Pós-Graduação em Ciência da Computação

"RUP-pe: Uma Metodologia Ágil, Baseada no RUP e no TSP, Para Pequenas Equipes"

Por

Joaquim Pedro C. de Oliveira

Dissertação de Mestrado

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

RECIFE, SETEMBRO/2003

Joaquim Pedro Carvalho de Oliveira

RUP-pe: Uma Metodologia Ágil, Baseada no RUP e no TSP, Para Pequenas Equipes

Dissertação apresentada à Coordenação da Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Orientador:
Prof. Alexandre Marcos Lins de Vasconcelos

Recife, setembro de 2003

Agradecimentos

Aos meus pais, Joaquim e Dulce, pelo amor, carinho, amizade e atenção que sempre me deram, e por me incentivarem, desde o começo, a traçar meus próprios caminhos. Amo vocês.

Às minhas irmãs, Cecilia, Camila e Milena, por estarem sempre ao meu lado, muitas vezes sem nem perceber, e me deixarem sempre com saudade de casa. Também amo vocês.

Aos meus avôs, João Baptista e Waldemiro, por me ensinarem a trabalhar duro pelo que se deseja. A minha avó Dulce, por me ensinar o quão importante são a simplicidade e a humildade. A minha avó Idelzuith, por me ensinar que "juízo é uma coisa que só serve para atrapalhar a vida da gente".

À Thatiane, por estar sempre ao meu lado nos momentos difíceis e não me deixar abrir mão da realização de um sonho. Você sempre será meu anjo da guarda.

Aos amigos, quase irmãos, Ciro e Arthur, pela paciência em dividir o mesmo apartamento todo este tempo, com companheirismo e compreensão, e pelas discussões produtivas (ou não!), na hora do jantar. E, lógico, pelas memoráveis partidas de Counter-Strike e Age of Empires 2! *Roger that*!

Ao professor Alexandre Vasconcelos, pela orientação deste trabalho, pelas sugestões valiosas e por ser um grande incentivador em diversos momentos difíceis. Muito obrigado!

Aos amigos das listas "NCODEB" e "amigos-recife", que ajudaram a amenizar a saudade de casa e tornar a nova vida mais feliz. Vocês foram essenciais nesta longa caminhada!

Aos amigos do peito Misa, Marcelo "magrão", Rafael, Carla, e todos aqueles que tornaram uma grande festa os últimos fins-de-semana de escrita deste trabalho. Equipe, *brothers*! E à minha "prima" Camila Caram pela ajuda com a análise dos dados e dúvidas de estatística. Querida prima, você é 10!

Aos professores Hermano Perrelli e Jorge Fernandes, pela avaliação deste trabalho.

A todos que contribuíram para a aplicação dos questionários, seja respondendo-os ou distribuindo-os, mas que não puderam ter seus nomes citados neste trabalho.

Resumo

Pesquisas recentes têm mostrado que projetos de software de curta duração e que utilizam equipes pequenas possuem maiores chances de sucesso. Entretanto, a escolha da metodologia a ser utilizada neste tipo de projeto deve levar em consideração suas características, como a comunicação facilitada e o menor grau de formalismo nos artefatos. Para se utilizar metodologias de desenvolvimento pesadas, como o RUP e o OPEN, deve-se fazer uma adaptação da metodologia escolhida, pois elas se propõem servir a uma grande diversidade de projetos, possuindo um grande número de atividades, muitas vezes desnecessárias. Apesar do detalhamento de suas atividades ser útil para sua adoção, isto torna a adaptação da metodologia uma atividade complexa e custosa. Por outro lado, a adoção de metodologias ágeis, como XP e Scrum, muitas vezes não é suficiente. Por adotarem uma abordagem mais informal para o desenvolvimento de software, estas metodologias muitas vezes carecem de descrições mais detalhadas e deixam atividades importantes em segundo plano, como por exemplo, gerenciamento de riscos e planejamento de recursos. Caso se deseje estar alinhado ou de acordo com um modelo de qualidade como o *Capability Maturity Model for Software* (SW-CMM), esta informalidade é um obstáculo ainda maior. Todavia, várias práticas propostas por estas metodologias têm se mostrado eficazes em projetos pequenos. Como exemplos, temos a integração contínua e o uso de iterações de curta duração. Neste trabalho, apresentamos o RUP-pe, uma metodologia ágil, voltada para equipes de até 15 desenvolvedores. Ele foi elaborado com base no RUP e incorpora práticas de diversas metodologias ágeis. Além disto, também foi definido com base no TSP, um processo de software completamente alinhado com o SW-CMM. O objetivo do RUP-pe é ser uma metodologia adequada às pequenas equipes, sem ser excessivamente informal, e alinhada, desde sua definição, com um modelo de qualidade reconhecido mundialmente. São apresentadas as disciplinas Implementação e Gerenciamento de Projeto, e um *website* que possui todo o detalhamento das atividades propostas, servindo de guia para as equipes que desejem utilizá-lo.

Palavras-chave: Pequenas equipes, Metodologias Ágeis, RUP, TSP.

Abstract

Recent research has shown that software projects that have short duration and use small teams have greater chances of success. However, the choice of a methodology to be used in this kind of project should take into account the project's characteristics, such as the easier communication and the lower degree of formality of the artifacts. In order to use heavyweight development methodologies, like RUP and OPEN, a tailoring of the chosen methodology is necessary, because they are designed to fit a large variety of projects, and have a large number of activities that are, in most of cases, unnecessary. Although the high degree of detail of these activities is useful to its adoption, it turns the methodology tailoring into a complex and costly activity. On the other side, the adoption of agile methodologies, like XP and Scrum, is not sufficient in most of the cases. Since they adopt a more informal approach to software development, these methodologies lack, many times, more detailed descriptions and let important activities in second place, for example, risk management and resources planning. If it is desired an alignment with a quality model, like the Capability Maturity Model for Software (SW-CMM), this informality is an even greater obstacle. However, many practices proposed by theses methodologies have proven being effective in small projects. As examples, we have continuous integration and the use of small length iterations. In this work we present RUP-pe, an agile methodology, designed for teams up to 15 developers. It was elaborated based on RUP and incorporates practices of the agile methodologies. In addition to that, it was also defined based on TSP, a software process completely aligned with SW-CMM. Its objective is being a methodology suitable for small teams, without being excessively informal, and aligned with a worldwide recognized quality model since its definition. We present the disciplines Implementation and Project Management, and a website that contains all the details of the proposed activities, serving as a guide to the teams that want to adopt it.

Keywords: Small teams, Agile Methodologies, RUP, TSP.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1: Introdução

1.1 Motivação

Em virtude da popularização e do barateamento dos computadores pessoais, e da expansão da Internet, a indústria de software vem experimentando um grande crescimento nas últimas décadas. Como conseqüência, o software produzido é cada vez mais complexo e a exigência por qualidade é cada vez maior, por causa da alta competitividade.

Boehm [1] aponta que a tendência é que o desenvolvimento de software se torne ainda mais árduo, devido à crescente complexidade do produto final, causada pela disseminação das redes e da internet, pelo surgimento de softwares orientados a agentes e pelo aumento da complexidade dos próprios problemas, e pela velocidade cada vez maior com que o software deve ser produzido, em virtude das questões de mercado.

Entretanto, hoje em dia, ainda é grande o número de projetos que são cancelados ou não obtêm sucesso por cumprir somente parte dos requisitos definidos inicialmente. De acordo com estudos do Standish Group [2], entre 30.000 projetos de diversas companhias norte-americanas, apenas 28% obtiveram sucesso no período 2000/2001, ou seja, foram completados no tempo e no custo previstos, e possuíam todas as funcionalidades inicialmente planejadas. Apesar do aumento de sucessos em relação ao ano de 1994, onde apenas 16% dos projetos conseguiram êxito, este ainda é um percentual bastante reduzido.

Ainda segundo o levantamento do Standish Group, os principais fatores que contribuem para o sucesso dos projetos são, em ordem de importância:

- Suporte executivo;
- Envolvimento do usuário;
- Gerente de projeto experiente;
- Objetivos de negócio claros;
- Escopo reduzido, ou seja, o projeto deve ter objetivos bem definidos e marcos de acompanhamento bem próximos uns dos outros;

- Uma infra-estrutura padrão para o software, para que a equipe possa se concentrar nas regras de negócio em vez de se concentrar na tecnologia;
- Requisitos básicos, ou seja, os requisitos fundamentais do sistema, estáveis;
- Metodologia[1] formal, ou seja, uma metodologia definida, documentada e conhecida por todos;
- Estimativas confiáveis.

Além disto, foi observado que o tipo de projeto que mais possui chances de sucesso é o chamado micro-projeto [2] [3], aquele cuja duração é de, no máximo, 3 meses e cujo custo não ultrapassa 250.000 dólares. Um micro-projeto contém quatro elementos básicos [2]: processo de desenvolvimento iterativo, infra-estrutura padrão de software, gerenciamento baseado em colaboração, e testes e inspeções automatizados. Devido ao seu pequeno porte, é razoável dizer que estes projetos contarão, na maioria dos casos, com equipes de desenvolvimento de, no máximo, 15 desenvolvedores.

