente de tecnologia, e o *design*, por sua vez, vai lidar com estas questões mais físicas.

Os dois principais papéis relacionados a esta disciplina são:

a) *Arquiteto de Software*. Ele coordena as atividades técnicas do projeto, estabelecendo a estrutura sobre a qual a arquitetura final é construída. Ele define os agrupamentos de elementos (como componentes ou pacotes) e decide sobre suas interfaces. Sua visão do sistema ocorre mais em extensão, porém, do que em profundidade.
b) *Designer*. Cada *designer* define as responsabilidades, operações, atributos e associações de um conjunto de classes ou pacotes colocados sob sua responsabilidade.

Pode-se ainda ter os seguintes papeis opcionais:

a) *Designer de banco de dados*. Usualmente é necessário quando o sistema utiliza banco de dados, especialmente se este for complexo.
b) *Designer de cápsula*. É necessário apenas no caso de sistemas de tempo real e é responsável por garantir que o sistema responderá da forma apropriada.
c) *Revisor de arquitetura* e *revisor de design*. São especialistas que revisam os artefatos produzidos nesta disciplina.

Os principais artefatos desta disciplina são:

a) *Modelo de design*. O principal *design* do sistema sendo construído. Usualmente é um diagrama de componentes ou pacotes incluindo as principais classes do sistema.
b) *Documento de arquitetura de software*. Captura vários aspectos da arquitetura do sistema sob diferentes pontos de vista. Por exemplo, visão de módulos (como o sistema é estruturado como unidades de código), visão de tempo de execução (como o sistema é estrututado como elementos com comportamento e interações), visão de implantação ou *deployment* (como o sistema é implantado no hardware), visão de implementação (como os artefatos se organizam em um sistema de arquivos) e modelo de dados (qual a estrutura do repositório de dados).

O *workflow* da disciplina de análise e *design* é mostrado na Figura 5-5.

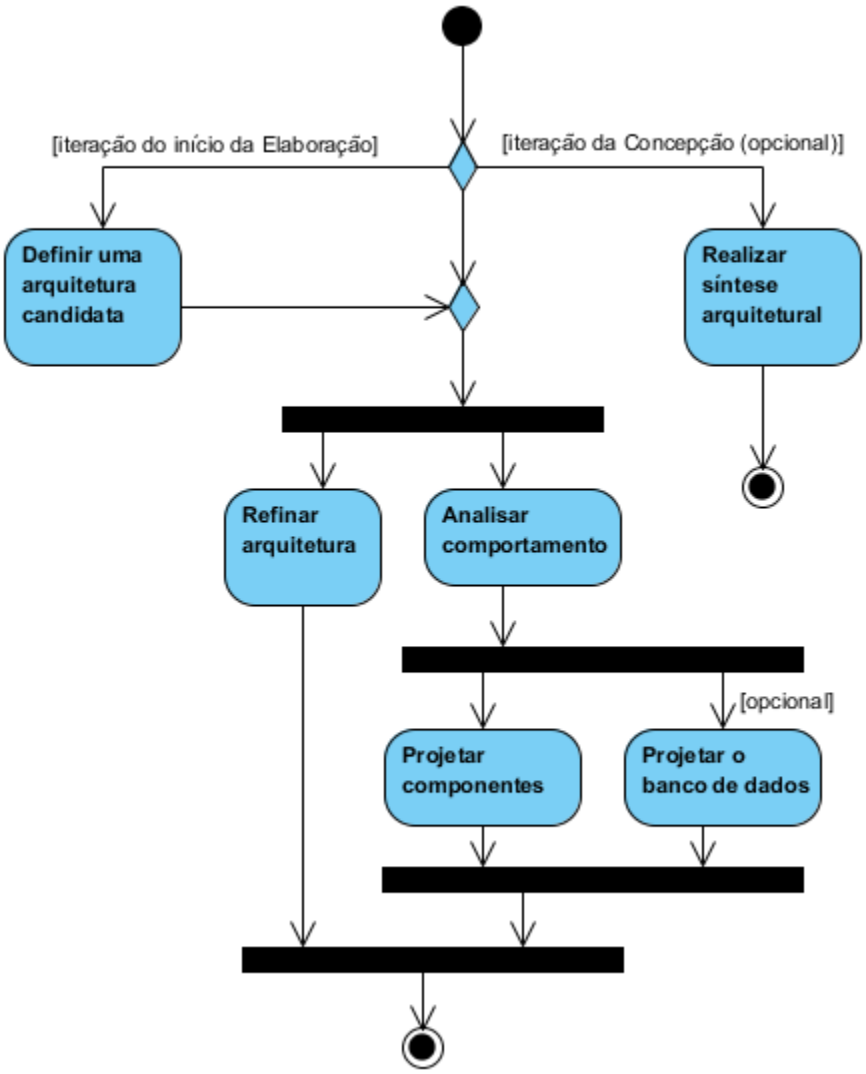

Figura 5-5: *Workflow* da disciplina de análise e *design*.

A figura mostra que algumas atividades são dependentes da fase em que o projeto se encontra. Na fase de concepção poderá ser feita a *síntese arquitetural*, que é uma prova de conceito de que a arquitetura é viável. No início da elaboração será, então, definida uma arquitetura candidata inicial possivelmente baseada nessa prova de conceito. Nas iterações subsequentes, a ênfase muda para a adição ou refinamento de funcionalidades sobre a arquitetura que vai se estabilizando até a fase de construção.

A Tabela 5-2 mostra o detalhamento das atividades deste *workflow*.

Tabela 5-2: Detalhamento do *workflow* para a disciplina de análise e *design*.

Atividade/Objetivo	Subatividades	Responsável
Realizar síntese arquitetural. Deve-se construir uma prova de conceito para mostrar que a arquitetura é viável.	Análise arquitetural	Arquiteto
	Construir prova de conceito arquitetural	
	Avaliar viabilidade de prova de conceito arquitetural	
Definir uma arquitetura candidata. Deve criar um esboço inicial da arquitetura a partir da modelagem conceitual e casos de uso.	Análise arquitetural	Arquiteto
	Análise de casos de uso	*Designer*
Refinar arquitetura. Deve fazer a transição natural da análise para o *design* identificando os elementos e mecanismos apropriados e mantendo a consistência e integridade da arquitetura.	Identificar mecanismos de *design*	Arquiteto
	Identificar elementos de *design*	
	Incorporar elementos de *design* existentes	
	Descrever a arquitetura de tempo de execução	
	Descrever a distribuição do sistema	
	Revisar arquitetura	Revisor de arquitetura
Analisar comportamento. Deve transformar as descrições de comportamento do caso de uso em elementos arquiteturais.	Análise de casos de uso	*Designer*
	Identificar elementos de *design*	Arquiteto
	Revisar o *design*	Revisor de *design*
Projetar componentes. Deve detalhar completamente classes e relações, interfaces de subsistemas e casos de uso reais.	*Design* de casos de uso	*Designer*
	Design de classes	
	Design de subsistemas	
	Revisar o *design*	Revisor de *design*
Projetar o banco de dados. Deve-se identificar as classes persistentes, projetar estruturas de BD adequadas para armazená-las e definir os mecanismos de salvamento e recuperação que atendam a eventuais requisitos suplementares.	*Design* de banco de dados	*Designer* de banco de dados
	Design de classes	*Designer*
	Revisar o *design*	Revisor de *design*

As atividades de análise e *design* e seus conceitos correlatos são detalhadamente apresentadas por Wazlawick (2011).

5.3.2.5 Implementação

A implementação é mais do que simplesmente produzir código. Ela implica também na organização deste código em componentes ou pacotes, na definição de possíveis camadas de implementação, nos testes de unidade e na integração de código de forma incremental.

Apenas o teste de unidade é tratado na disciplina de implementação. Os demais tipos de teste, inclusive testes de integração, são tratados na disciplina de teste.

A implementação de código em RUP pode ser de três tipos:

a) *Versões* (*builds*). São versões operacionais de um sistema que introduzem novas funcionalidades. Versões devem ser controladas e rastreadas para que alterações possam ser desfeitas em caso de necessidade. Projetos típicos apresentam novas versões regularmente; pelo menos uma versão por semana é desejável, podendo chegar a uma por dia.
b) *Integração*. A integração consiste na combinação de pelo menos duas partes de software independentes. A integração no RUP deve ser contínua, como nos métodos ágeis e diferente da integração por fase que ocorre no modelo cascata com subprojetos. Uma das maiores vantagens da integração contínua consiste na maior facilidade de encontrar erros já que a maior fonte potencial de erros durante o processo é o novo módulo que estiver sendo integrado e seus pontos de interação com o restante do sistema. RUP prevê que no mínimo uma integração por ciclo deva acontecer, mas o ideal é que aconteçam várias integrações por dia.
c) *Protótipos*. Devem ser usados primordialmente para reduzir risco, seja para obter uma melhor compreensão dos requisitos, seja para obter provas de conceito de arquitetura ou usabilidade.

