Capítulo 5

UP – Processo Unificado

Este capítulo apresenta o *Processo Unificado*, uma importante referência como modelo de desenvolvimento de software. Embora seu uso, na prática, não seja tão significativo quanto se imaginou que seria há vinte anos atrás, ele incorpora muitas das boas práticas de engenharia de software aprendidas nas últimas décadas e apresenta tanto vertentes ágeis quanto prescritivas e orientadas a ferramentas. Assim, seu estudo ainda hoje pode ser considerado relevante.

Inicialmente, são explicadas suas *características* principais (Seção 5.1) e, em seguida, suas *quatro fases* originais (Seção 5.2). Na sequência, algumas implementações específicas do Processo Unificado são apresentadas, iniciando por RUP (Seção 5.3), a mais conhecida de todas. EUP (Seção 5.4) é uma implementação que acrescenta aspectos empresariais que não constam na definição original do processo. Já o RUP-SE (Seção 5.5) é uma implementação orientada a sistemas complexos de grande porte. OUM (Seção 5.6) é uma implementação para uso específico com ferramentas proprietárias. Finalmente, as implementações ágeis DAD (Seção 5.7) e *OpenUP* (Seção 5.8) são brevemente apresentadas.

O Processo Unificado (UP ou *Unified Process*) foi criado a partir da experiência de três pioneiros da orientação a objetos nos anos 1990 (Jacobson, Booch & Rumbaugh, 1999), sendo o resultado de muitos anos de experiência acumulada em projetos, notações e processos. Os modelos existentes na época, Objectory de Jacobson, OMT de Rumbaugh e o Método Booch, foram combinados, refinados e complementados por Per Kroll e especialmente Philippe Kruchten (Kroll & Kruchten, 2003; Kruchten, 2003).

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:

- Elas têm uma descrição clara e precisa.
- Apresentam responsáveis.
- Nelas são determinados artefatos de entrada e saída.
- São determinadas dependências entre as atividades.
- Seguem um modelo de ciclo de vida bem definido.
- São acompanhadas de uma descrição sistemática de como podem ser executadas com as ferramentas disponibilizadas (procedimentos).
- Preconizam o uso da linguagem UML.

As atividades incluem *workflows*, que são grafos que descrevem as dependências entre diferentes atividades. Os *workflows* estão 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 na maneira como as diferentes disciplinas são definidas e organizadas. As implementações também podem variar na 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. Algumas das principais implementações são abordadas neste capítulo, mas com maior detalhamento para RUP.

5.1 Caracterização do Processo Unificado

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

- Dirigido por casos de uso.
- Centrado na arquitetura.
- Iterativo e incremental.
- Focado em riscos.

Essas características serão discutidas nas próximas subseções.

5.1.1 Dirigido por Casos de Uso

O *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 (2015) 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:

- *Definição e validação da arquitetura do sistema*: em geral, classes e atributos são obtidos a partir dos textos dos casos de uso expandidos.
- *Criação dos casos de teste*: os casos de uso podem ser vistos como um roteiro para o teste de sistema e de aceitação, em que as funcionalidades são testadas do ponto de vista do cliente.
- *Planejamento das iterações*: os casos de uso são priorizados e o esforço para desenvolvê-los é estimado de forma que cada iteração desenvolva certo número deles (Seção 6.5).
- *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. Essas 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 fundamental do caso de uso no desenvolvimento de sistemas é a incorporação dos requisitos funcionais de 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 à parte.

5.1.2 Centrado na Arquitetura

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

A arquitetura, inicialmente, pode ser compreendida como o conjunto de classes, possivelmente agrupadas em componentes ou pacotes, 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. Assim, a arquitetura existe para que as funcionalidades sejam possíveis.

Wazlawick, R. S. *Engenharia de Software: Conceitos e práticas*, 2ª. ed. Elsevier, 2019.

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

Segundo o UP, a cada iteração devem-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 deixando muitas surpresas para depois.

5.1.3 Iterativo e Incremental

Assim como nos métodos ágeis, o UP preconiza o desenvolvimento baseado em iterações de duração fixa, em que, a cada iteração, a equipe incorpora à arquitetura as funcionalidades necessárias para realizar os casos de uso abordados.

Cada iteração produz um incremento no *design* do sistema, seja produzindo mais conhecimento sobre seus requisitos e arquitetura, seja produzindo um código executável. Espera-se que, em cada iteração, todas as disciplinas previstas sejam executadas com maior ou menor intensidade (ver. Figura 5.1 adiante). Então, assim como nos métodos ágeis, cada iteração vai implicar executar todas as atividades usuais de desenvolvimento de software.

5.1.4 Focado em Riscos

Em função das priorizações dos casos de uso mais críticos nas primeiras iterações, pode-se dizer que o UP é focado em riscos. Se esses casos de uso são os que apresentam maior risco de desenvolvimento, então devem ser tratados o quanto antes para que esse risco seja resolvido enquanto o custo para tratá-lo ainda é baixo e o tempo disponível para lidar com as surpresas é relativamente grande.