Entretanto, a escolha de uma metodologia para um projeto deste tipo deve ser feita com cautela, pois é sabido que não existe uma metodologia que sirva a todos os tipos de situações. Cada projeto possui características próprias, que devem ser levadas em conta na hora da escolha ou da definição da metodologia a ser utilizada [4]. A adoção de uma metodologia que não seja adequada traz prejuízos para a produtividade da equipe. Cockburn [5] aponta, por exemplo, que um aumento relativamente pequeno no tamanho da metodologia, ou seja, no número de atividades, artefatos e papéis envolvidos, leva a um aumento considerável no custo do projeto, devido ao tempo gasto em atividades de coordenação e em artefatos que não irão contribuir diretamente para o progresso do projeto.

O tamanho da equipe é uma dessas características e influencia bastante as atividades a serem executadas. Um menor número de participantes facilita, por exemplo, a comunicação entre os membros da equipe, permitindo um menor grau de formalidade nos artefatos a serem produzidos. Além disto, a facilidade de coordenar as atividades entres os membros é maior, já que existem menos pessoas e menos atividades a serem executadas.

Existe hoje uma grande diversidade de metodologias de desenvolvimento. De um lado, temos metodologias mais detalhadas, tais como OPEN [6] e o Rational Unified Process [7], que foram concebidas para servirem a diferentes tipos de projetos. Nestas metodologias, é forte a ênfase na produção de artefatos que irão documentar

[1] Apesar de não serem totalmente sinônimos, neste trabalho não será feita qualquer distinção entre os termos "metodologia" e "processo".

detalhadamente o desenvolvimento do sistema em questão, e no planejamento detalhado das atividades, mesmo que feito de forma iterativa. Este tipo de metodologia será referenciado, ao longo deste trabalho, pelo termo "metodologia pesada", de acordo com a terminologia proposta por Cockburn em [5].

De outro lado temos metodologias mais simplificadas, que defendem uma abordagem mais direta para o desenvolvimento de software, onde a capacidade de reagir rapidamente a mudanças e a comunicação constante entre os envolvidos no projeto, inclusive o cliente, são considerados mais importantes do que uma documentação extensa e um planejamento detalhado. Estas metodologias são conhecidas como Metodologias Ágeis ou Metodologias Leves. Elas são voltadas a tipos específicos de projetos: aqueles que possuem equipes pequenas e cujo produto final não seja considerado um sistema crítico. Como exemplo, podemos citar Extreme Programming [8] e Scrum [9].

As Metodologias Ágeis parecem, em um primeiro momento, mais adequadas às pequenas equipes. Entretanto, devido ao seu alto grau de informalidade, muitas atividades importantes ficam subentendidas ou não são sequer citadas, como o monitoramento de riscos e o planejamento de recursos e de infra-estrutura necessária para o projeto. Isto faz com que estas atividades acabem sendo deixadas em segundo plano ou até mesmo esquecidas, caso a equipe não seja altamente capacitada.

As metodologias mais detalhadas, por sua vez, possuem atividades e passos que muitas vezes podem ser simplificados ou eliminados, caso a equipe tenha poucos membros. Adotar uma metodologia como esta sem fazer uma adaptação é trazer uma grande sobrecarga para o processo, ou seja, será gasto muito esforço em atividades que irão agregar pouco valor ao projeto como um todo. Adaptar, para uma determinada situação, uma metodologia detalhada não é uma tarefa fácil, pois é necessário um conhecimento da metodologia completa para que se possa escolher que atividades serão realmente úteis. Devido ao grande escopo e a ao alto grau de detalhamento destas metodologias, esta é uma tarefa bastante trabalhosa e complexa.

Assim, para as pequenas equipes, o ideal é encontrar um meio-termo entre essas duas abordagens: uma metodologia que possua o nível de detalhe adequado para guiar a execução de suas atividades e que contemple as boas práticas de desenvolvimento, mas que traga somente o necessário para este tipo de cenário, sem tornar o processo de desenvolvimento pesado e burocrático.

1.2 Escopo do Trabalho e Contribuições Esperadas

Neste trabalho propomos uma metodologia de desenvolvimento ágil voltada para equipes de até 15 desenvolvedores: o Rational Unified Process para pequenas equipes – RUP-pe. Esta metodologia foi definida com base no RUP, compartilhando seu ciclo de vida e a mesma maneira de descrever as atividades, artefatos e responsabilidades, e incorpora várias práticas das principais metodologias ágeis que têm se mostrado efetivas na comunidade de engenharia de software.

Além disto, o RUP-pe também foi definido com base no *Team Software Process* (TSP) [10], um processo de desenvolvimento proposto pelo Instituto de Engenharia de Software (*Software Engineering Institute* – SEI) da Universidade de Carnegie-Mellon, voltado para a formação de equipes de desenvolvimentos que sejam coesas e altamente produtivas, completamente alinhado com o *Capability Maturity Model for Software* (SW-CMM) [11]. Assim, o RUP-pe também está alinhado, desde sua definição, com um modelo de qualidade bastante reconhecido.

Por ter sido concebido para ser uma metodologia mais simplificada, o RUP-pe não se aplica a todos os tipos de projeto. Seu escopo são os projetos que possuem as seguintes características, seguindo o Modelo de Caracterização de Projetos proposto por Coelho [22]:

- Tamanho da equipe: entre 7 e 15 pessoas, tamanho de equipe considerado pequeno pelo Modelo de Caracterização de Projetos. Como já mencionado anteriormente, é razoável supor que este é o tamanho da equipe dos micro-projetos;
- Distribuição geográfica da equipe: mesma sala;
- Criticidade do projeto: prejuízos moderados, perdas facilmente recuperáveis;
- Tamanho do projeto: projetos de até R$ 500.000,00.

Por limitações de tempo, foram definidas somente as atividades relativas à implementação do produto e ao gerenciamento do projeto. As justificativas para esta escolha são apresentadas no Capítulo 5.

Como principal contribuição deste trabalho esperamos obter uma metodologia de fácil uso, que seja voltada para a realidade dos tipos de projeto que têm se mostrado mais adequados a cumprirem seus objetivos, ou seja, aqueles de menor duração e cuja equipe possua, no máximo 15 pessoas.

1.3 Estrutura da Dissertação

Além deste capítulo inicial, este trabalho possui mais 6 capítulos e 2 apêndices, organizados da seguinte forma:

- No Capítulo 2 é apresentado o Rational Unified Process (RUP), um *framework* para a definição de metodologias de desenvolvimento bastante difundido e estabelecido na comunidade de desenvolvimento de software. São apresentados seu ciclo de vida, a maneira como as atividades são organizadas e como o RUP pode ser adaptado para pequenas equipes;
- No Capítulo 3 é apresentado o movimento conhecido como Metodologias Ágeis, que propõe metodologias de desenvolvimento menos burocráticas e que respondam mais rapidamente às mudanças constantes de um projeto de software. Além disto, são apresentadas as 5 metodologias deste movimento que mais têm ganhado destaque;
- No Capítulo 4 é apresentado o *Team Software Process*, com suas fases, atividades e papéis. Também é descrito como o TSP auxilia a obtenção do SW-CMM;
- No Capítulo 5 é apresentado o RUP-pe, nossa proposta de metodologia para pequenas equipes. Primeiramente, é apresentada uma análise crítica da adequação do RUP em relação às pequenas equipes. Em seguida, são descritos os fluxos de atividades referentes à implementação do produto e ao gerenciamento do projeto, com a apresentação de cada atividade, bem como as mudanças incorporadas;
- No Capítulo 6 é apresentada uma análise crítica do RUP-pe, com o objetivo de validar a viabilidade da nossa proposta. Para isto, as mudanças sugeridas e práticas incorporadas à nossa metodologia foram submetidas à análise de diversos desenvolvedores;
- No Capítulo 7 é apresentada a conclusão deste trabalho, com as principais contribuições obtidas, as dificuldades encontradas, e os trabalhos relacionados existentes na literatura. Além disto, são apresentados os possíveis trabalhos futuros decorrentes da nossa proposta;

- No Apêndice A são apresentados alguns exemplos do *site* elaborado para descrever o RUP-pe [12], o qual serve de referência para as equipes que desejarem utilizá-lo;
- No Apêndice B são apresentados os modelos de questionários distribuídos para a realização da análise crítica da metodologia.

Capítulo 2: Rational Unified Process

2.1 Visão Geral

O *Rational Unified Process* (RUP) [7] é um processo que busca solucionar o problema clássico da engenharia de software: garantir a produção de software de alta qualidade dentro do prazo e do custo planejados, que atenda aos requisitos dos usuários finais. Ele suporta todo o ciclo de vida do desenvolvimento de software, desde o estudo de sua viabilidade até a entrega ao usuário final, e contém uma descrição detalhada das várias atividades que devem ser executadas para que isto seja possível.