Os principais papeis relacionados com a disciplina de implementação no RUP são:

a) *Implementador*. Desenvolve os componentes, artefatos relacionados e realiza o teste de unidade.
b) *Integrador*. Constrói as versões (*builds*).
c) *Arquiteto de software*. Define a estrutura ou modelo de implementação, estabelecendo, por exemplo, camadas ou partições de sistema.
d) *Revisor de código*. Inspeciona o código de acordo com padrões de codificação estabelecidos e boas práticas.

Os principais artefatos relacionados a esta disciplina são:

a) *Subsistema de implementação*. Coleção de elementos de implementação e outros subsistemas de implementação usados para estruturar o modelo em subsistemas que dividem o todo em partes menores.
b) *Elementos de implementação*. Um artefato de código (fonte, binário ou executável) ou arquivo contendo informação (*startup*, *readme* etc.), podendo agregar outros elementos de implementação, como por exemplo, uma aplicação contendo vários executáveis.
c) *Plano de integração de implementação*. Um documento que define a ordem em que os elementos e subsistemas devem ser implementados e especifica as versões a serem criadas a medida que as integrações ocorrem.

A Figura 5-6 mostra o *workflow* da disciplina de implementação no RUP.

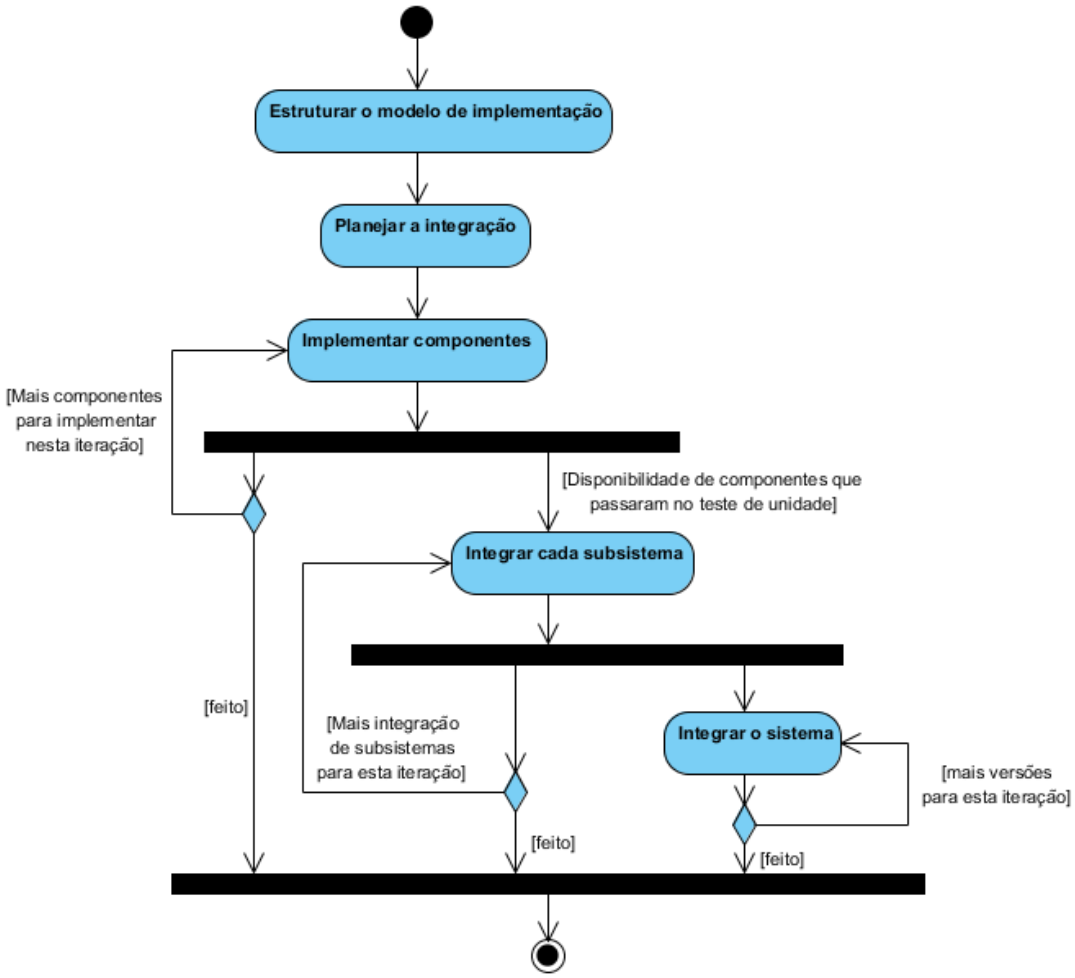

Figura 5-6: *Workflow* da disciplina de implementação.

Em cada iteração o plano de integração deve indicar quais sistemas serão implementados e em que ordem eles serão integrados. O responsável por cada subsistema definirá a ordem em que as classes serão implementadas (Seção 13.2.2).

Se os implementadores encontrarem erros de *design*, devem submeter um *feedback* de retrabalho sobre o *design*.

5.3.2.6 Teste

A disiciplina de teste no RUP exclui os testes de unidade, que são executados pelo programador na disciplina de implementação. O propósito da disciplina de testes é:

a) Verificar a interação entre objetos.
b) Verificar se todos os componentes foram integrados adequadamente.
c) Verificar se todos os requisitos foram corretamente implementados.
d) Verificar e garantir que defeitos tenham sido identificados e tratados antes da entrega do produto final.
e) Garantir que todos os defeitos tenham sido consertados, retestados e estancados.

No RUP a disciplina de teste é executada ao longo de todos os ciclos, o que facilita a detecção prematura de erros de requisitos e de implementação. Ao contrário das demais disciplinas, a

disciplina de teste visa encontrar as fraquezas do sistema, isto é, verificar como e quando o sistema poderia falhar.

Os principais papeis envolvidos com a disciplina de teste são:

a) *Gerente de teste*. Tem a responsabilidade geral pela disciplina de teste. Deve planejar o uso de recursos e gerenciar o andamento dos testes ajudando a resolver conflitos. É responsável pelo plano de teste e pelo sumário de avaliação de testes.
b) *Analista de teste*. É responsável pela identificação e definição dos testes requeridos. É responsável pela lista de ideias de teste, pelos casos de teste, pelo modelo de análise de carga, pelos dados de teste e pelos resultados dos testes.
c) *Designer de teste*. É responsável pela definição da abordagem de teste e pela garantia do sucesso em sua implementação. É responsável pela estratégia de teste, pela arquitetura de automação de testes, pela especificação da interface de teste, pela configuração do ambiente de teste e pela sequência de testes.
d) *Testador*. É responsável pela execução dos testes. É responsável pelos *scripts* de teste e pelo *log* de testes.

Em relação aos artefatos mencionados acima, pode-se destacar:

a) *Plano de teste*. Contém informação sobre os objetivos e metas do teste, bem como de seu cronograma. Identifica as estratégias a serem usadas, os recursos necessários e as configurações de teste requeridas.
b) *Lista de ideias de teste*. É uma lista mantida pelo analista de teste que enumera ideias, as vezes, parcialmente formadas, sobre como executar testes que possam ser úteis.
c) *Casos de teste*. São a evolução natural de algumas ideias de teste, especificando de forma mais detalhada o teste, suas condições e os dados a serem usados.
d) *Scripts de teste*. São procedimentos manuais ou automáticos usados pelo testador para proceder aos testes.
e) *Modelo de análise de carga*. É um tipo especial de teste de performance que identifica variáveis e seus valores a serem usados em diferentes testes de performance para simular ou emular características dos atores.
f) *Log de testes*. É o conjunto de dados crus capturados como resultado da execução das sequências de testes.
g) *Resultados de testes*. Consistem na análise do *log* que produz um relatório sobre defeitos encontrados e melhorias necessárias.

Um típico *workflow* da disciplina de teste em RUP é mostrado na Figura 5-7.