Esse tipo de abordagem (tratar primeiro os problemas mais difíceis) tem sido um valor incorporado a 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 maior aprendizado sobre o sistema e decisões arquiteturais mais importantes, vai fazer 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:

- *Concepção* (*inception*): trata-se da elaboração de uma visão em abrangência do sistema. Nessa fase são levantados os principais requisitos, um modelo conceitual preliminar é construído, bem como são identificados os casos de uso de alto nível, que implementam a funcionalidade requerida pelo cliente. Na fase de concepção calcula-se o esforço de desenvolvimento dos casos de uso e constrói-se o plano de desenvolvimento, composto por um conjunto de iterações nas quais são acomodados os casos de uso. Pode haver alguma implementação e teste, caso seja necessário elaborar protótipos para redução de risco.
- *Elaboração* (*elaboration*): nessa fase, as iterações têm como objetivo, predominantemente, detalhar a 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, sobre o qual serão aplicados padrões de *design* e uma descrição funcional poderá ser feita, bem como o *design* lógico e físico do sistema. Código será gerado e testado, especialmente para confirmar que as decisões de design

arquitetural são factíveis. Contudo, 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.

- *Construção* (*construction*): a fase de construção possui iterações nas quais os casos de uso mais complexos já foram detalhados e incorporados à arquitetura, que por isso já pode ser considerada estabilizada. Assim, as atividades de suas iterações consistem predominantemente na geração de código final e testes do sistema. Com a automatização da geração de código e a introdução de modelos de desenvolvimento dirigidos por teste, pressupõe-se que um bom *design* possa dar origem rapidamente a um código de alta qualidade.
- *Transição* (*transition*): a fase de transição consiste na implementação 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, que variam de caso a caso. Nessa fase pode haver ainda alguma revisão de requisitos, design e código, mas não se espera que seja muito significativa.

Uma das características do UP 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 da fase de elaboração, os requisitos devem ter sido extensivamente compreendidos e uma arquitetura estável deve ter sido 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). As próximas subseções apresentam um detalhamento sobre estas quatro fases.

5.2.1 CONCEPÇÃO

No Processo Unificado, a fase de concepção não deve ser muito longa. Recomenda-se um período de duas semanas a dois meses, dependendo da dimensão relativa do projeto.

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

O UP espera que, nessa fase, sejam estabelecidos também os casos de uso de sistema. Os casos de uso de sistema são a única expressão dos requisitos, não havendo outro documento de requisitos, exceto para os suplementares. De posse de um conjunto significativo de casos de uso, a equipe deve proceder como nas fases iniciais de planejamento do *Scrum* ou do XP, priorizando os casos de uso mais complexos e críticos em detrimento dos mais simples e triviais.

Na fase de concepção, a equipe deve produzir estimativas de esforço. 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 durante o processo de desenvolvimento. Contudo, esse planejamento não precisa ser muito detalhado.

Apenas a primeira iteração deve ter um planejamento mais detalhado, com atividades determinadas de acordo com o processo em uso e os tempos, responsáveis e recursos para a execução de cada atividade bem definidos. As demais iterações só deverão ter seu planejamento detalhado pouco antes de serem iniciadas.

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

Wazlawick, R. S. *Engenharia de Software: Conceitos e práticas*, 2ª. ed. Elsevier, 2019.

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 no detalhamento da análise e da realização do projeto para o sistema como um todo. A elaboração ocorre em iterações, com partes de *design* sendo desenvolvidas e integradas ao longo de cada iteração. Os principais objetivos dessa fase, segundo Ambler e Constantine (2000), são:

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

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

Assim, embora o Processo Unificado também trabalhe com a perspectiva de acomodação de mudanças, procura minimizar seu impacto mitigando riscos e elaborando uma arquitetura o mais próximo 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, que vão precisar de mais trabalho de análise do que de implementação neste momento, já que será necessário entender e modelar seu funcionamento. À medida que esses casos de uso são estudados e desenvolvidos, a arquitetura do sistema vai se formando. Espera-se que depois disso, a análise de casos de uso mais simples e a implementação final de todos eles não chegue a impactar muito na arquitetura.

O marco final (*milestone*) da fase de elaboração é conhecido como LCA, ou *Lifecycle 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. Essa fase é realizada em ciclos iterativos e o projeto é desenvolvido também de forma incremental, com novas funcionalidades sendo adicionadas ao sistema a cada iteração.

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

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

A fase de construção caracteriza-se pela implementação final de toda funcionalidade do sistema,

incluindo casos de uso de alta e baixa complexidade. Como os casos de uso mais complexos já foram estudados na fase anterior, o esforço de análise e *design* será menor nessa fase, ficando a maior parte do trabalho concentrada na implementação e teste dos componentes da arquitetura.

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 na 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.

Nessa 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 produção e 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 implementar novas funcionalidades necessárias (ver Capítulo 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 – IBM *Rational Unified Process*

A mais detalhada e mais antiga implementação do UP é conhecida como RUP (*IBM Rational Unified Process*). Ela foi desenvolvida principalmente por Kruchten a partir de modelos de desenvolvimento anteriores criados por Booch, Rumbaugh e Jacobson e de outras tecnologias adquiridas pela Rational, empresa na qual eles trabalhavam. A Rational tornou-se subsidiária da IBM em 2003 e, por isso, o nome do modelo passou a ser prefixado com "IBM", embora a sigla, na grande maioria das publicações ainda permaneça a mesma. A versão RUP é tão importante que algumas vezes é confundida ou considerada sinônima de UP.