Por ser um *framework* de processos, ou seja, um processo genérico, que abrange uma grande diversidade de projetos, o RUP não deve ser visto como um processo fixo, que deve ser seguido à risca. As atividades descritas no RUP devem ser adaptadas de acordo com as características da organização e do projeto.

Ele tem sua origem no *Rational Objectory Process*, resultado da união do processo Objectory [13], de Ivar Jacobson, com a abordagem da Rational para desenvolvimento de software, quando a Rational Corporation se fundiu com a Objectory AB. De lá para cá, o RUP tem sofrido várias evoluções, como a incorporação de práticas dos métodos BOOCH [14] e OMT [15], de Grady Booch e James Rumbaugh, respectivamente. O RUP é comercializado como um produto pela Rational e atualmente encontra-se na versão 2002 [16], a qual será adotada neste trabalho.

Segundo Krutchen [7], o RUP captura as "melhores práticas" do desenvolvimento de software, ou seja, práticas que têm sido identificadas como responsáveis pelo sucesso dos projetos na indústria de software. Estas práticas são:

- **Desenvolver iterativamente:** o produto é desenvolvido em uma série de ciclos, denominados iterações. Em cada ciclo, uma parte das funcionalidades do produto é implementada passando-se por todas as etapas tradicionais, de requisitos a testes, e produzindo ao final um código executável. Cada iteração melhora o produto da iteração anterior, implementando mais funcionalidades ou melhorando as já existentes. A idéia é que os riscos sejam identificados o quanto antes, para que possam ser tratados apropriadamente. O desenvolvimento iterativo e incremental traz vários benefícios, como a mitigação dos riscos, melhor acomodação das mudanças no projeto, maior visibilidade do progresso, *feedback* e envolvimento do usuário desde o início, levando a um refinamento do sistema que atenderá melhor as suas reais necessidades. Além disto, dá a oportunidade de se melhorar aspectos do desenvolvimento com base nos erros e acertos das iterações anteriores [17].
- **Gerenciar requisitos:** o gerenciamento dos requisitos, segundo o RUP, provê uma abordagem sistemática para elicitar, organizar, comunicar e gerenciar as mudanças de requisitos de um projeto de software. Para isto, é definido o escopo do projeto, entendendo-se o que o cliente quer e até onde vão os limites do sistema, do ponto de vista da funcionalidade. Este escopo é gerenciado durante todo o projeto, controlando-se as mudanças nos requisitos e seus impactos, e detalhando os requisitos já existentes, garantindo-se que tudo o que foi acordado será cumprido.
- **Usar arquitetura baseada em componentes:** o sistema é desenvolvido com base no conceito de uma arquitetura robusta, formada por componentes, os quais são unidades coesivas de código que fornecem uma interface bem definida e um forte encapsulamento do seu conteúdo, sendo facilmente substituíveis. O uso de uma arquitetura baseada em componentes reduz a complexidade da solução e, possivelmente, o tempo de desenvolvimento, já que se pode reusar componentes já desenvolvidos anteriormente ou adquirir componentes já prontos.

- **Modelar visualmente:** um modelo é uma representação simplificada do sistema, o que permite lidar melhor com sua complexidade e, conseqüentemente, obter um melhor entendimento do problema a ser resolvido. O RUP adota a Linguagem Unificada de Modelagem [18] (*Unified Modeling Language* – UML) como forma de descrever o sistema, utilizando os diversos modelos fornecidos por ela. A utilização de uma linguagem visual facilita a comunicação e o entendimento entre os membros da equipe;
- **Verificar continuamente a qualidade:** a qualidade é vista de dois modos: qualidade do processo e do produto. A qualidade do processo está ligada à qualidade dos artefatos que são produzidos, e é avaliada nas diversas disciplinas, durante as atividades de revisão. Um exemplo disto é a atividade "Revisar Código". A qualidade do produto é verificada especialmente na disciplina Testes, onde as funcionalidades do sistema são avaliadas em relação aos requisitos que deveriam ser implementados.
- **Gerenciar as mudanças:** mudanças são constantes durante o desenvolvimento de um software. Por exemplo, o cliente muda de opinião sobre uma funcionalidade, ou artefatos precisam ser atualizados para refletir uma melhoria identificada pela equipe. Todavia, se estas modificações não forem feitas de maneira controlada, aparecerão problemas de retrabalho, falta de sincronia entre os membros da equipe e inconsistência entre os artefatos. O Gerenciamento de Mudanças fornece uma abordagem sistemática para o controle destas alterações necessárias em um projeto de software.

O RUP é descrito a partir de alguns conceitos básicos, cujos relacionamentos descrevem a estrutura do processo. Os principais conceitos são apresentados a seguir:

- **Disciplina:** é uma coleção de atividades relacionadas a um determinado assunto, por exemplo, testes. As atividades são agrupadas em disciplinas para organizar o processo e facilitar seu entendimento;
- **Fluxo de Trabalho:** uma seqüência de atividades de uma disciplina que produz um resultado de valor para o projeto. Um fluxo de

trabalho descreve o processo, indicando, de maneira macro, a ordem em que as atividades devem ser executadas;

- **Detalhes do Fluxo:** os detalhes do fluxo, ou sub-fluxos, auxiliam a descrição dos fluxos de trabalho. Eles mostram conjuntos de atividades que geralmente são executadas em conjunto, indicando os papéis e artefatos envolvidos;
- **Papel:** um papel define o comportamento e o conjunto de responsabilidades de um indivíduo ou de um conjunto de indivíduos que trabalham como uma equipe. Os papéis não são necessariamente indivíduos: um mesmo membro da equipe pode desempenhar vários papéis e um papel pode ser desempenhado por várias pessoas;
- **Atividade:** uma atividade descreve os passos a serem seguidos por um papel para prover resultados significativos para o projeto. Em geral isto consiste em produzir ou atualizar um artefato. A atividade é considerada a unidade básica de trabalho, tendo duração de poucas horas a poucos dias. Uma mesma atividade pode ser executada várias vezes em um mesmo artefato, principalmente para detalhá-lo, no decorrer de uma série de iterações;
- **Artefato:** um artefato é o produto de trabalho do processo. As atividades produzem artefatos, que irão servir de entrada para outras atividades. Documentos, modelos, o código-fonte e uma versão executável do sistema são exemplos de artefatos. O RUP desencoraja a produção de artefatos em papel. O ideal é que sejam produzidos e mantidos em ferramentas apropriadas e disponibilizados como documentos quando necessário.

2.2 Ciclo de Vida

No RUP, o ciclo de vida do projeto é dividido em quatro fases: Concepção, Elaboração, Construção e Transição. Cada fase possui objetivos específicos e um marco final, onde o cumprimento destes objetivos é verificado. Ao final das quatro fases, é produzida uma versão do produto, que pode ser evoluída, passando-se novamente pelas quatro fases. A Figura 2-1 ilustra este ciclo.

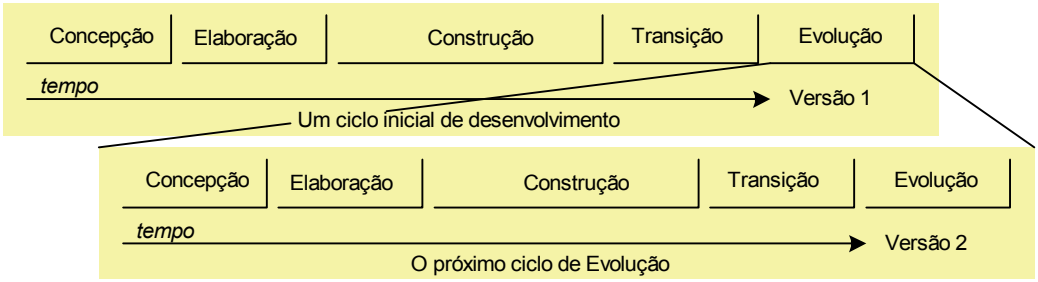


Figura 2-1 – Ciclo de vida do RUP

A fase de Concepção é a primeira do ciclo e tem como objetivos principais definir o escopo do projeto, identificar os casos de uso críticos para o sistema e propor uma arquitetura que os atenda. No que diz respeito ao gerenciamento do projeto, são elaboradas as estimativas iniciais de custo e de cronograma para o projeto inteiro e feito o levantamento inicial de riscos, bem como estimativas detalhadas para a próxima fase e preparação do ambiente de suporte, como instalação e configuração de ferramentas. Ao final desta fase, deve-se decidir se é viável ou não levar o projeto adiante.