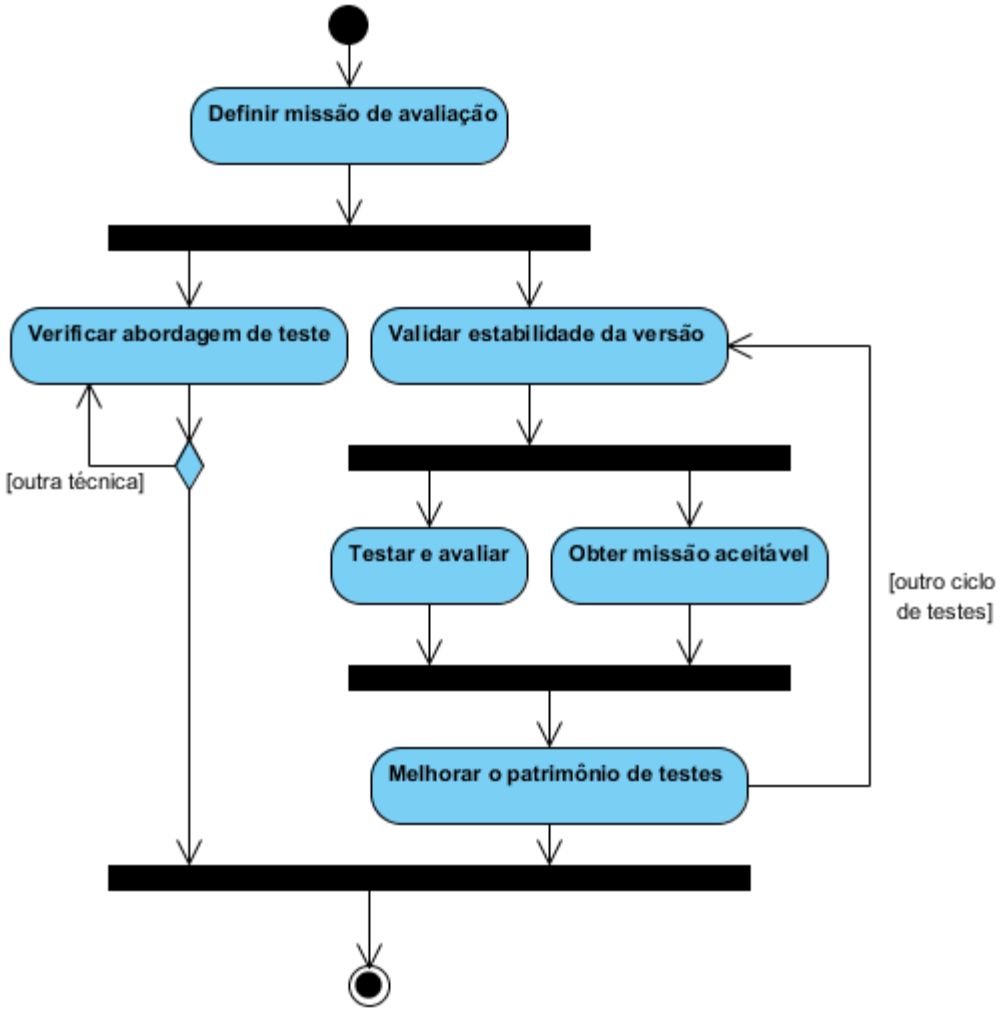

Figura 5-7: *Workflow* da disciplina de testes.

As atividades apresentadas no *workflow* são detalhadas da seguinte forma:

a) *Definir missão de avaliação*. Deve-se definir o foco e objetivos dos testes para a iteração e conseguir a concordância dos interessados.
b) *Verificar abordagem de teste*. Deve-se demonstrar que as várias abordagens de teste delineadas na estratégia de teste são adequadas e podem produzir resultados confiáveis.
c) *Validar estabilidade da versão*. Antes de iniciar o ciclo de teste para uma nova versão, deve-se verificar se a versão é suficientemente estável para que se inicie o ciclo de teste. Isso evita que se perca tempo fazendo testes sistemáticos em uma versão que ainda não está suficientemente madura.
d) *Testar e avaliar*. Consiste nas atividades clássicas de teste, ou seja, implementação, execução e avaliação dos resultados do teste.
e) *Obter missão aceitável*. Deve produzir para os interessados resultados úteis e compreensíveis de acordo com a missão de avaliação.
f) *Melhorar o patrimônio de testes*. Deve manter e aprimorar o patrimônio de testes, que compreende os seguintes artefatos: lista de ideias de teste, casos de teste, *script*s de

teste etc. É importante capitalizar estes elementos porque podem frequentemente ser úteis em testes futuros.

A disciplina de teste é detalhadamente explorada no Capítulo 13 e é fortemente relacionada com a noção de *qualidade* de software (Capítulo 11). Basicamente, qualidade é uma característica que não pode ser adicionada a um sistema no final de sua produção; ela precisa estar presente todo o tempo. A disciplina de teste é assim exercitada em todas as fases do projeto.

5.3.2.7 Implantação

O propósito da disciplina de implantação é a produção de versões (*release*s) do produto a serem entregues aos usuários finais. Entre outras atividades inclui-se a produção do software, seu empacotamento, instalação, migração de dados e apoio aos usuários (treinamento e suporte).

Apesar da disciplina de implantação se concentrar na fase de transição, quando se está entregando o produto definitivo, poderão haver várias *releases* ao longo do projeto, quando produtos preliminares poderão ser entregues, mesmo nas fases de elaboração ou construção.

Existem vários tipos de implantação de software conforme a maneira como ele é distribuído, que vão desde sistemas personalizados a serem implantados diretamente nos computadores do cliente até software disponibilizado para *download* pela internet. A diferença entre os casos consiste no grau de envolvimento da empresa desenvolvedora do software com a instalação do produto no ambiente final.

Os seguintes papeis podem ser citados na disciplina de implantação:

a) *Gerente de implantação*. Planeja e organiza a implantação. É responsável pelos resultados do beta-teste e pela verificação do empacotamento correto do produto para envio.
b) *Gerente de projeto*. É o responsável pela interface com o cliente e deve aprovar a versão final do sistema baseado nos resultados dos beta-testes.
c) *Redator técnico*. Planeja e produz o material escrito de apoio aos usuários.
d) *Desenvolvedor de cursos*. Planeja e produz material de treinamento em conjunto com o redator técnico.
e) *Artista gráfico*. Responsável pela produção artística quando necessária.
f) *Testador*. Executa os testes de aceitação e operação.
g) *Implementador*. Cria os *script*s de instalação e artefatos relacionados à instalação do produto.

Dependendo do tipo de implantação, diferentes artefatos poderão ser ou não necessários. O artefato principal em todos os casos é a *release*, ou seja, a versão final do software, que poderá ser composta pelos seguintes artefatos:

a) O software executável.
b) Artefatos de instalação como *script*s, ferramentas, arquivos, informação de licenciamento e orientações.
c) Notas sobre a versão (*release notes*), descrevendo a versão para o usuário final.

d) Material de suporte, como manuais de operação e de manutenção do sistema.
e) Material de treinamento.

No caso de software empacotado para venda em prateleira, os seguintes artefatos ainda poderão ser necessários:

a) *Lista de materiais*. Uma lista completa dos itens incluídos no produto.
b) *Arte do produto*. A parte do empacotamento que permite a identificação visual do produto.

Outros artefatos importantes da disciplina de implantação, mas que não são usualmente enviados ao cliente são os resultados dos testes.

O *workflow* da disciplina de implantação é apresentado na Figura 5-8.