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

- *Desenvolver iterativamente tendo o risco como principal fator de determinação de iterações*: é preferível conhecer todos os requisitos antes de se iniciar o desenvolvimento propriamente dito, mas em geral isso não é viável, de forma que o desenvolvimento iterativo orientado à redução de risco é bastante adequado.
- *Gerenciar requisitos*: deve-se manter controle sobre o grau de incorporação dos requisitos do cliente ao produto.
- *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.
- *Modelar software de forma visual com diagramas*: UML é indicada como padrão de modelagem de diagramas.
- *Verificar a qualidade de forma contínua*: o processo deve ser o mais orientado a testes quanto for possível. Se for o caso, utilizam-se técnicas de desenvolvimento orientado a testes, como TDD, ou *Test-Driven Development* (Beck, 2003).
- *Controlar as mudanças*: a integração deve ser contínua e gerenciada adequadamente com

o uso de um sistema de gerenciamento de configuração (Capítulo 10).

O RUP é um produto que, entre outras coisas, inclui uma base de dados com *hiperlinks* com vários artefatos e *templates* necessários para usar bem o modelo. Assim, o processo como um todo é descrito a partir de um hipertexto que associa atividades, artefatos e responsáveis em workflows de dependência. Mais do que isso, RUP é um modelo fortemente orientado a ferramentas de modelagem e gerência de ciclo de vida. Essas ferramentas, bem como a certificação para seu devido uso são fornecidas pela IBM.

5.3.1 Os Blocos de Construção do RUP

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

- *Quem*: um *papel* define um conjunto de habilidades necessário para realizar determinadas atividades.
- *O quê*: o *produto do trabalho* (*work product*) define algo produzido por alguma atividade, como diagramas, relatórios ou código funcionando, ou seja, os *artefatos*.
- *Como*: uma *atividade* descreve uma unidade de trabalho atribuída a um papel que produz determinado conjunto de artefatos.
- *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 contêineres para os quatro elementos mencionados, ou seja, cada disciplina é determinada a partir de um ou mais *workflows*, que estabelecem dependências entre as atividades, as quais são realizadas por pessoas representando papéis e produzindo ou transformando artefatos bem definidos.

5.3.1.1 Papéis

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

Papéis não são pessoas específicas nem cargos. A mesma pessoa pode exercer vários papéis em diferentes momentos, num mesmo dia, no mesmo projeto. São descritos detalhadamente diferentes papeis de analista, desenvolvedor, testador, gerente e outros.

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

Os papeis de *desenvolvedor* incluem aqueles que transformam os requisitos em produto, como por exemplo, o implementador, o integrador, o revisor de código, o arquiteto de software etc.

Os papeis de *testador* incluem várias responsabilidades que vão desde a identificação das melhores técnicas de teste (*designer* de teste) até sua execução na forma de testes manuais (testador).

Os papéis de *gerente* estão relacionados especialmente às atividades de planejamento, controle e organização do projeto. Eles incluem ainda um papel de engenheiro de processo, que aqui funciona tipicamente como o engenheiro de software conforme definido no primeiro capítulo deste livro.

Finalmente, o modelo também identifica outros papeis não classificados nos grupos anteriores

como interessados, desenvolvedor de curso, artista gráfico, etc.

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 nem muito curta (durar poucas horas) nem muito longa (durar vários dias). Sugere-se pensar em atividades que possam ser realizadas em períodos de um a três dias, porque essa duração facilita o seu acompanhamento.

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

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

Os artefatos são as entradas e também as saídas para as atividades. Em geral, 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 forma geral ao artefato.

Do ponto de vista do Processo Unificado, o artefato 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 do XP, que incentiva 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 conceder a outros o direito de fazer alterações.

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 seja iniciada na fase de concepção e refinada ao longo do projeto, sendo finalizada na fase de implantação. Também se espera que os artefatos iniciais, como requisitos e *design*, sejam finalizados mais cedo do que os artefatos finais, como os de implementação e de implantação.

5.3.1.4 Workflows

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

O RUP define três tipos de *workflow*:

- *Workflow núcleo* (*core*): define a forma geral de condução de uma dada disciplina.
- *Workflow detalhe*: apresenta um refinamento do *workflow* núcleo, indicando atividades em um nível mais detalhado, bem como artefatos de entrada e saída de cada atividade.
- *Planos de iteração*: consistem em uma instanciação do processo para uma iteração específica. Embora o RUP tenha uma descrição geral dos *workflows* para cada atividade, elas costumam ocorrer de forma diferente em projetos diferentes e até 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 apresenta outros elementos que auxiliam na aplicação do processo a um projeto. Esses elementos são:

- *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 para que não apenas pessoas acostumadas ao processo, mas também os novatos possam saber o que fazer. No RUP, os procedimentos são basicamente regras, recomendações e heurísticas sobre como executar a atividade. Elas devem focar também a descrição dos atributos de qualidade dos artefatos a serem feitos, como o que é um bom caso de uso, uma classe coesa etc.
- *Templates*: são modelos ou protótipos de artefatos e podem ser usados para criar os respectivos artefatos. Devem estar disponíveis na ferramenta usada para criar e gerenciar o artefato. Exemplos de *templat*es 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.
- *Mentores de ferramenta*: consistem em uma 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 qual descrevendo uma possível abordagem ao problema de gerenciar o desenvolvimento de um sistema. As disciplinas do UP englobam diferentes atividades e papéis relacionados por área de especialidade, e suas implementações variam de acordo com o número e a descrição dessas disciplinas.