A fase seguinte, denominada Elaboração, tem como objetivo estabilizar a arquitetura do sistema, para prover uma base estável para o trabalho de projeto e implementação da próxima fase: a fase de Construção. Por isso, são atacados nessa fase todos os riscos relacionados com a arquitetura do sistema, escolhendo-se os casos de uso ou cenários mais críticos. É importante que todas as funcionalidades do sistema sejam exploradas, pelo menos em seu cenário mais comum, para garantir que todos os riscos técnicos sejam levantados o quanto antes. Esta abordagem é denominada "larga e superficial".

Além disto, deve-se garantir que os requisitos e planos estão estáveis o bastante para que se possa fazer uma estimativa segura do custo e do prazo final de conclusão do projeto. Ao final da fase, também são produzidos um protótipo evolucionário do sistema e planos detalhados para as iterações da fase de Construção.

Na fase de Construção, a ênfase é em completar o produto iniciado na fase de Elaboração. Nesta fase, devem ser completados os cenários restantes de todos os casos de uso, produzindo versões de teste do software para os usuários. Nesta fase, deve existir uma preocupação com a gestão dos recursos e com a qualidade do projeto, com o objetivo de evitar possíveis descartes ou re-trabalhos desnecessários. Ao final desta fase, deve-se ter uma versão *beta* do produto que possa ser testada no ambiente do usuário final.

A última fase, a Transição, tem como objetivo disponibilizar a versão final do sistema no ambiente do usuário final. Isto inclui disponibilização e instalação do produto, migração de dados, caso seja necessário, treinamento dos usuários, pequenos ajustes de

desempenho e correções finais de erros. No caso de um produto que será comercializado, também podem ser incluídas atividades de divulgação, empacotamento e distribuição do produto. Ao final, deve-se ter concordância entre os envolvidos de que o projeto pode ser considerado concluído.

Cada fase é dividida em iterações, que são pequenos desenvolvimentos em "cascata", onde apenas uma parte da funcionalidade do sistema é atacada, passando-se por todas as disciplinas do processo. A ênfase em cada disciplina será ditada pela fase em que o projeto se encontra. A Figura 2-2 mostra o relacionamento entre as fases e disciplinas do RUP. Na fase de Concepção, por exemplo, há uma grande ênfase nas disciplinas Modelagem do Negócio e Requisitos, enquanto na Construção, a ênfase nas atividades de Implementação é maior.

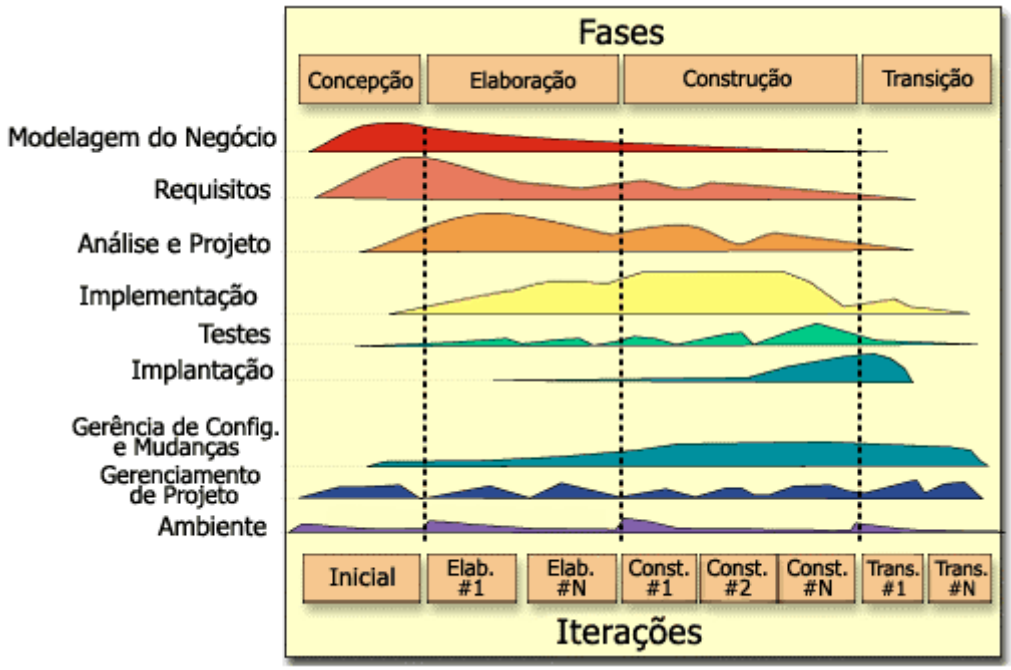


Figura 2-2 – Relacionamento entre fases e disciplinas do RUP

2.3 Disciplinas

Como já dito anteriormente, o propósito das disciplinas é agrupar atividades relacionadas, com o intuito de facilitar o entendimento do processo. A seguir, são apresentadas as disciplinas do RUP, com seus objetivos e suas principais atividades. Primeiramente, são apresentadas as disciplinas de desenvolvimento e, em seguida, as disciplinas de suporte.

2.3.1 Modelagem do Negócio

O objetivo da disciplina Modelagem do Negócio é entender a organização onde o sistema será implantado, desde sua estrutura e funcionamento até os problemas existentes. As atividades desta disciplina são importantes quando o sistema desenvolvido irá reformular os processos já existentes na organização ou quando a equipe de desenvolvimento não tem conhecimento do domínio do negócio.

Esta disciplina pode ser vista como opcional pois, de acordo com o RUP, todos os seus artefatos podem ser excluídos numa eventual adaptação. Inclusive, da versão 2000 para a versão 2002, a Modelagem do Negócio, que era colocada na lista de disciplinas antes da disciplina de Requisitos, foi transferida para o final das disciplinas, reforçando ainda mais seu caráter específico para determinadas situações.

2.3.2 Requisitos

Na disciplina Requisitos, o objetivo principal é definir, em acordo com os *stakeholders*[2] o que o sistema deve fazer. Isto inclui definir os limites do sistema, especificar os requisitos funcionais e não funcionais e definir a interface com o usuário. Para isto, são identificados os atores e casos de uso do sistema, e elaborados protótipos para validar a interface do sistema e as funcionalidades mais críticas.

Além disto, é elaborado um Plano de Gerenciamento de Requisitos, que define como as mudanças nos requisitos do sistema serão controladas, reportadas e medidas. Os requisitos identificados nesta disciplina, a cada iteração, devem prover uma base para as estimativas e planejamento do conteúdo das iterações seguintes.

2.3.3 Análise e Projeto

O objetivo da disciplina Análise e Projeto é transformar os requisitos definidos em uma especificação de como o sistema será implementado. Nesta disciplina, a arquitetura do sistema é considerada bastante importante.

Uma das preocupações iniciais do projeto é estabelecer uma arquitetura estável e robusta, que atenda aos requisitos e ataque o maior número de riscos técnicos possíveis. Também são levados em conta requisitos não-funcionais, como distribuição do sistema, persistência e escalabilidade. Por isso, no início do projeto, devem ser selecionados os casos de uso mais críticos para o sistema, ou seja os de maior risco e/ou de maior importância para o negócio.

---

[2] *Stakeholders* são todas as pessoas que são afetadas pelo sistema de alguma forma. Inclui desenvolvedores, usuários finais e o patrocinador do projeto, entre outros.

2.3.4 Implementação

As atividades da disciplina Implementação lidam com a codificação da especificação do sistema, definida na disciplina Análise e Projeto. Isto inclui desde a organização do código, para refletir a estrutura de subsistemas definida durante o projeto, até a codificação dos componentes.

Uma vez codificados, os componentes são testados unitariamente e, após a correção dos defeitos encontrados, integrados à versão existente do sistema. Esta integração ocorre em duas etapas: primeiramente o componente é integrado a um subsistema, que é testado. Em seguida, os subsistemas são integrados para formar o produto, que passa novamente por uma série de testes.

2.3.5 Testes

O objetivo das atividades da disciplina Testes é avaliar a qualidade do produto, identificando defeitos existentes e garantindo que eles serão corrigidos antes da entrega final do produto. Para isso, são verificadas a interação entre os componentes do sistema e sua integração apropriada, e a correta implementação dos requisitos.

As atividades de testes do RUP tratam somente dos testes de integração e de sistemas, já que os testes unitários são parte da disciplina Implementação. Envolvem o planejamento das atividades de testes, a definição de que tipos de testes serão executados, a elaboração dos casos de testes e, se necessário, a automatização dos procedimentos de testes, execução dos testes e avaliação dos resultados.

2.3.6 Implantação

A disciplina de Implantação agrupa as atividades relacionadas com a disponibilização do produto para seus usuários finais. Isto inclui instalação, testes no ambiente do usuário, treinamento e migração de dados, caso o sistema vá substituir um já existente.

As atividades de Implantação do RUP contemplam produtos desenvolvidos por encomenda, que geralmente serão instalados no cliente pela equipe desenvolvedora, produtos distribuídos pela Internet ou softwares de prateleira, que são produzidos, empacotados e distribuídos em larga escala.