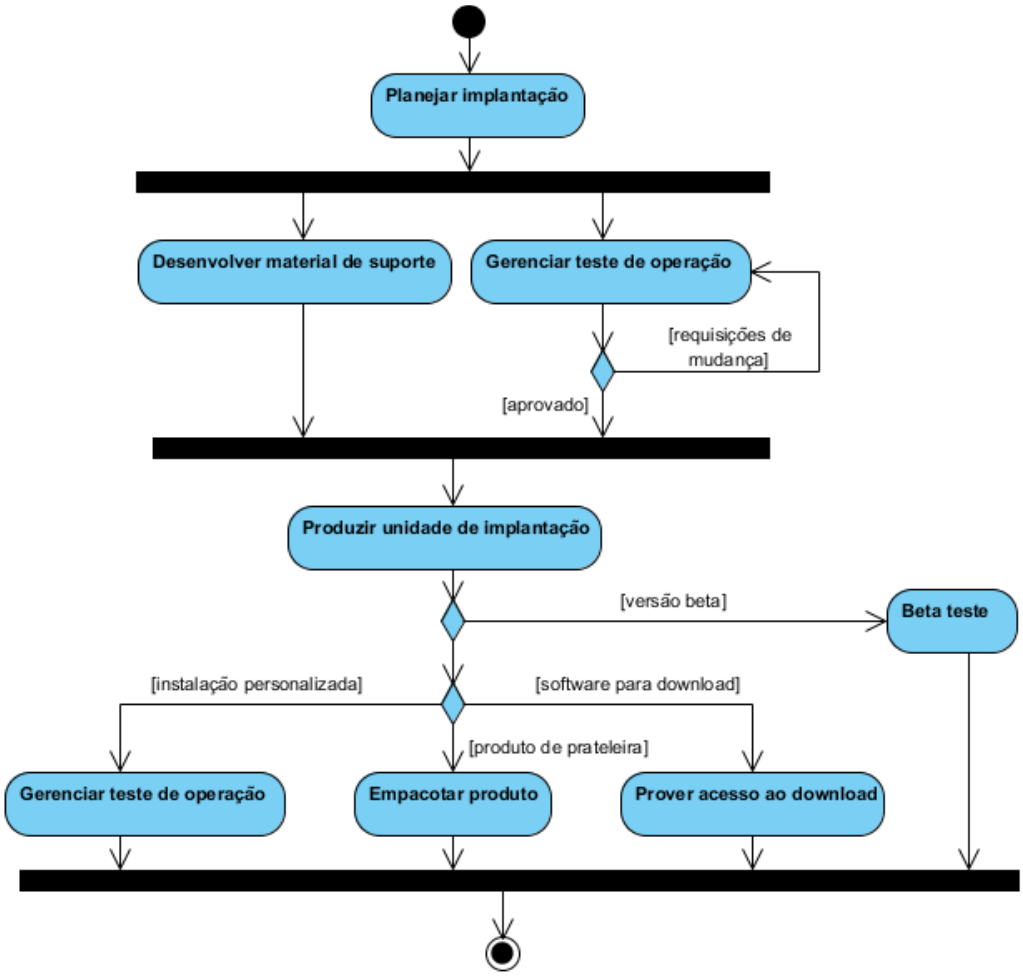

Figura 5-8: *Workflow* da disciplina de implantação.

Dentre estas atividades destaca-se:

a) *Planejar implantação*. É necessário ter uma previsão adequada das atividades e recursos que serão necessários para a implantação, bem como contar com a participação intensa do cliente neste planejamento.

b) *Desenvolver material de suporte*. Este material cobre todos os documentos e treinamentos que serão necessários para que o usuário final possa instalar, operar, manter e usar o sistema efetivamente.
c) *Produzir unidade de implantação*. A unidade de implantação é o produto empacotado para uso com todos os artefatos necessários. Pode se tratar de versões beta ou mesmo de versões finais do produto dependendo de seu estágio de desenvolvimento.
d) *Gerenciar teste de operação*. O teste de operação é semelhante ao teste de sistema e teste de aceitação, mas á feito no local definitivo de uso do software com máquinas alvo e dados reais.
e) *Beta teste*. O beta teste é feito pelo usuário, que vai avaliar a funcionalidade, performance e usabilidade do sistema, dando *feedback* para uma nova iteração de desenvolvimento.
f) *Empacotar produto*. Neste caso, o sistema e seus anexos são salvos em uma matriz, para posterior reprodução e todos os artefatos que o acompanham são efetivamente produzidos e juntados para envio.
g) *Prover acesso ao download*. O produto não só deve ser disponibilizado como deve-se tentar garantir acesso contínuo ao mesmo, mesmo no caso das conexões mais lentas. Além disso, a internet é uma boa fonte de *feedback* para o produto, mesmo depois de testado e este retorno dos usuários pode ser capitalizado através do site de *download*.

5.3.2.8 Gerenciamento de Mudança e Configuração

A disciplina de gerenciamento da mudança e configuração tem como principal objetivo manter a integridade do conjunto e artefatos produzidos ao longo do projeto. Muito esforço é despendido na produção destes artefatos e assim eles devem estar rastreáveis e disponíveis para reuso futuro. No RUP, a disciplina lida com três diferentes subáreas:

a) *Gerenciamento de configuração*. Responsável pela estruturação sistemática dos produtos. Artefatos como documentos e diagramas devem estar sob controle de versão e mudanças feitas devem ser visíveis e localizáveis. Também é necessário manter um registro de dependências ou rastreabilidade entre artefatos de forma que artefatos relacionados sejam atualizados quando necessário.
b) *Gerenciamento de requisições de mudança*. A área de CRM (*Change Request Management*) cuidará do controle das requisições de mudança em artefatos que produzem as diferentes versões destes. Nem todas as requisições de mudança são efetivamente realizadas; elas dependem de uma decisão de gerência que considere custos, impacto e a real necessidade da mudança.
c) *Gerenciamento de status e medição*. Requisições de mudança têm estados como: novo, atribuído, retido, concluído e entregue. Elas também podem ter atributos como causa, fonte, natureza, prioridade, etc. O gerenciamento de *status* coloca todos estes atributos em um sistema de informação tal que o andamento de cada solicitação seja verificável a todo momento pelo gerente do projeto.

Estas três dimensões formam o assim chamado "*Cubo CCM*"[66] (*Configuration and Change Management*), que indica o forte inter-relacionamento entre as três subáreas.

[66] flylib.com/books/en/1.560.1.134/1/

Os principais papeis relacionados a esta disciplina no RUP são:

a) *Gerente de configuração*. É o responsável por definir a estrutura do produto (versões de elementos e seus inter-relacionamentos) e por definir e alocar espaços de trabalho para desenvolvedores e integração. Ele também extrai os relatórios de *status* apropriados para o gerente do projeto.
b) *Gerente de controle de mudança*. Supervisiona o processo de controle de mudança. Em projetos grandes este papel pode ser executado por um comitê com representantes dos interessados e em projetos pequenos por uma única pessoa, usualmente o gerente do projeto ou arquiteto de software.

Os principais artefatos desta disciplina são:

a) *O plano de gerenciamento de configuração*. Descreve as políticas e práticas a serem usadas para definir versões, variantes, espaços de trabalho e procedimentos para gerenciamento de mudança. Este plano é parte do plano de desenvolvimento de software.
b) *Requisições de mudança*. As quais podem ser de vários tipos: relatórios de defeitos, mudança de requisitos, ou incremento de um ciclo a outro. Cada mudança é associada a um originador e uma causa principal. Mais tarde a mudança também terá uma análise de impacto.

O *workflow* da disciplina de gerenciamento de configuração e mudança é apresentado na Figura 5-9.

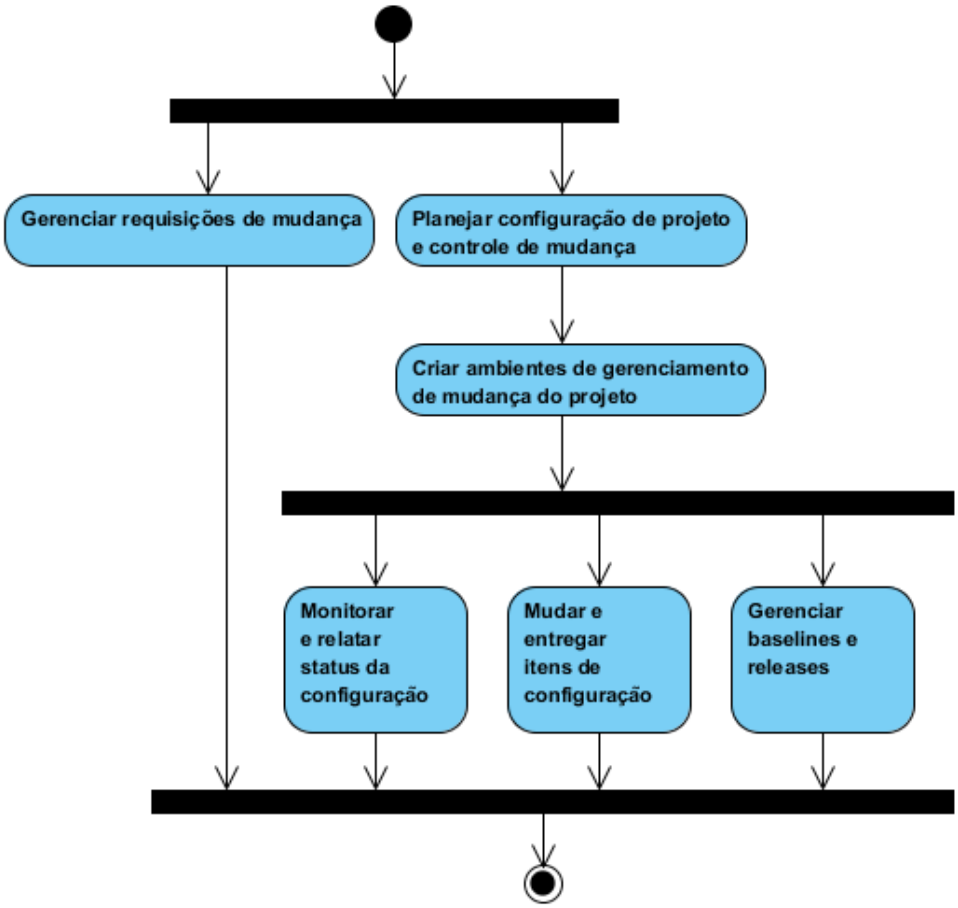

Figura 5-9: *Workflow* da disciplina de gerenciamento de mudança e configuração.