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

- Modelagem de negócio.

- Requisitos.
- Análise e *design*.
- Implementação.
- Teste.
- Implantação.

E as disciplinas de suporte são:

- Gerenciamento de mudança e configuração;.
- Gerenciamento de projeto.
- Ambiente.

Cada uma dessas 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 para as disciplinas de projeto e suporte.

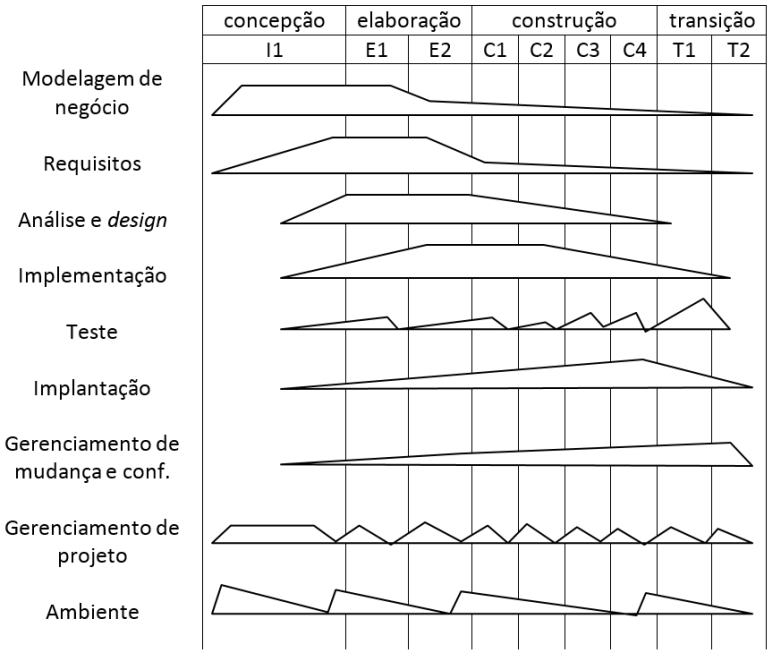

Figura 5.1 Diferentes ênfases de cada disciplina nas diferentes fases do RUP

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, visto que o RUP é um processo iterativo: suas quatro fases são sequenciais, mas as disciplinas ocorrem de forma simultânea dentro de cada fase.

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 (que pagam pelo desenvolvimento) e dos usuários finais.

Os objetivos da disciplina de *gerenciamento de projeto* são 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, o RUP não trata os seguintes aspectos:

- Gerenciamento de pessoas, incluindo contratação e treinamento.
- Gerenciamento de orçamento.
- Gerenciamento de contratos.

Tais aspectos são cobertos por uma extensão de RUP, a *Enterprise Unified Process* - EUP (Seção 5.4).

A disciplina de gerenciamento de projeto produz planos. No RUP, o planejamento 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:

- *Plano de medição* (*measurement plan*): estabelece as métricas a serem usadas para definir o andamento do projeto ao longo de sua execução (Seção 9.5).
- *Plano de gerenciamento de riscos*: detalha como os riscos serão tratados ao longo do projeto, criando atividades de mitigação e monitoramento de riscos e atribuindo responsabilidades quando necessário.
- *Lista de riscos*: 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.
- *Plano de resolução de problemas*: descreve como problemas identificados ao longo do projeto devem ser reportados, analisados e resolvidos (Seção 9.4.2).
- *Plano de aceitação de produto*: descreve como o produto será avaliado pelo cliente para verificar se satisfaz suas necessidades. O plano deve incluir a identificação dos testes de aceitação.

O plano de fase não precisa ser longo. Em geral, algumas 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* são 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.

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

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

- Uma *lista de casos de uso*, que, pelo plano da fase, devem ser finalizados na iteração corrente.
- Uma *lista dos riscos* que devem ser tratados na iteração corrente.
- Uma *lista das modificações* que devem ser incorporadas ao produto na iteração corrente (defeitos e erros encontrados ou mudanças de funcionalidade).

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

O plano de iteração costuma ser 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.

5.3.2.2 Modelagem de Negócio

A *modelagem de negócio* consiste em estudar e compreender a organização-alvo e seus processos, visto que o sistema a ser desenvolvido não será um produto isolado, mas parte orgânica do funcionamento dessa organização. Os objetivos dessa disciplina são os seguintes:

- Entender a estrutura e a dinâmica da organização-alvo na qual o software será utilizado.
- Entender os problemas atuais na organização-alvo e identificar potenciais melhorias que podem ser produzidas com o software.
- Certificar-se de que clientes, usuários e desenvolvedores tenham um conhecimento comum da organização-alvo.
- Derivar os requisitos para dar suporte a essas 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 esse investimento 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 essa 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.