2.3.7 Gerenciamento de Projeto

Entregar um produto que atende às necessidades do usuário, dentro do prazo e do custo planejados é o principal objetivo do Gerenciamento de Projeto. Para isso, é

necessário gerenciar os riscos, superar as restrições existentes e balancear objetivos conflitantes. No RUP, as atividades de Gerenciamento de Projeto têm início no estudo da viabilidade do produto, antes do projeto propriamente dito ser iniciado, e abrange todo o ciclo de vida do produto, até sua entrega final ao cliente.

Para a gestão do projeto, são executadas atividades de planejamento, como estimativas e elaboração de planos e cronogramas, acompanhamento do progresso do projeto, através de avaliações da iteração e coleta de métricas, gerenciamento de riscos, obtenção de recursos (humanos ou não) para o projeto, entre outros. Algumas destas atividades são executadas somente no início do projeto, enquanto outras são repetidas a cada iteração.

No RUP, o planejamento do projeto é feito em dois níveis: um planejamento-macro do projeto, feito durante a fase de concepção, e planos detalhados, feitos ao final de cada iteração, cujo escopo será a iteração seguinte. O projeto é acompanhado principalmente através de revisões ao final das iterações e ao final das fases, onde deve ser julgado se o projeto deve ou não ser levado adiante.

2.3.8 Gerência de Configuração e Mudanças

O propósito das atividades desta disciplina é garantir que os artefatos produzidos durante o projeto estejam sempre íntegros, ou seja, que as versões corretas estão sendo usadas pela equipe e que não há problemas na atualização destas versões devido a atualizações simultâneas, por exemplo.

Além disto, as mudanças nos artefatos devem ser feitas de forma controlada, sendo requisitadas e analisadas antes de serem efetivamente implementadas, e informadas a todos quando concluídas. Para isto, são identificados os artefatos que serão controlados, definidos procedimentos para a atualização destes artefatos, e realizadas auditorias para garantir que os produtos controlados estão íntegros e que a equipe está seguindo os procedimentos definidos.

2.3.9 Ambiente

O objetivo da disciplina Ambiente é descrever as atividades necessárias para a adaptação do RUP no âmbito da organização ou de um projeto. Isto inclui a definição de quais atividades serão ou não executadas, de guias para execução das atividades, como por exemplo guias de modelagem específicos da organização, e de ferramentas e modelos de documentos.

Para a configuração do RUP, são levados em conta aspectos como tipo de desenvolvimento, por exemplo, desenvolvimento interno à organização ou comercial, tamanho do projeto e da equipe, experiência da equipe no processo e no tipo de aplicação a ser desenvolvido, e fatores organizacionais. A ênfase nas atividades da disciplina Ambiente é maior, em geral, no início das fases, e em especial no início do projeto, pois é nesses momentos em que eventuais adaptações no processo ou nas ferramentas de apoio são necessárias.

2.4 Instâncias do RUP para Pequenas Equipes

Por se tratar de um processo que busca abranger uma grande diversidade de situações no desenvolvimento de software, o RUP tem sido criticado por ser bastante abrangente e extenso, o que o torna abstrato e genérico demais para ser aplicado na prática [19], principalmente em projetos de menor porte. Em virtude disto, nos últimos tempos, a Rational vem publicando diversos artigos que mostram como ele pode ser utilizado para projetos de menor escala.

Em [20], Evans mostra uma maneira simples de começar a implantar o RUP em uma empresa que usa o tradicional processo em cascata. A abordagem sugerida consiste em adotar inicialmente os principais conceitos do RUP e, em seguida, ir adicionado as atividades que a equipe julgar necessária. Evans propõe que as iterações durem de 3 a 4 semanas, para que a equipe tenha oportunidade de avaliar constantemente seu trabalho e fazer as correções necessárias no processo.

Além disto, propõe dois guias passo-a-passo das atividades que devem ser feitas na primeira iteração do projeto, durante a fase de Concepção, e nas iterações seguintes. O principal objetivo destes guias é entender o problema a ser resolvido e atacá-lo aos poucos, através do ciclo de vida iterativo e incremental, sempre preocupando-se em mitigar os riscos o mais rápido possível.

Augustine mostra em [21] como o RUP pode ser utilizado para projetos de pequeno porte, definidos como aqueles cujo nível de formalidade dos artefatos é baixo. São definidos quais são os artefatos mínimos necessários para este tipo de projeto: Visão[3], Lista de Riscos, Caso de Desenvolvimento[4], os conjuntos de Casos de Uso e de Testes, a definição da arquitetura do sistema, o Plano de Projeto e o Glossário. O grau de formalidade destes artefatos fica a critério da equipe. O Plano de Projeto, por exemplo,

[3] Descreve, do ponto de vista do usuário, as principais funcionalidades e características do produto a ser desenvolvido.
[4] Descreve o processo de desenvolvimento utilizado no projeto.

pode estar em uma cartolina na parede ou numa página da intranet da empresa. As iterações devem ser breves e sua avaliação bastante objetiva, para que a equipe saiba como está o andamento do projeto e possa corrigir os pontos problemáticos da iteração que se encerrou, sem muita burocracia.

Por fim, em sua versão 2002, o RUP fornece um guia de instanciação para pequenos projetos, provavelmente fruto do amadurecimento das idéias do artigo citado anteriormente. Neste guia, projetos pequenos são definidos como aqueles em que a equipe possui de três a dez membros e cujo tempo de desenvolvimento é de até um ano.

São apresentados, como exemplo, um Caso de Desenvolvimento e um Plano de Projeto para um projeto com estas características. No Caso de Desenvolvimento, há uma lista dos artefatos que devem ser produzidos, ressaltando-se que a principal característica deste tipo projeto é o baixo grau de formalidade dos artefatos. Assim, o esforço de produção e atualização de alguns destes artefatos deve ser pequeno.

Um problema deste guia é que ele é focado nos artefatos e não leva em consideração simplificações nas atividades, ou seja, quando o usuário do RUP consultar a atividade que gera um determinado artefato, irá se deparar com as atividades e subfluxos do RUP original, sem nenhuma adaptação para os projetos pequenos. Este problema acontece, por exemplo, com a atividade Avaliar Iteração, que possui uma série de artefatos de entrada que, de acordo com o guia, não precisam ser produzidos. Isto pode trazer dificuldades no uso deste guia.

Além disto, algumas atividades certamente poderiam ter seus passos simplificados, já que alguns artefatos não serão produzidos, como é o caso da atividade Monitorar Status do Projeto. Nela, uma vez que o Plano de Medições não é produzido, os passos Derivar Indicadores de Qualidade e Derivar Indicadores de Progresso não fazem sentido.

2.5 Considerações Finais

Neste capítulo foi apresentado o Rational Unified Process, um *framework* de processos que serve a diversos tipos de projeto. Ele captura um conjunto de melhores práticas do desenvolvimento de software, como o desenvolvimento iterativo e incremental, e fornece guias detalhados sobre a execução das atividades de engenharia de software. Além disto, abrange todo o ciclo de vida do desenvolvimento de software, desde sua concepção até a implantação no ambiente de produção, é suportado por ferramentas, incluindo guias de uso das mesmas, e é amplamente difundido na indústria de desenvolvimento de software.

Todavia, por ser bastante genérico, o RUP deve ser adaptado à realidade da organização e/ou do projeto onde será utilizado. Esta adaptação, em geral, é bastante custosa, pois o processo é muito extenso, o que requer um grande tempo de análise antes de se decidir que atividades serão ou não executadas. Além disto, a disciplina de Ambiente não provê comentários detalhados o suficiente para a efetiva adaptação do processo. Esta deficiência, inclusive, já foi tratada em um trabalho de dissertação de mestrado [22].

O RUP fornece uma boa base para a definição de metodologias mais específicas. Estas metodologias devem conter os componentes do RUP necessários a cada tipo de desenvolvimento (ex: desenvolvimento para web, sistemas embarcados, projetos geograficamente distribuídos, equipes pequenas, etc.), com as alterações que sejam necessárias. As instanciações devem ser detalhadas o suficiente para que possam ser usadas sem a necessidade de recorrer constantemente ao processo completo.

Capítulo 3: Metodologias Ágeis

3.1 Introdução

O crescente interesse em definir metodologias de desenvolvimento que pudessem ser utilizadas nos diversos tipos de projeto, e cujas atividades fossem detalhadas o suficiente para que pudessem sempre ser executadas de forma a se obter o mesmo sucesso, independente de quem as tivesse executando, fez com muitas metodologias se tornassem bastante complexas.