As atividades descritas no *workflow* podem ser assim resumidas:

a) *Planejar configuração de projeto e controle de mudança*. O plano de configuração de projeto descreve todas as atividades referentes a controle de mudança que devem ser atribuídas a responsáveis e ter recursos alocados durante o projeto. O plano também deve apresentar os meios para garantir que todos os interessados sejam informados sobre mudanças que ocorram em artefatos.

b) *Criar ambientes de gerenciamento de mudança de projeto*. Deve-se criar um ambiente de trabalho onde todos os artefatos em desenvolvimento possam ser acessados, modificados, integrados ou arquivados para uso posterior. O ambiente de gerenciamento de mudança oferece aos desenvolvedores e integradores espaços de trabalho privados e compartilhados para construção e integração de software.

c) *Mudar e entregar itens de configuração*. Esta atividade se refere à forma como cada papel pode criar um espaço de trabalho. Dentro de um espaço de trabalho, um papel pode acessar e alterar artefatos e disponibilizá-los para integração. A entrega de mudanças é feita em um espaço de integração que pode receber mudanças de vários papeis. A partir do espaço de integração o integrador constrói o produto criando suas *baselines*.

d) *Gerenciar baselines e releases*. Uma *baseline* é a descrição de todas as versões de artefatos que formam o produto em um dado momento. Tipicamente são criadas ao

final das iterações. Cada vez que uma entrega (*release*) for feita, deve haver necessariamente uma *baseline*.

e) *Monitorar e relatar status da configuração*. O relatório de *status* da configuração indica a versão de cada artefato e seus relacionamentos com versões anteriores, o que pode ser útil para verificar onde e quando ocorreram modificações no sistema.
f) *Gerenciar requisições de mudança*. Deve-se garantir que as mudanças em um projeto sejam feitas de maneira consistente.

O gerenciamento de configuração e mudança é uma tarefa árdua que pode ser impossível de realizar sem ferramentas adequadas. Existem implementações livres de sistemas de controle de versões como CVS[67], Git[68], Mercurial[69] e Subversion[70]. Dentre as implementações comerciais pode-se citar o Microsoft Visual SourceSafe[71] e o IBM Rational ClearCase[72]. Mais detalhes sobre essa disciplina são apresentados no Capítulo 10.

5.3.2.9 Ambiente

A disciplina de ambiente trata principalmente da configuração do próprio processo a ser usado para desenvolver o projeto. Em função do processo específico escolhido essa disciplina deve também tratar das ferramentas de apoio necessárias para que a equipe tenha sucesso no projeto.

Se uma equipe tenta implementar RUP integralmente sem perceber que ele na verdade é um *framework* para adaptação de processos, poderá ficar a impressão de que se trata de um processo altamente complexo (o que de fato é) e virtualmente impraticável. É necessário que um especialista em RUP avalie o escopo do projeto e a realidade da equipe para decidir quais elementos do RUP efetivamente devem ser usados para que sua adoção possa ser facilitada (ver também Seção 12.5).

Além disso, algumas versões mais simples de *RUP*, como *AUP* e *OpenUP*, já se estabeleceram como implementações independentes e podem ser consideradas ao invés da original.

Em relação à disciplina de ambiente, convém mencionar que ela não se refere ao produto, assim como as outras, mas à engenharia do processo em si, visando seu aprimoramento. Os principais papeis relacionados a esta disciplina são:

a) *Engenheiro de processo*. Responsável pelos seguintes artefatos: processo de desenvolvimento, caso de desenvolvimento, regras específicas de projeto e *template*s específicos de projeto.
b) *Especialista em ferramentas*. Responsável pelas ferramentas a serem usadas.
c) *Administrador de sistema*. Responsável pela infraestrutura de desenvolvimento.
d) *Redator técnico*. Responsável manual de guia de estilos.

[67] savannah.nongnu.org/projects/cvs/
[68] git-scm.com/
[69] mercurial.selenic.com/
[70] subversion.apache.org/
[71] msdn.microsoft.com/pt-br/library/ms181038%28v=vs.80%29.aspx
[72] www-01.ibm.com/software/awdtools/clearcase/

Dentre os artefatos mencionados, talvez o principal para um projeto específico seja o *caso de desenvolvimento*, que é a versão do processo geral especialmente adaptada para o projeto em questão, disciplina por disciplina.

O *workflow* para esta disciplina é apresentado na Figura 5-10.

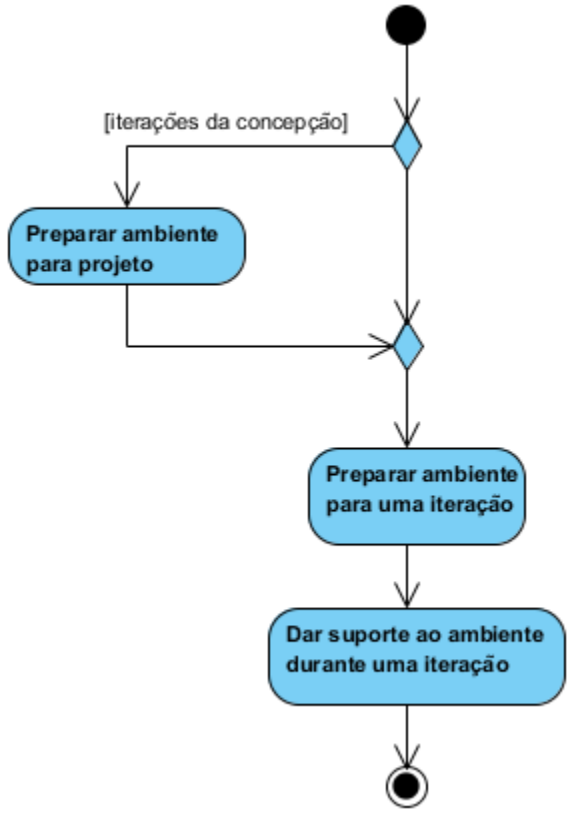

Figura 5-10: *Workflow* da disciplina de ambiente.

Em relação às atividades do *workflow*, pode-se indicar o seguinte:

a) *Preparar o ambiente para o projeto*. Seus objetivos incluem: avaliar a organização de desenvolvimento atual, avaliar o *status* das ferramentas de suporte, desenvolver um primeiro rascunho do caso de desenvolvimento, produzir uma lista de ferramentas candidatas e uma lista de artefatos-chave candidatos para o projeto específico.
b) *Preparar o ambiente para uma iteração*. Seus objetivos incluem: refinar ou completar o caso de desenvolvimento para a iteração corrente, preparar e, se necessário, personalizar as ferramentas, verificar se as ferramentas foram preparadas e instaladas corretamente, produzir o conjunto de *template*s específicos a serem usados na iteração e treinar as pessoas para usar as ferramentas e compreender o caso de desenvolvimento. Além disso, para cada iteração pode ser necessário produzir orientações sobre como fazer: modelagem de negócio, modelagem de casos de uso, *design*, programação, manuais de estilo, interfaces com usuário, testes e ferramentas.
c) *Dar suporte ao ambiente durante uma iteração*. É necessário porque apesar das instruções e treinamento, os desenvolvedores poderão precisar de orientação a medida que executam seu trabalho durante a iteração.

5.4 AUP - *Agile Unified Process*

O *Agile Unified Process*, ou *AUP*, é uma versão simplificada do RUP desenvolvida por Ambler (Ambler & Jeffries, 2002). *AUP* aplica técnicas ágeis como desenvolvimento dirigido por testes (TDD – *Test Driven Development*), modelagem ágil e refatoração.

Ao contrário de RUP, AUP tem apenas sete disciplinas:

a) *Modelagem*. Entender o negócio da empresa através de modelos produzidos que buscam também identificar possíveis soluções para estes problemas. Um guia de modelagem AUP usando diagramas UML pode ser encontrado na página do AUP na Internet[73].
b) *Implementação*. Transformar os modelos em código executável e criar testes de unidade.
c) *Teste*. Efetuar testes de integração, de sistema e de aceitação para garantir que o sistema tenha qualidade e implemente os requisitos corretos corretamente.
d) *Implantação*. Planejar as entregas de sistema para tornar o sistema acessível para usuários.
e) *Gerenciamento de configuração*. Gerenciar as versões e o acesso aos artefatos do projeto.
f) *Gerenciamento de projeto*. Controlar as atividades de projeto, garantindo que atividades sejam atribuídas às pessoas certas e executadas no prazo.
g) *Ambiente*. Dar suporte à equipe garantindo que as ferramentas, guias e padrões estejam disponíveis quando necessário.