Essa disciplina, porém, nem sempre é necessária. Em alguns casos, quando o objetivo do projeto é simplesmente adicionar algumas funcionalidades a 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 são introduzidas para um grupo de pessoas. Nesse 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:

- *Organograma*: pode-se querer apenas construir um organograma da organização para saber quais são os setores e as responsabilidades. Nesse caso, a modelagem de negócio em geral acontece apenas na fase de concepção.
- *Modelagem de domínio*: se o objetivo for construir aplicações para gerenciar e apresentar informação, então faz-se a modelagem de negócio com a modelagem do domínio, ou seja, um modelo de informação estático em que os *workflows* da empresa não são considerados. A modelagem de domínio costuma ser feita nas fases de concepção e elaboração.
- *Uma empresa, vários sistemas*: pode-se chegar ao ponto de desenvolver toda uma família de sistemas para uma empresa. Nesse caso, a modelagem de negócio vai tratar não apenas de um sistema, mas de vários projetos, e poderá inclusive ser tratada como um projeto à parte. Ela ajudará a descobrir os requisitos dos sistemas individuais, bem como a determinar uma arquitetura comum para a família de sistemas.
- *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 em que as especificidades das empresas seja visível.
- *Novo negócio*: se uma organização resolve iniciar um novo negócio e todo um conjunto de sistemas de informação precisa ser desenvolvido para dar suporte a ele, então um esforço

significativo de modelagem de negócio deve ser realizado. Nesse 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 nessa situação costuma ser um projeto à parte.

- *Renovação*: se uma organização resolve renovar completamente seu modo de fazer negócio, então a modelagem de negócio será um projeto à parte, executado em várias etapas: visão do novo negócio, engenharia reversa do negócio existente, engenharia direta do novo negócio e instalação do novo negócio.

Ao contrário do modelo de caso de uso de sistema, 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).

O modelo de casos de uso de negócio pode ser usado para gerar o modelo de casos de uso de sistema. Objetos de negócio ativos possivelmente serão atores ou sistemas-atores no modelo de casos de uso de sistema, e os processos de negócio possivelmente serão quebrados em processos individuais que serão identificados como casos de uso de sistema. Exemplos podem ser encontrados em Kroll e Kruchten (2003) e Wazlawick (2015). A vantagem de se 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 de 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), mas também do *design* da interface do sistema (que é uma possível solução para as necessidades apresentadas).

Os objetivos dessa disciplina são os seguintes (Kruchten, 2003):

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

Deve-se salientar que RUP preconiza o uso da UML; assim, requisitos em RUP são expressos como casos de uso de sistema.

5.3.2.4 Análise e Design

Análise implica o estudo mais aprofundado do problema. Os requisitos são detalhados e incluídos em modelos de análise, como o modelo conceitual, o de interação e o 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 (2015) apresenta muitas informações sobre como realizar essa 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, independentemente de tecnologia, e o *design*, por

sua vez, vai lidar com essas questões mais físicas.

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. Com RUP espera-se que a arquitetura candidata desenvolvida na Elaboração seja um protótipo executável do tipo pedra-fundamental. Nas iterações subsequentes, a ênfase mudará para a adição ou o refinamento de funcionalidades sobre a arquitetura, que vai se estabilizando até a fase de construção.

5.3.2.5 Implementação

Implementar é mais do que simplesmente produzir um código. Implica também organizar esse código em componentes ou pacotes, definir possíveis camadas de implementação, realizar testes de unidade e integrar o código de forma incremental.

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

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

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

5.3.2.6 Teste

A disciplina 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:

- Verificar a interação entre objetos.
- Verificar se todos os componentes foram integrados adequadamente.
- Verificar se todos os requisitos foram corretamente implementados.
- Verificar e garantir que defeitos tenham sido identificados e tratados antes da entrega do produto final.
- 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.

A disciplina de teste é detalhadamente explorada no Capítulo 13 e está fortemente relacionada com a noção de qualidade de software (Capítulo 11). Basicamente, a qualidade é uma característica que dificilmente pode ser adicionada a um sistema apenas 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 para manter essa qualidade.

5.3.2.7 Implantação

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

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

Wazlawick, R. S. *Engenharia de Software: Conceitos e práticas*, 2ª. ed. Elsevier, 2019.

Existem vários tipos de implantação de software, de acordo com a maneira como ele é distribuído. 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.

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 composto pelos seguintes artefatos:

- O software executável.
- Artefatos de instalação, como *script*s, ferramentas, arquivos, informação de licenciamento e orientações.
- Notas sobre a versão (*release notes*) descrevendo-a para o usuário final.
- Material de suporte, como manuais de operação e de manutenção do sistema.
- Material de treinamento.

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

- *Lista de materiais*: lista completa dos itens incluídos no produto.
- *Arte do produto*: 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.

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 desses artefatos e, por isso, eles devem ser rastreáveis e disponíveis para reuso futuro. No RUP, a disciplina lida com três diferentes subáreas:

- *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 os artefatos, de forma que artefatos relacionados sejam atualizados quando necessário.
- *Gerenciamento de requisições de mudança*: a área de CRM (*Change Request Management*) cuidará do controle das requisições de mudança em artefatos que produzem suas diferentes versões. 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.
- *Gerenciamento de status e medição*: requisições de mudança têm *status* como: *novo*, *atribuído*, *retido*, *concluído* e *entregue*. Elas também podem ter atributos como *causa*, *fonte*, *natureza*, *prioridade* etc. O gerenciamento de *status* coloca todos esses atributos em um sistema de informação, tal que o andamento de cada solicitação seja verificável a todo momento pelo gerente do projeto.