Nestas metodologias, a quantidade de diferentes papéis e artefatos é enorme, e existe uma excessiva preocupação com a produção de documentação, tornando o processo de desenvolvimento excessivamente formal, com uma série de atividades que agregam pouco valor ao projeto, e deixando em segundo plano as atividades que realmente tratam da produção do software. Além disto, a enorme competitividade da economia em que vivemos torna obrigatório que as empresas produzam mais e melhor em menos tempo, o que muitas vezes torna inviável a produção de uma especificação detalhada antes da implementação do sistema ou um planejamento a longo prazo, uma vez que as necessidades de negócio mudam constantemente.

Em virtude destas deficiências das metodologias de desenvolvimento pesadas, um grupo de dezessete consultores e especialistas em desenvolvimento de software fundou a Aliança Ágil [23], uma associação que tem por objetivo encontrar abordagens mais simples e eficientes para o desenvolvimento de software. Eles publicaram um manifesto, chamado "Manifesto Para o Desenvolvimento Ágil de Software" que sintetiza a preocupação dos membros da associação, composto por quatro itens:

"Estamos descobrindo melhores maneiras de se desenvolver software, fazendo isto e ajudando os outros a fazer isto. Através deste trabalho, nós passamos a valorizar:

- **Indivíduos e interações** mais que processos e ferramentas;
- **Software funcionando** mais que documentação abrangente;

- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder à mudança** mais que seguir um plano.

Ou seja, apesar de existir valores nos itens à direita, valorizamos mais os itens à esquerda".

As metodologias definidas por este grupo passaram a serem chamadas Metodologias Ágeis. Cada membro do grupo possui uma abordagem diferente para o desenvolvimento de software, ou seja, sua própria metodologia. Entretanto, elas possuem vários pontos em comum, como o ciclo de vida iterativo e incremental, com iterações de duração curta (até um mês), são voltadas para equipes pequenas, dão uma grande ênfase à comunicação constante e mais informal do que à documentação escrita e ao planejamento em curto prazo, porém feito constantemente, entre outros.

Neste capítulo apresentamos as cinco metodologias ágeis que mais têm se destacado: Extreme Programming, Scrum, Feature Driven Development, Dynamic System Development Method e Crystal Clear. São apresentadas suas principais características, papéis, atividades e limitações. Uma comparação mais detalhada das diversas metodologias ágeis pode ser encontrada em [24].

3.2 Extreme Programming

Extreme Programming (XP) é considerada a mais "leve" das metodologias ágeis. Ela é o resultado da experiência de Kent Beck durante o projeto C3 Payroll, na empresa Chrysler [8], e tem como objetivo tornar o desenvolvimento de software uma atividade simples e que mantenha a equipe constantemente motivada.

Ela é a metodologia ágil mais popular atualmente, sendo amplamente discutida e adotada em diversos projetos no mundo inteiro, como pode ser visto em [25], [26], [27], [28], [29] e [30]. A lista de discussão oficial [31], por exemplo, conta com mais de 3000 membros, entre eles, grandes "gurus" da orientação a objeto, como Ron Jeffries, Martin Fowler, Robert Martin, Kent Beck e Grady Booch.

XP é definida como um conjunto de 4 valores e 12 práticas que irão guiar as atividades da equipe envolvida no projeto. Ela é considerada "extrema" porque cada prática é adotada de maneira radical, sendo realizada ao extremo (como testes, por exemplo), e porque a metodologia resume-se a esse conjunto de práticas. Caso uma equipe não adote completamente as práticas ou inclua novas práticas, não estará, segundo a metodologia, utilizando XP [32].

Segundo Kent Beck, criador da metodologia, eXtreme Programming não é a solução para todos os problemas enfrentados hoje no desenvolvimento de software, mas para um subconjunto específico deles. XP foi concebida para projetos cujos requisitos são voláteis, ou seja, estão em constante modificação, e cuja equipe de desenvolvimento seja pequena, possuindo no máximo 12 pessoas.

Um projeto XP é desenvolvido de forma iterativa e incremental. As iterações têm uma duração curtíssima, entre 1 e 4 semanas, e são sempre de tamanho fixo (*time-boxed*). Além disso, a metodologia é voltada para projetos orientados a objetos, embora grande parte de suas práticas também possam ser usadas em outros tipos de projetos.

3.2.1 Valores

Comunicação: para que o projeto tenha sucesso é necessário que haja comunicação constante entre seus participantes. A comunicação constante entre o cliente e a equipe traz a certeza de que os programadores sempre estão trabalhando no que há de mais importante para o cliente e de que o que já foi feito está de acordo com o que foi pedido. A comunicação constante entre os membros da equipe faz com que todos tenham consciência do andamento do projeto e aumenta a integração e a motivação do time.

Feedback: em XP o *feedback* constante permite saber se o projeto está caminhando corretamente e permite identificar cedo desvios que estejam ocorrendo. Esse *feedback* é resultado dos pequenos *releases*, onde o cliente verifica constantemente o resultado do trabalho; dos testes constantes, que permitem saber se o que foi feito está realmente funcionando; e da interação direta e constante com o cliente, o qual é considerado parte da equipe.

Coragem: a coragem garante que a equipe nunca deixará de fazer algo necessário ou que possa trazer grandes benefícios para o projeto, por medo de tentar. A coragem, por exemplo, fará com que a equipe esteja sempre melhorando o código, mesmo que isso signifique ter que reestruturar uma grande parte do sistema ou alterar o código escrito por outra pessoa e que, por isso, o programador não domina bastante.

Simplicidade: a simplicidade baseia-se na idéia de que é melhor fazer algo simples hoje, e pagar um pouco mais amanhã para modificá-lo, do que investir muito em algo extremamente geral, que pode ser que nunca seja usado completamente. A simplicidade facilita a comunicação da equipe e a implantação de mudanças no sistema.

3.2.2 Práticas Chave

O modo de se desenvolver um projeto XP é resumido em 12 práticas. Essas práticas se relacionam fortemente e eliminam as fraquezas umas das outras em uma grande sinergia. Pode-se adotar parte das práticas, mas os benefícios reais só aparecem quando todas estão sendo usadas.

Pequenos *releases*: em XP, a melhor maneira de se saber se o projeto está sendo desenvolvido corretamente é obtendo-se um *feedback* do cliente. Para isso, são feitos *releases* constantes do sistema, normalmente a cada 2 meses. Dessa forma, o cliente pode verificar se a equipe entendeu bem o que estava sendo pedido e pode ver o progresso alcançado, ficando claro o quanto o projeto está andando e quais as dificuldades encontradas.

Cliente presente: num projeto XP, o cliente faz parte da equipe de desenvolvimento. O cliente não é necessariamente aquele que está pagando pelo projeto, mas aquele que entende bem o que o sistema deve fazer e qual a importância do sistema para os negócios da empresa. O papel do cliente pode ser desempenhado por várias pessoas, desde que falem como uma só voz. Além disso, o cliente não é fixo, podendo ser representado por diferentes pessoas em diferentes momentos do projeto. Como parte da equipe, o cliente possui um papel ativo, principalmente nas atividades de planejamento. Ele é o responsável por escrever as histórias[5], priorizá-las segundo seu valor para o negócio e escolher em qual *release* elas serão implementadas, entre outras tarefas.

Integração Contínua: em XP, cada correção ou nova implementação realizada é imediatamente integrada ao código já existente. Em geral, existe uma máquina dedicada para a integração do código, para que somente uma dupla integre suas mudanças por vez. Isso permite que eventuais erros de integração sejam isolados mais facilmente, pois cada dupla é responsável por corrigir os erros que aparecerem e fazer com que todos os testes estejam executando perfeitamente. A atividade de integração deve ser realizada constantemente, de preferência várias vezes ao dia.

Testes constantes: um dos grandes diferenciais de XP em relação a outras metodologias é a grande ênfase dada em testes. Tudo o que pode falhar deve ser testado. Os programadores são responsáveis por elaborar testes unitários e o cliente, os testes funcionais de aceitação. Os testes unitários são executados várias vezes ao dia, para garantir que nenhuma alteração contém erros, e o sistema deve sempre passar em todos. Se pelo menos um teste falhar, não se deve adiar a correção do problema. Para que os testes

[5] Pequenas descrições de funcionalidades que o sistema deve prover.

constantes sejam possíveis e não atrasem a equipe, a atividade de testes deve ser automatizada ao máximo.

Jogo do planejamento: o planejamento de um *release* ou de uma iteração é feito com base em histórias, e envolve toda a equipe de desenvolvimento, inclusive o cliente, em dois papéis:

- Comercial: sabe o que é mais importante do ponto de vista do negócio e, por isso, decide que funcionalidades devem ser entregues a cada *release*.
- Técnico: estima o tempo necessário para se implementar uma história e avisa a equipe do papel Comercial sobre os riscos envolvendo cada uma.

Dadas as estimativas do pessoal técnico, a equipe de negócios deve escolher o que acha mais importante para os próximos *releases*. O planejamento é feito para um período curto de tempo, no máximo dois *releases* de antecedência.