O *Agile Unified Process* baseia-se na seguinte filosofia:

a) *Sua equipe sabe o que está fazendo*. As pessoas dificilmente gostarão de ter que ler instruções detalhadas sobre como fazer o seu trabalho, mas de tempos em tempos precisarão de alguma instrução para guiá-las. É importante identificar o ponto de equilíbrio entre as necessidades da equipe e a saturação com instruções demasiadas.
b) *Simplicidade*. Tudo deve poder ser descrito com simplicidade em algumas páginas, e não milhares de páginas.
c) *Agilidade*. AUP aceita e se conforma aos princípios ágeis.
d) *Foco em atividades de alto valor*. O foco está nas atividades que realmente produzem valor, não em qualquer coisa que eventualmente poderia ser feita.
e) *Independência de ferramentas*. A equipe pode usar qualquer ferramenta para desenvolver o projeto desde que seja bem adaptada às suas necessidades.
f) *AUP pode ser ajustado para satisfazer suas necessidades*. Como outros processos ágeis, AUP não possui cláusulas pétreas e pode ser ajustado de acordo com a necessidade.

Um refinamento importante do AUP em relação ao RUP é que o AUP distingue dois tipos de iterações:

[73] www.agilemodeling.com/style/

a) *Iterações de desenvolvimento*. Têm como objetivo desenvolver resultados para garantia de qualidade, prova de conceito ou redução de risco.
b) *Iterações de produção*. Têm como objetivo a produção de uma nova versão de código potencialmente entregável.

Os principais papeis AUP são:

a) *Administrador ágil de base de dados*: trabalha colaborativamente com a equipe para implementar, testar e evoluir a base de dados.
b) *Modelador ágil*: aquele que cria e evolui modelos de baixa cerimônia, ou seja, modelos que sejam simplesmente bons o suficiente, mas não excessivamente detalhados.
c) *Gerente de configuração*: o responsável pelo sistema de gerência de configuração e mudança.
d) *Implantador*: o responsável pela instalação do sistema.
e) *Desenvolvedor*: aquele de desenvolve o produto, escrevendo seu código.
f) *Engenheiro de processo*: aquele que cuida do processo de desenvolvimento e seus artefatos anexos, como padrões, guias, *templates*, etc.
g) *Gerente de projeto*: o que gerencia e protege os membros da equipe, constrói e coordena relacionamentos com os interessados, planeja, gerencia e aloca recursos, define prioridades e mantém a equipe focada.
h) *Revisor*: avalia os produtos de trabalho, fornecendo *feedback* para a equipe.
i) *Redator técnico*: responsável pela escrita de documentos e textos necessários em interfaces do sistema.
j) *Gerente de teste*: responsável pelo planejamento e gerenciamento dos esforços de teste.
k) *Testador*: responsável pela criação e execução dos testes.
l) *Especialista em ferramentas*: responsável pela instalação e manutenção das ferramentas necessárias ao processo de desenvolvimento.

A filosofia de produção de artefatos AUP é minimalista, ou seja, deve-se produzir o mínimo necessário de artefatos para produzir um sistema com qualidade. Os seguintes artefatos são, porém, considerados necessários pelo método em ordem de prioridades (os mais importantes primeiro):

a) *Sistema*. O software, hardware e documentação necessários para o cliente.
b) *Código fonte*. O código fonte do programa que segue um dos padrões de codificação existentes[74].
c) *Conjunto de testes de regressão*. A sequencia (suíte) de testes de unidade, integração e sistema, que possa ser executada sempre que houver alguma modificação no código.
d) *Scripts de instalação*. O código necessário para automatizar a instalação do produto no ambiente de produção.
e) *Documentação de sistema.* É toda a documentação que possa ser útil para o processo de operação e evolução do sistema.
f) *Notas de versão.* É o resumo das características da atual versão do sistema, incluindo problemas conhecidos, quando for o caso.

[74] www.ambysoft.com/essays/codingGuidelines.html

g) *Modelo de requisitos.* São os modelos gerados durante o processo de análise, incluindo o modelo de negócio, casos de uso, modelo conceitual, entre outros.
h) *Modelo de design.* São os modelos gerados durate o processo de *design*, incluindo os modelos de segurança, base de dados, arquitetura do software, interface com usuário, entre outros.

A documentação completa referente ao processo AUP pode ser baixada gratuitamente pela Internet[75].

5.5 *OpenUp - Open Unified Process*

Anteriormente conhecido como *Basic Unified Process* (*BUP*) ou *OpenUP/Basic*, o *Open Unified Process* (*OpenUP*) é uma implementação aberta do *UP* desenvolvida como parte do *Eclipse Process Framework* (*EPF*)[76].

A primeira versão do modelo, conhecida como *BUP* foi originada pela IBM que abriu a definição de uma versão mais leve do *RUP*. Entre 2005 e 2006 essa versão foi abraçada pela Fundação Eclipse[77] e passou a ser um de seus projetos.

OpenUP aceita, embora de forma simplificada, a maioria dos princípios *UP*. Porém, é um método independente de ferramenta e de baixa cerimônia, ou seja, não é exigida grande precisão e detalhes nos documentos.

O processo se baseia em quatro princípios:

a) *Colaboração* para alinhar interesses e compartilhar entendimentos.
b) *Evoluir* para continuamente obter *feedback* e melhorar.
c) *Balancear* prioridades que competem entre si de forma a maximizar valor para os interessados.
d) *Foco* na articulação da arquitetura.

O ciclo de vida também é estruturado em quatro fases, como no UP. As fases também são divididas em iterações. Mas aqui, as equipes se auto-organizam para planejar cada iteração. O esforço pessoal é organizado em micro-incrementos, que representam pequenas unidades de trabalho que produz um ritmo mais fino e mensurável para o projeto. A Figura 5-11 apresenta de forma esquemática o ciclo de vida do OpenUP em seus três níveis: projeto, iteração e esforço pessoal.

[75] www.ambysoft.com/unifiedprocess/agileUP.html
[76] www.methodsandtools.com/archive/archive.php?id=69p2
[77] epf.eclipse.org/wikis/openup/index.htm.
Em português: epf.eclipse.org/wikis/openuppt/index.htm