Essas três dimensões formam o assim chamado *Cubo CCM* (*Configuration and Change Management*), que indica a forte inter-relação entre as três subáreas.

O gerenciamento de mudança e configuração é uma tarefa árdua que talvez seja impossível de se realizar sem as ferramentas adequadas. Existem implementações livres de sistemas de controle de versões, como Git (ver *QR code*) [QRC 5.1] que são bastante populares. 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 na verdade ele 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 quase impraticável. É necessário, normalmente, que um especialista em RUP avalie o escopo do projeto e a realidade da equipe para decidir quais elementos do RUP precisam ser usados para que sua adoção seja facilitada (ver também Seção 12.5).

Em relação à disciplina de ambiente, convém mencionar que ela não se refere ao produto, como as outras, mas à engenharia do processo em si, visando o seu aprimoramento.

5.4 EUP – *Enterprise Unified Process*

O *Enterprise Unified Process*, ou EUP, [QRC 5.2] foi originalmente definido como uma extensão de RUP por Ambler e Constantine em 1999 e, posteriormente, refinado por Ambler, Nalbone e Vizdos (2005). Desde 2013 o modelo tem sido evoluído de forma a ser mais uma complementação do modelo DAD (Seção 5.7) do que de RUP propriamente dito[1]

[1]No momento em que este livro está sendo escrito, um recado na página oficial de EUP (enterpriseunifiedprocess.com/. Consultado em 19 de setembro de 2018) avisa que em breve a documentação do processo será atualizada para refletir a evolução de EUP para um modelo baseado em DAD (*Disciplined Agile Delivery*).

.

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. A principal motivação para sua proposição foi o fato de que RUP considera o projeto de desenvolvimento como um processo fechado em si próprio, sem levar em conta que projetos são desenvolvidos no contexto de empresas e que as atividades das empresas não se limitam apenas ao desenvolvimento pontual de projetos.

Assim, EUP considera que o ciclo de vida do produto de software não se limita às quatro fases originais do processo unificado. Ele adiciona, então duas novas fases após a transição, que são a fase de *produção* (*production*) e *desativação* (*retirement*). Durante a produção o sistema estará em uso e possivelmente em manutenção e evolução. Já a fase de desativação indica os processos necessários para tirar um sistema de uso, possivelmente substituindo-o por um novo.

Além dessas duas fases, várias novas disciplinas relacionadas às rotinas de empresa foram adicionadas. Essas disciplinas contemplam os aspectos da empresa que transcendem ao escopo dos projetos individuais.

Outra nova disciplina de suporte adicionada pelo EUP é a disciplina de *operação e suporte*, considerada fundamental durante a fase de produção. Assim, as oito novas disciplinas adicionadas

às do RUP pelo EUP são:

- *Operação e suporte*: esta disciplina indica como o sistema deve ser mantido em operação, como as necessidades dos usuários devem ser atendidas e como devem ser preparados e executados os planos de recuperação de desastre.
- *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, ou seja, da empresa cliente. A modelagem de negócio de empresa do EUP porém consiste na realização de atividades que buscam entender e aprimorar o próprio negócio da empresa desenvolvedora ou prestadora de serviços.
- *Gerenciamento de portfólio*: um portfólio é uma coleção de projetos de software em andamento e concluídos. A ideia é inicialmente identificar projetos que a empresa já tem em andamento ou concluídos e depois preparar um plano para manter e evoluir este portfólio.
- *Arquitetura de empresa*: a arquitetura de empresa define como ela trabalha. Essa disciplina é especialmente útil se a empresa possui muitos produtos de software. Deve haver consistência na forma como eles são desenvolvidos, negociados e entregues. Uma arquitetura de empresa eficiente promove a consistência entre os diferentes projetos e sistemas desenvolvidos pela empresa. Linhas de produto de software são um exemplo de prática que pode ser empregada em uma arquitetura de empresa. Mais detalhes em McGovern et al. (2003).
- *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. Poucas empresas conseguem realmente realizar reuso efetivo; para que isso ocorra é necessário olhar os projetos da empresa como um todo, e então se preparar para reuso estratégico e não apenas incidental.
- *Gerenciamento de pessoas*: essa disciplina define uma abordagem para gerenciamento de recursos humanos da área de tecnologia de informação. É preciso gerenciar o pessoal, contratar, demitir, substituir, alocar pessoas a projetos e investir em seu crescimento.
- *Administração de empresa*: essa disciplina define como a empresa cria, mantém, gerencia e entrega produtos e serviços de forma segura. Assim, os papeis ligados a esta disciplina são o de administrador de rede, de instalações, de segurança e de informações.
- *Melhoria de processo de software*: essa disciplina trata da adequação e evolução do processo de software para a empresa como um todo, não apenas da adequação do processo a cada projeto (veja também a Seção 12.5).

As principais razões que podem ser citadas para a adoção deste modelo são relacionadas a empresas que tem dificuldades em gerenciar seu próprio negócio e em manter seus produtos em funcionamento e evolução.

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

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

O modelo 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.).

O RUP-SE é especialmente adequado a projetos:

- Que são grandes o suficiente para comportar várias equipes de desenvolvimento trabalhando em paralelo.
- Que necessitam de desenvolvimento concorrente de hardware e software.
- Cuja arquitetura é impactada por questões relativas à implantação.
- Que incluem a reengenharia de uma infraestrutura de tecnologia de informação para dar suporte à evolução do negócio.