No decorrer do trabalho, a equipe vai medindo o quanto pode produzir. Caso não seja possível cumprir tudo o que foi planejado, o cliente deve julgar o que é menos importante e pode ser deixado para o próximo *release*. Se a equipe terminar o trabalho antes do prazo, o cliente pode escolher novas histórias para serem implementadas nesse *release*.

Propriedade coletiva do código: para que não haja um atraso no projeto devido a uma grande dependência entre os membros da equipe, XP define que todos são responsáveis por todo o código do projeto. Assim, qualquer pessoa pode alterar qualquer parte do código quando for necessário. À primeira vista, isso parece levar a grandes confusões, mas isso é possível devido ao Padrão de Codificação, à existência dos testes, que garantem que as mudanças efetuadas funcionam corretamente, e à Integração Contínua, que garante que as alterações não conflitarão com o que já existe. De preferência, essa atividade deve ser suportada por uma ferramenta de controle de versão.

Refatoramento constante: o código deve ser mantido o mais claro possível, comunicando o que ele realmente faz, e facilitando sua alteração. Para isso, todo programador deve sempre melhorar o código do sistema, evitando a duplicação e estruturando bem suas classes. Isso deve ser feito sempre que o programador sentir que o código está ficando complexo e confuso.

Padrão de codificação: para que a equipe possa desenvolver a Propriedade Coletiva do Código e fazer o refatoramento (*refactoring)* do código com segurança é necessário que todos compartilhem um padrão de codificação, para facilitar a comunicação

entre a equipe. Esse padrão deve fazer com que o código pareça familiar a quem o lê, não devendo ser algo complexo ou rebuscado.

Programação em pares: todo o código é produzido por duas pessoas trabalhando em uma mesma máquina. Cada uma desempenha um papel: quem está com o teclado deve se preocupar com a melhor maneira de se implementar o código atual. O seu parceiro preocupar-se com questões mais gerais, como casos de teste que possam falhar, se o código realmente está claro, se esta é a melhor solução, sugerir possíveis refatoramentos e observar erros do parceiro que está digitando. Os papéis devem ser trocados constantemente e as duplas também. Dessa forma, todos têm conhecimento, ainda que superficial, de todas as partes do sistema, a equipe fica mais integrada e os membros dela aprendem uns com os outros.

Projeto simples: em XP, o projeto do sistema deve ser o mais simples possível. Não se deve perder muito tempo elaborando algo mais complexo do que o que *pode* ser usado amanhã. XP parte do princípio de que se deve projetar para hoje. Só deve-se fazer uma mudança quando ela realmente for necessária. A idéia é que, como os requisitos estão em constante mudança, não vale a pena investir na generalização de algo que pode não ter mais importância amanhã. Além disso, como o projeto do sistema é sempre mantido o mais simples possível, as mudanças não devem ser difíceis de serem efetuadas. Um projeto simples, segundo Beck [8] e Jeffries [33] é aquele que:

- Todos os testes executam com sucesso;
- Não possui lógica duplicada;
- Expressa a idéia que o programador teve;
- Contém o menor número possível de classes e métodos.

Metáfora: a metáfora é uma analogia entre o software em desenvolvimento e um outro sistema (não necessariamente um software) conhecido pela equipe. Segundo Kent Beck, ela serve para facilitar o entendimento do projeto e criar o "vocabulário comum", que servirá como base para a criação de nomes de classes, subsistemas, etc. Por exemplo, um sistema de recuperação de informação baseado em agentes pode ser descrito como um enxame de abelhas, que saem procurando pólen e o trazem de volta para colméia. Martin Fowler critica esse conceito [25], argumentando que essa idéia só funcionou no projeto C3 Payroll e que a melhor maneira de se criar um vocabulário comum é fazendo um modelo conceitual do domínio usando uma notação como a UML.

Semana de 40 horas: em XP, não se deve trabalhar além do horário normal por 2 semanas seguidas. Como o ritmo de trabalho é "extremo", todos devem estar

descansados e motivados para que o trabalho seja bem desenvolvido. Além disso, ao se trabalhar cansado, o risco de se inserir *bugs* no programa é muito maior. Em alguns casos, é necessário fazer horas extras para se cumprir um prazo, mas isso deve ser evitado ao máximo.

3.2.3 Fases

Beck divide um projeto XP em várias fases [8], entretanto, outras referências não levam estas fases em consideração [33] [35]. A seguir, apresentamos as fases propostas e seus respectivos objetivos.

Exploração: nesta fase são feitas experiências para se iniciar o projeto. A equipe de desenvolvimento deve testar ferramentas e tecnologias que irá utilizar, para identificar possíveis problemas e fazer as estimativas com mais precisão. O cliente deve habituar-se a escrever as histórias e identificar um conjunto delas suficiente para se compor um primeiro *release*.

Planejamento inicial: nesse planejamento deve-se entrar em acordo sobre a data do primeiro *release* e quais histórias farão parte dele. Esse planejamento é feito através do Jogo do Planejamento e deve ser feito num horizonte de 2 a 6 meses.

Iterações para o primeiro *release*: o planejamento inicial é dividido em iterações que possuam entre 1 e 4 semanas de duração. Uma parte das histórias escolhidas para o *release* irá ser implementada a cada iteração. Na primeira iteração, deve-se dar ênfase às histórias que afetem mais a estrutura do sistema. Ao final da iteração deve-se obter uma versão executável que implemente as histórias, e o cliente irá executar os testes funcionais de aceitação escritos por ele.

Produção: após o primeiro *release*, quando já se tem uma idéia mais clara do sistema a ser desenvolvido e das tecnologias envolvidas, entra-se na fase de produção. Nesta fase o sistema será realmente construído. As iterações podem ser mais curtas que as do primeiro *release* e a equipe deve fazer um encontro periódico, de preferência diário, para que todos saibam dos progressos feitos e dos problemas encontrados. Antes de cada iteração a equipe faz um planejamento, tomando como base o desempenho da iteração passada.

Manutenção: a fase de manutenção é apontada por Beck como "o estado normal de um projeto XP". Nela, a equipe está adicionando novas funcionalidades, mantendo o sistema existente em execução e, possivelmente, alterando a equipe. Essa fase é apresentada como uma etapa avançada da fase de produção.

Morte: o final do projeto acontece quando o cliente não possui mais novas histórias a serem implementadas. Neste caso, o desenvolvimento é encerrado e a equipe prepara uma documentação concisa para os usuários. Caso o projeto seja acabado por inviabilidade econômica, toda a equipe deve ser reunida para ser informada do porquê dessa decisão e para tirar lições para os próximos projetos.

3.3 Scrum

Scrum é uma metodologia ágil que parte do princípio que o desenvolvimento de software é um processo empírico, ou seja, não obedece a leis pré-definidas e as transformações das "entradas" do processo em "saídas" são complexas e variáveis. Por isso, para se obter sucesso em um projeto, é necessário um monitoramento constante das variáveis envolvidas, como requisitos, recursos e tecnologia, e a capacidade de reagir rapidamente a mudanças [34]. Foi proposta por Ken Schwaber e Jeff Shutterland, inicialmente na forma de um *pattern*. [9].

Devido à imprevisibilidade dos projetos de software, Scrum não define técnicas ou práticas específicas de desenvolvimento. Ela concentra-se em atividades de gerenciamento do projeto, como planejamento e acompanhamento, que devem ser executadas para a produção de um sistema flexível e que esteja sendo desenvolvido em um ambiente onde os requisitos estão constantemente evoluindo. Scrum é voltada para equipes de seis a dez pessoas. Caso haja um time maior, o mesmo deve ser sub-dividido em vários times menores, assim como o projeto em subprojetos.

Seu ciclo de vida, apresentado na Figura 3-1, é iterativo e incremental, e divido em três fases: Pré-jogo, Desenvolvimento e Pós-jogo. Nas fases inicial e final, as atividades são bem determinadas, enquanto a fase de Desenvolvimento é vista como uma caixa-preta, onde as técnicas e ferramentas utilizadas para a construção do sistema ficam a cargo da equipe.

No Pré-jogo é feito um planejamento inicial e um projeto de arquitetura do sistema em alto nível. Esta fase pode possuir várias iterações.O planejamento inicial irá, entre outras coisas, identificar riscos e como controlá-los, definir os times que trabalharam no *release* e desenvolver um *backlog* inicial para o sistema, ou seja, uma lista de todos os requisitos do sistema.

A fase de Desenvolvimento é onde o produto é efetivamente construído. Esta construção é feita em uma série de iterações, denominadas *sprints*, que têm duração de um

mês, e termina com uma versão executável do sistema. Durante um *sprint*, passa-se por todas as etapas tradicionais do desenvolvimento, desde os requisitos até os testes.

O Pós-jogo marca o final da elaboração do produto. Nesta fase, são feitos a integração final e os testes de sistema. Atividades de treinamento, elaboração de documentação e implantação do sistema também são feitas nesta fase. A seguir, detalhamos cada fase da metodologia.