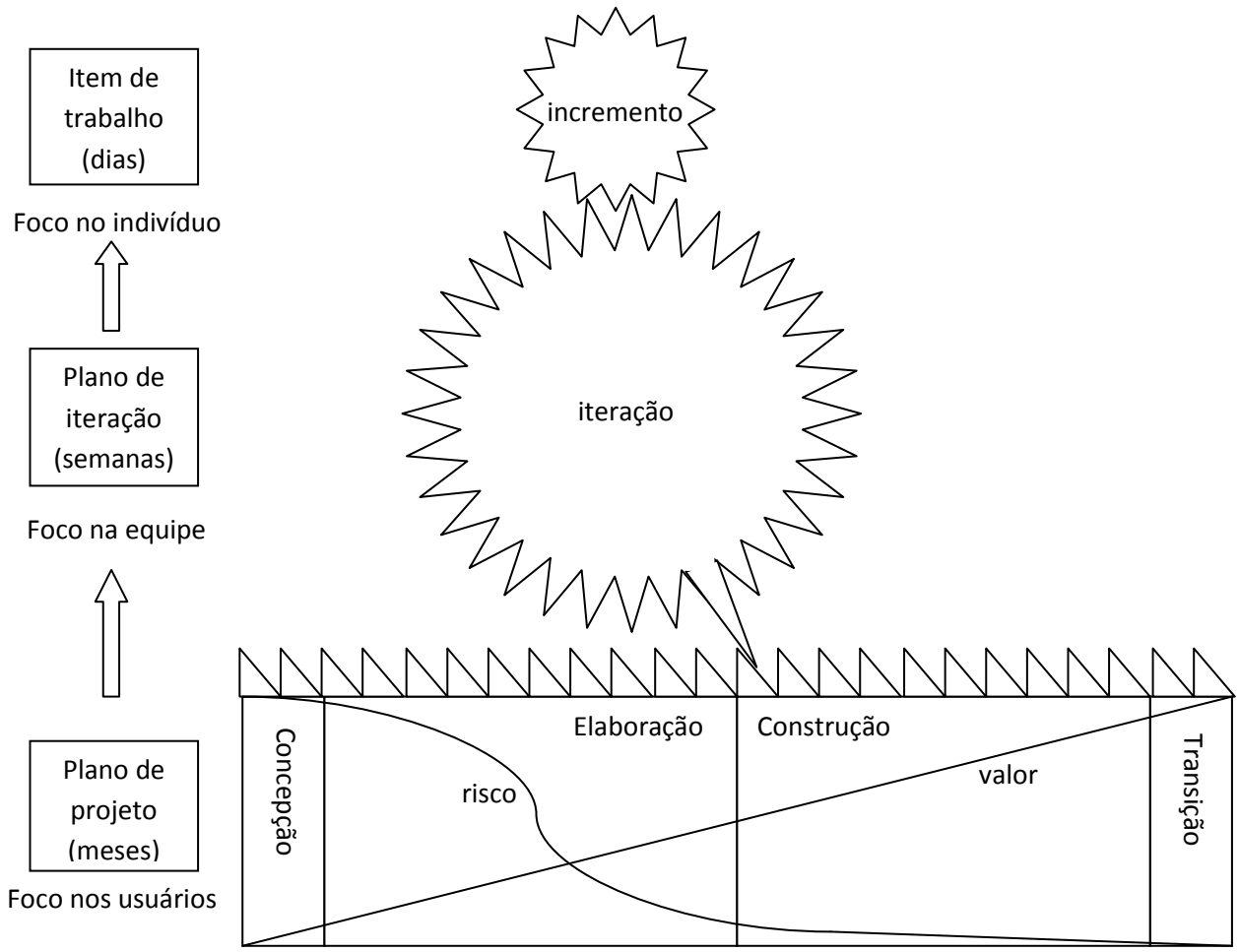

Figura 5-11: Ciclo de vida OpenUP.

Em relação ao RUP, a maioria das práticas opcionais foram eliminadas e outras foram mescladas. O resultado é um processo mais simples, mas ainda fiel aos princípios de RUP.

Em OpenUP cada prática pode ser adotada como um princípio independente que agrega valor à equipe. Desta forma, as práticas podem ser adotadas de forma progressiva, ao longo de vários projetos. Além disso, como nos modelos de software livre de código aberto, novas práticas podem ser sugeridas pela comunidade e passar a ser adotadas se houver interesse de outros.

OpenUP sugere um microciclo de vida para a iteração que é semelhante ao mostrado na Figura 5-12.

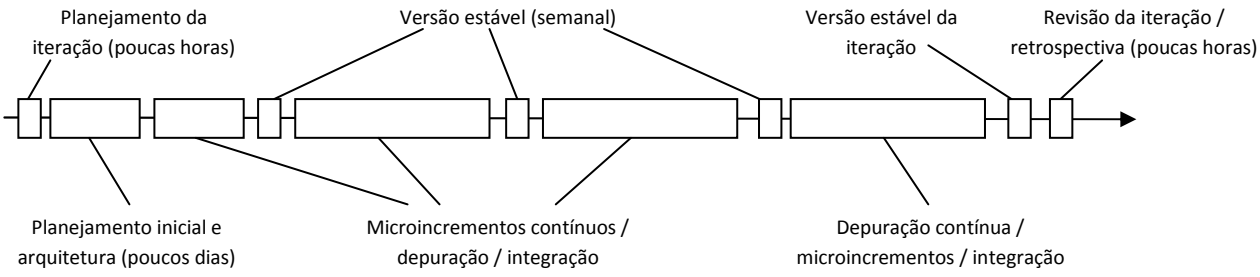

Figura 5-12: Modelo de ciclo de vida de uma iteração segundo *OpenUP*.

OpenUP é acessível gratuitamente a partir do site da Fundação Eclipse[78].

5.6 EUP - *Enterprise Unified Process*

O *Enterprise Unified Process*, ou *EUP*, foi inicialmente definido por Ambler e Constantine em 1999 e posteriormente refinado por Ambler, Nalbone e Vizdos (2005). O modelo EUP vê o desenvolvimento de software não apenas como um projeto a ser executado, mas como algo intrínseco ao ciclo de vida da própria empresa.

EUP foi proposto como uma extensão ao modelo RUP para prover, além das fases de RUP duas novas fases para tratar a evolução ou suporte ao sistema e a aposentadoria do sistema. Além destas duas fases, várias novas disciplinas relacionadas à empresa foram adicionadas. As novas fases introduzidas no modelo EUP são:

a) *Produção*. O desenvolvimento de sistemas usualmente não acaba quando o produto é entregue e colocado em uso. A fase de produção trata exatamente das atividades que ocorrem após a transição, incluindo o suporte, correção e ajustes e evolução do sistema.
b) *Aposentadoria*. A fase de aposentadoria consiste na retirada de um sistema de operação. É a fase final de qualquer sistema. Sistemas antigos retirados de operação para serem substituídos por sistemas novos podem causar sérios danos à empresa se o processo não for gerenciado adequadamente.

Em relação às disciplinas, além das nove disciplinas de RUP, o modelo introduz uma nova disciplina de projeto que é "operação e suporte", e um novo grupo de disciplinas, as *disciplinas de empresa*, que são sete:

a) *Modelagem de negócio de empresa*. O RUP já apresenta uma disciplina de modelagem de negócio, mas do ponto de vista do sistema a ser desenvolvido. A modelagem de negócio de empresa do EUP é mais abrangente, incluindo todos os processos da empresa e, desta forma, relações entre diferentes sistemas.
b) *Gerenciamento de portfólio*. Um portfólio é uma coleção e projetos de software em andamento e concluídos. Trata-se de projetos de alto risco e alto retorno e projetos de baixo risco e baixo retorno que devem ser gerenciados como ações para satisfazer as necessidades estratégicas da empresa.
c) *Arquitetura de empresa*. A arquitetura de empresa define como ela trabalha. Essa disciplina é especialmente útil se a empresa possuiu muitos produtos de software. Deve haver consistência na forma como eles são desenvolvidos, negociados e entregues. A arquitetura de empresa é a chave para compreender isso como um processo.
d) *Reuso estratégico*. O reuso estratégico vai além do reuso que se consegue dentro de um único projeto. Ele se estende entre diferentes projetos. Esse tipo de reuso costuma produzir mais valor do que o reuso simples dentro de um único projeto.
e) *Gerenciamento de pessoas*. Esta disciplina define uma abordagem para gerenciamento de recursos humanos da área de Tecnologia de Informação. É preciso gerenciar o

[78] epf.eclipse.org/wikis/openup/

pessoal, contratar, demitir, substituir, alocar pessoas a projetos e investir em seu crescimento.

f) *Administração de empresa*. Esta disciplina define como a empresa cria, mantém, gerencia, e entrega produtos físicos e informações de forma segura.

g) *Melhoria de processo de software*. Esta disciplina trata da adequação e evolução do processo de software para a empresa como um todo, não apenas a adequação do processo a cada projeto individual.

O objetivo principal da disciplina de *operação e suporte* é manter o produto funcionando no ambiente de produção. Deve-se garantir que o software funcione corretamente, que a rede esteja funcionando, que os dados sejam guardados em *backups,* e possam ser restaurados se necessário. Planos de recuperação de desastre devem ser criados e, se for o caso, executados (Ambler S. W., 2010)[79].

5.7 *OUM - Oracle Unified Method*

O *Oracle Unified Method*, ou OUM (Oracle, 2009)[80], é um *framework* de processo de desenvolvimento de software iterativo e incremental adequado a uso com produtos Oracle: bancos de dados, aplicações e *middleware*.

OUM é uma implementação do Processo Unificado que suporta, entre outras características: *Service Oriented Architecture* (SOA), *Enterprise Integration*, software personalizado, gerenciamento de identidade (*Identity Management*, IdM), governança, risco e adequação (Governance, Risk and Compliance, GRC), segurança de banco de dados, gerenciamento de performance e inteligência empresarial.

A Figura 5-13 apresenta esquematicamente o ciclo de vida OUM, que é composto por cinco fases e 14 disciplinas (Oracle, 2007).