O *framework* é disponibilizado como um *plugin*, ou anexo, ao modelo RUP original (ver *QR code*). Ele basicamente introduz novos artefatos, bem como modificações nos papeis e nas disciplinas RUP para a criação destes artefatos (Rational Software, 2001). [QRC 5.3] Em especial, é adicionado o papel de engenheiro de sistemas, cuja responsabilidade é o projeto e especificação de hardware, bem como a implantação do sistema completo.

Existe uma preocupação especial com a arquitetura do sistema, que deve ser cuidadosamente considerada de acordo com pelo menos cinco pontos de vista:

- *Empresa*: relação entre os recursos da empresa e o sistema. Diz respeito às atividades dos trabalhadores, instalação e suporte logístico.
- *Computação*: decomposição lógica do sistema como um conjunto coerente de subsistemas UML que colaboram para gerar o comportamento do sistema. Diz respeito a funcionalidade adequada para realizar os casos de uso, sistema extensível e manutenível, reuso interno e boa coesão e conectividade.
- *Engenharia*: distribuição de recursos para suportar as funcionalidades. Diz respeito às características físicas do sistema serem adequadas para hospedar as funcionalidades e satisfazer os requisitos suplementares (requisitos não funcionais gerais do sistema).
- *Informação*: dados gerenciados pelo sistema. Diz respeito ao sistema ser capaz de armazenar os dados e ter capacidade suficiente para fornecer acesso adequado a estes.
- *Processo*: *threads* de controle que realizam os elementos computacionais. Diz respeito ao sistema ter particionamento de processamento suficiente para suportar as necessidades de concorrência e confiabilidade.

No mais, o modelo se concentra em explicar como esses diferentes pontos de vista podem ser capturados e representados por diferentes diagramas da UML ao longo do processo de desenvolvimento.

5.6 OUM – Oracle Unified Method

O *Oracle Unified Method*, ou OUM (Oracle, 2009), 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 5.2 apresenta esquematicamente o ciclo de vida OUM, que é composto por cinco fases

e 14 disciplinas (Oracle, 2007).

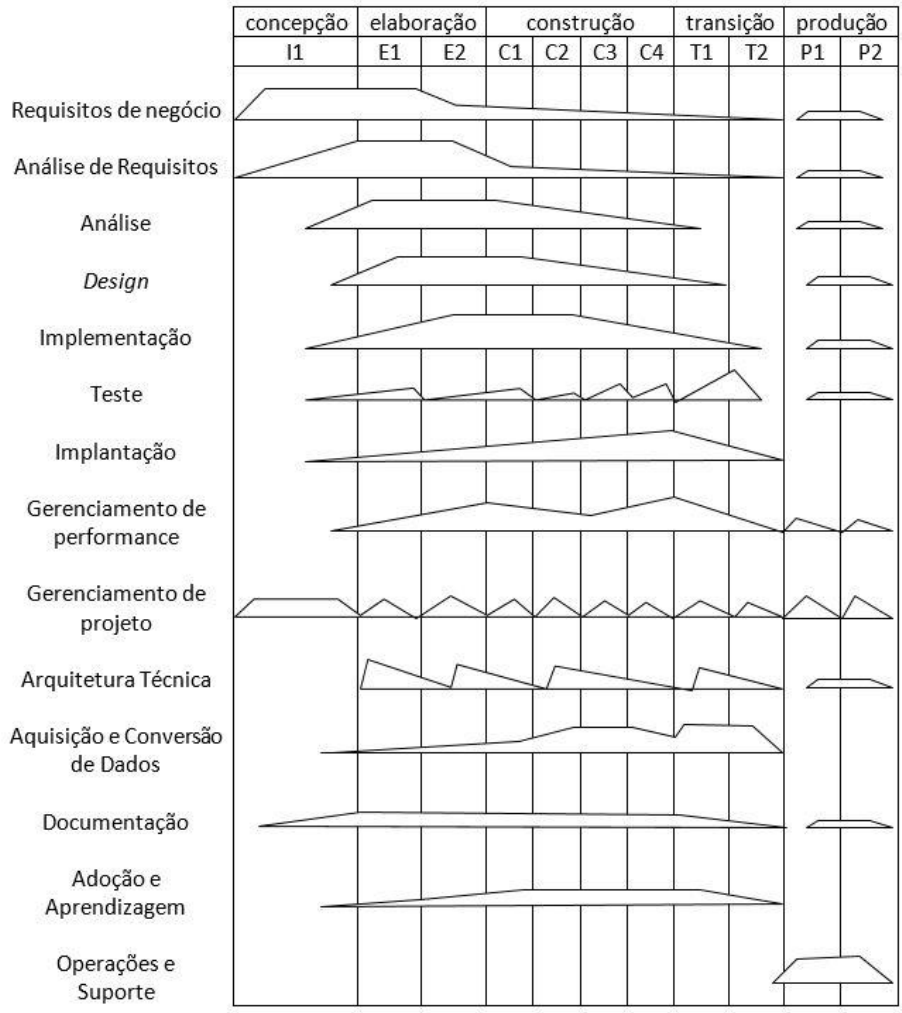

Figura 5.2 Ciclo de vida OUM

Em relação ao RUP, observa-se a introdução da fase de produção, com o mesmo significado que tem no EUP, e também a eliminação das disciplinas de ambiente e gerenciamento de configuração. Além disso, a disciplina RUP de Análise e *Design* é dividida em duas partes. Novas disciplinas são também acrescentadas:

- Gerenciamento de performance.
- Arquitetura técnica.
- Aquisição e conversão de dados.
- Documentação.
- Adoção e aprendizagem.
- Operações e suporte.