[79] www.enterpriseunifiedprocess.com/essays/operationsAndSupport.html
[80] www.oracle.com/consulting/library/briefs/oracle-unified-method.pdf

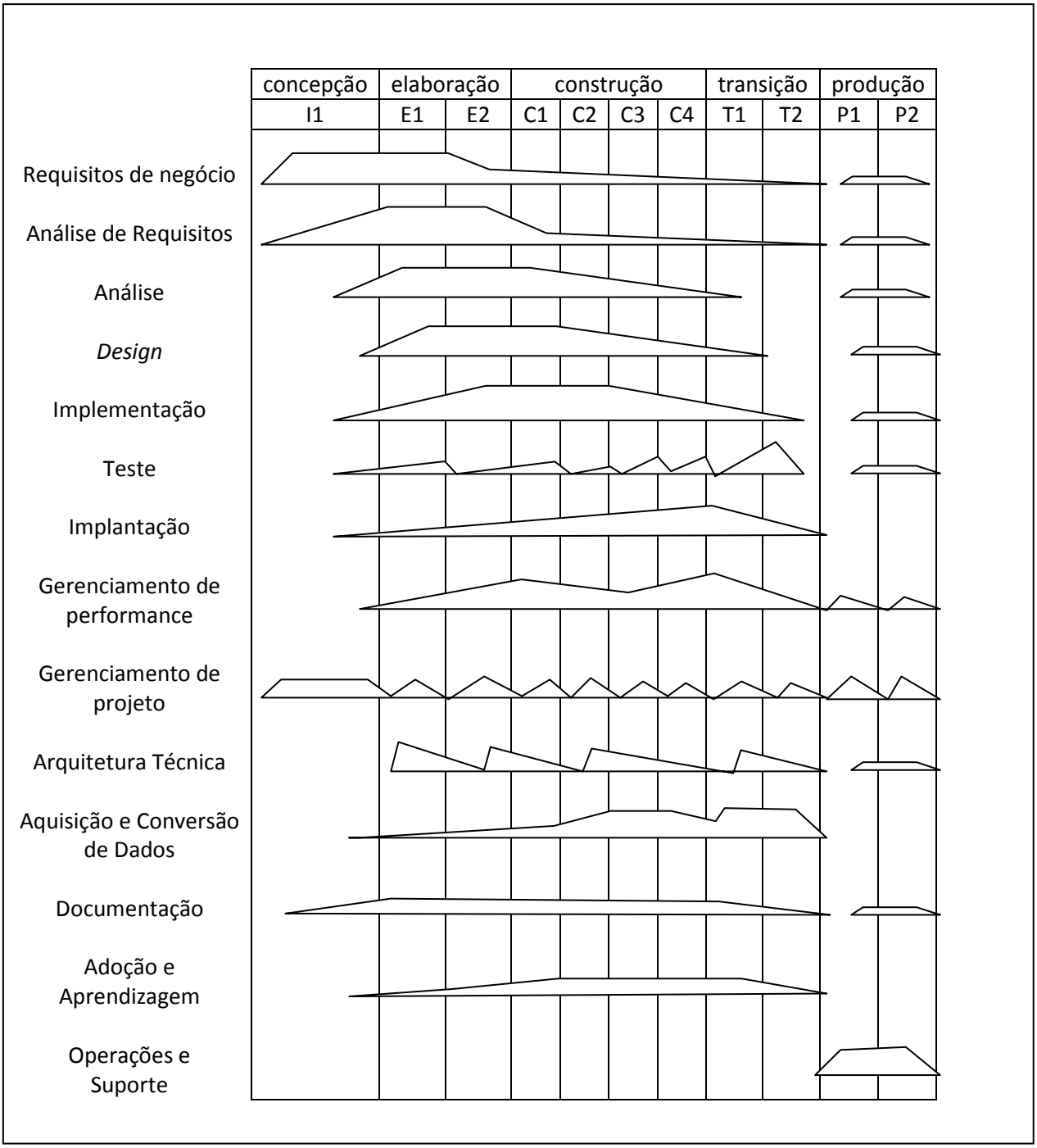

Figura 5-13: Ciclo de vida OUM.

Em relação ao RUP observa-se a introdução da fase de Produção, como o mesmo significado que tem no EUP, e também a eliminação das disciplinas de Ambiente (visto que o ambiente já é definido pelas ferramentas Oracle) e Gerenciamento de Configuração (também já disponibilizado pelas ferramentas).

Além disso, a disciplina RUP de Análise e *Design* é dividida em duas partes. Novas disciplinas são também acrescentadas:

a) Gerenciamento de Performance.
b) Arquitetura Técnica.

c) Aquisição e Conversão de Dados.
d) Documentação.
e) Adoção e Aprendizagem.
f) Operações e Suporte.

OUM é ao mesmo tempo uma instanciação do Processo Unificado, mas também é fortemente um modelo orientado a ferramentas (Seção 3.13). Em resumo, as disciplinas OUM podem ser assim caracterizadas:

a) *Requisitos de negócio*. Os requisitos de negócio da nova aplicação ou de sua evolução são identificados e modelados. As principais saídas desta disciplina são: objetivos e metas de negócio e a lista de requisitos funcionais e não funcionais.
b) *Análise de requisitos*. Os requisitos são organizados em um modelo de casos de uso. As principais saídas são: modelo de caso de uso, protótipos de interface e descrição em alto nível da arquitetura do sistema.
c) *Análise*. O modelo de casos de uso é usado para a descoberta dos conceitos e seus atributos e associações, formando assim o modelo conceitual, ou modelo de classes de análise. As principais saídas são: modelo conceitual e a revisão da arquitetura de sistema.
d) *Design*. O modelo de análise é agora instanciado em um modelo de *design*, derivado da arquitetura inicial, e que além de informações sobre classes e funcionalidades, vai indicar também os aspectos técnicos da implementação, a maioria dos quais é mencionada nos requisitos suplementares (não funcionais). As principais saídas são: modelo de *design*, arquitetura detalhada e modelo de implantação (*deployment*).
e) *Implementação*. A implementação visa produzir os elementos de código necessários para o funcionamento do sistema, bem como os testes de unidade. As principais saídas são: a versão do software pronta para os testes de sistema e a arquitetura do software enriquecida com os aspectos de implementação.
f) *Teste*. Os testes de sistema e de aceitação devem ser feitos para garantir a qualidade e conformação do sistema aos requisitos. As principais saídas são os casos de teste e a versão do sistema validada.
g) *Gerenciamento de performance*. São atividades integradas de garantia de qualidade da aplicação em relação aos requisitos suplementares de performance.
h) *Arquitetura técnica*. Sua meta é criar uma arquitetura que dê suporte à visão de negócios da empresa. Ela é fundamental para sistemas distribuídos e não triviais, como, por exemplo, sistemas com grande número de acessos.
i) *Aquisição e conversão de dados*. Usualmente novos sistemas estarão substituindo outros, sejam manuais ou informatizados, e seus dados precisarão ser adquiridos ou convertidos de alguma forma. Normalmente não é um processo simples, por isso, OUM acrescenta esta disciplina.
j) *Documentação*. As ferramentas Oracle suportam a produção de documentação-chave ao longo do projeto.
k) *Adoção e aprendizagem*. Focam o uso e aceitação de novas práticas associadas às novas ferramentas ou à evolução das antigas.
l) *Transição*. Inclui as atividades necessárias para a instalação do produto.

m) *Operações e suporte*. Essa disciplina trata do monitoramento e resposta aos problemas do sistema, atualização e correção de defeitos.

Uma das características do OUM é que não existe um fim abrupto após a fase de transição. A fase de operação é vista como a continuação natural do projeto e vários requisitos de baixa prioridade que eventualmente não foram implementados até a fase de transição poderão sê-lo durante a operação.

5.8 RUP-SE - *Rational Unified Process–Systems Engineering*

RUP-SE é uma extensão do modelo RUP para Engenharia de Sistemas (Cantor, 2003)[81]. Em outras palavras, é uma versão de RUP especialmente adequada para desenvolvimento de sistemas de grande porte, envolvendo, software, hardware, pessoas e componentes de informação.

RUP-SE inclui um *framework* de arquitetura que permite considerar o sistema a partir de diferentes perspectivas (lógica, física, informacional, etc.). Isso pode ser bastante útil quando se deseja demonstrar ou discutir aspectos de um sistema com diferentes interessados (cliente, analistas programadores, engenheiro de informação, etc.).

RUP-SE é especialmente adequado a projetos com as seguintes características:

a) São grandes o suficiente para comportar várias equipes de desenvolvimento trabalhando em paralelo.
b) Necessitam desenvolvimento concorrente de hardware e software.
c) A arquitetura é impactada por questões relativas à implantação.
d) Incluem a reengenharia de uma infraestrutura de tecnologia informação para suportar a evolução do negócio.

O *framework* é disponibilizado como um *plugin*, ou anexo, ao modelo RUP original.

[81] www.ibm.com/developerworks/rational/library/content/RationalEdge/aug03/f_rupse_mc.pdf (Consultado em 23/03/2010)