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

- *Requisitos de negócio*: os requisitos de negócio da nova aplicação ou de sua evolução são identificados e modelados. As principais saídas dessa disciplina são objetivos e metas de negócio,

e a lista de requisitos funcionais e não funcionais.

- *Análise de requisitos*: os requisitos são organizados em um modelo de casos de uso. As principais saídas são o modelo de caso de uso, protótipos de interface e descrição em alto nível da arquitetura do sistema.
- *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 o modelo conceitual e a revisão da arquitetura de sistema.
- *Design*: o modelo de análise é instanciado em um modelo de *design* derivado da arquitetura inicial, que, além de informações sobre classes e funcionalidades, vai indicar 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*).
- *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.
- *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.
- *Gerenciamento de performance*: são atividades integradas de garantia de qualidade da aplicação em relação aos requisitos suplementares de *performance*.
- *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 os sistemas com grande número de acessos.
- *Aquisição e conversão de dados*: em geral, novos sistemas substituirão outros, sejam manuais, sejam informatizados, e seus dados precisarão ser adquiridos ou convertidos de alguma forma. Normalmente, não é um processo simples, por isso, OUM acrescenta essa disciplina.
- *Documentação*: as ferramentas Oracle dão suporte à produção de documentação-chave ao longo do projeto.
- *Adoção e aprendizagem*: focam o uso e a aceitação de novas práticas associadas às novas ferramentas ou à evolução das antigas.
- *Transição*: inclui as atividades necessárias para a instalação do produto.
- *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.

5.7 DAD – *Disciplined Agile Delivery*

DAD ou *Disciplined Agile Delivery* é considerada o fundamento de um *framework* mais amplo denominado *Disciplined Agile Framework*. Ela é definida pelos seus autores (Ambler & Line, 2012) como uma abordagem ágil híbrida centrada em pessoas e orientada a aprendizagem. Além disso, considera-se que ela define um ciclo de vida que mitiga risco e entrega valor, além de ser dirigida por objetivos, escalável e sensível à empresa (*enterprise aware*).

DAD vem sendo desenvolvida por Scott Ambler desde 2009. Este autor anteriormente desenvolveu o modelo AUP (*Agile Unified Process*), que desde 2006 foi descontinuado por ele, embora ainda seja usado em vários lugares. Enquanto AUP era explicitamente uma implementação ágil do Processo Unificado, o mesmo já não se pode dizer de DAD. Segundo Ambler, DAD é uma abordagem híbrida que estende *Scrum* com estratégias consolidadas de Modelagem Ágil (AM), *eXtreme Programming* (XP), Processo Unificado (UP), *Kanban*, *Lean* e vários outros métodos.

Uma das principais argumentações refere-se às lacunas que modelos como *Scrum* deixam em relação ao desenvolvimento de software. *Scrum* aborda muito bem o gerenciamento do processo, mas é completamente omisso em relação às estratégias de modelagem e produção de código. É nestas lacunas que DAD coloca sua atenção e apresenta sugestões.

DAD propõe dez papeis, que são divididos em primários e secundários. Os papeis primários são:

- Líder de equipe.
- *Product owner*.
- *Architecture owner*.
- Membro de equipe.
- Interessado.

Já os papeis secundários são:

- Especialista.
- Testador independente.
- Especialista de domínio.
- Especialista técnico.
- Integrador.

Considera-se que os papeis primários são aqueles que atuam no projeto praticamente todo o tempo enquanto que os papeis secundários são atuantes de forma mais esporádica.

Em relação a um possível questionamento sobre o número de papeis, já que *Scrum* tem apenas três, Ambler justifica dizendo que como DAD aborda vários aspectos do desenvolvimento sobre os quais Scrum é omisso, é natural que mais papeis apareçam.

5.8 *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).

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 e passou a ser um de seus projetos.

O *OpenUP* aceita, embora de forma simplificada, a maioria dos princípios do Processo Unificado. Porém, é um método independente de ferramenta (embora a plataforma Eclipse forneça várias ferramentas úteis para ele) e de baixa cerimônia, ou seja, não são exigidos grande precisão e detalhes nos documentos.

O processo se baseia em quatro princípios:

- *Colaboração* para alinhar interesses e compartilhar entendimentos;
- *Evolução* para continuamente obter *feedback* e melhorar;
- *Balanceamento de* prioridades que competem entre si de forma a maximizar valor para os interessados;

- *Foco* na articulação da arquitetura.

O ciclo de vida também é estruturado em quatro fases, como no UP. Essas fases são igualmente divididas em iterações, mas aqui as equipes se auto-organizam para planejar cada uma delas. O esforço pessoal é organizado em micro incrementos, que representam pequenas unidades de trabalho que produzem um ritmo mais fino e mensurável para o projeto.

Em relação ao RUP, a maioria das práticas opcionais foi eliminada 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. Dessa 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.