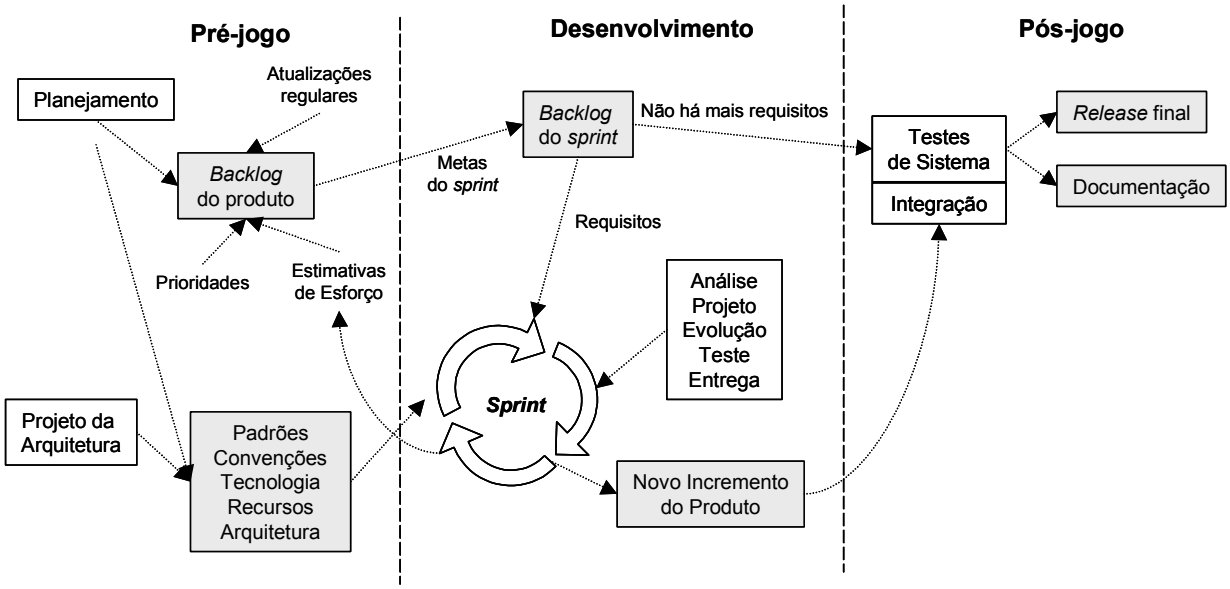


Figura 3-1 - Fases do Scrum

3.3.1 Pré-jogo

Além das atividades já citadas de planejamento inicial e projeto da arquitetura, o Pré-jogo possui as seguintes atividades:

- Definição da data de entrega e da funcionalidade de um ou mais *releases*;
- Escolha do *release* mais apropriado para o desenvolvimento imediato;
- Mapeamento entre pacotes de objetos e itens do *backlog* para o próximo *release*.
- Revisão e possível ajuste desses itens do *backlog* e dos respectivos pacotes;
- Validação ou seleção de ferramentas de desenvolvimento e infra-estrutura;
- Estimativa do custo do *release*, incluindo desenvolvimento, marketing, treinamento e entrega;
- Verificação da aprovação e do orçamento para o projeto.

No planejamento, o cliente[6] escolhe a data do primeiro *release*. Feito isso, o cliente trabalha em conjunto com a equipe de desenvolvimento para que o *release* tenha os requisitos de maior importância. O cliente é responsável por priorizar os requisitos e a equipe de desenvolvimento por estimar o esforço necessário para implementar esses requisitos. Caso as estimativas ultrapassem a data estipulada, o grupo deve entrar em acordo para escolher um subconjunto desses requisitos que possa ser entregue no prazo. A data do *release* não deve, a princípio, ser alterada.

O projeto da arquitetura irá determinar como os itens do *backlog* serão implementados, através de modificações na arquitetura já existente e de um projeto em alto nível. No início do projeto deve ser definido um Arquiteto Chefe. Ele será responsável por elaborar uma visão do projeto de desenvolvimento baseada na arquitetura definida e garantir que esta visão estará consistente durante todas as fases do projeto. Outras atividades do projeto da arquitetura incluem:

- Revisão dos itens do *backlog* já atribuídos;
- Identificação das mudanças necessárias para implementar os itens do *backlog*;
- Refinar a arquitetura do sistema para dar suporte aos novos requisitos;
- Identificar quaisquer problemas ou questões na implementação de mudanças;
- Um encontro para o projeto do sistema, onde cada time irá apresentar sua abordagem e as mudanças necessárias para implementar os itens do *backlog* que são de sua responsabilidade.

3.3.2 Desenvolvimento

Como já foi dito, o desenvolvimento é estruturado na forma de iterações que duram por volta de 30 dias, denominadas *sprints*. A estrutura de um *sprint*, com suas entradas e práticas, é apresentada na Figura 3-2.

[6] Por cliente entende-se a equipe que conhece o domínio da aplicação e a importância do software em desenvolvimento. Pode ser quem está financiando o projeto, a equipe de marketing, etc...

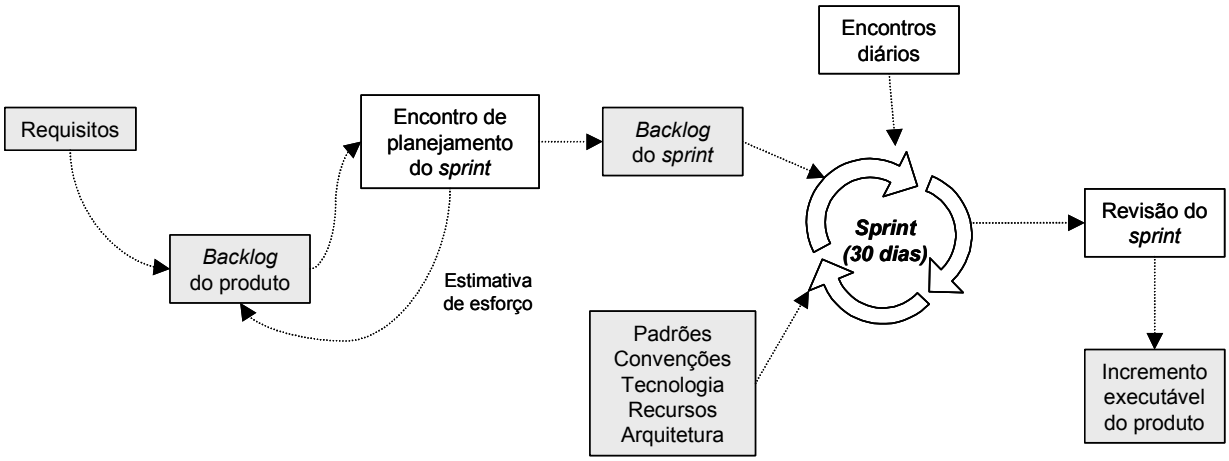


Figura 3-2 - Entradas e práticas de um *sprint*

Um *sprint* tem início com um planejamento, onde será definido que subconjunto dos itens do *backlog* do sistema será implementado. Este subconjunto irá formar o *backlog* do *sprint*, que não pode ser alterado até que o *sprint* seja concluído.

O progresso do projeto e os problemas encontrados pela equipe são monitorados através de encontros diários, que contam com a participação de toda a equipe. Estes encontros devem ser rápidos (não mais que 30 minutos) e as seguintes perguntas devem ser respondidas:

- O que foi completado desde o último encontro?
- Que dificuldades impediram o trabalho de ser completado?
- O que se planeja fazer até o próximo encontro?

Os encontros são mediados pela figura do Scrum Master que, além de mediar os encontros, é responsável por identificar os itens do *backlog* que precisam ser completados nesse *sprint* e medir o progresso da equipe.

Após a execução de um *sprint*, é promovida a Revisão do *Sprint*, um encontro envolvendo todos os participantes do projeto, desde a equipe de desenvolvimento até o pessoal de negócios. Nesse encontro é feito um relatório sobre as atividades do *sprint* e os próximos passos do projeto são discutidos. Além disto, é apresentado o executável gerado do *sprint*, para que seja validado pelos usuários.

Na Revisão do *Sprint*, qualquer aspecto do projeto pode ser modificado, desde a priorização dos requisitos até a inclusão de requisitos completamente novos. Novas estimativas e um novo planejamento são feitos, caso seja necessário.

3.3.3 Pós-jogo

A fase de Pós-jogo acontece ao final do projeto, marcando seu fechamento. Uma característica peculiar da metodologia é que o projeto pode acabar a qualquer momento. O

cliente deve decidir, ao final de cada *sprint*, se o projeto deve continuar ou não. Essa decisão deve ser baseada no tempo de desenvolvimento, requisitos, custo, qualidade e competitividade (no caso de um produto comercial).

O Pós-jogo prepara o produto desenvolvido para um *release* oficial aos usuários do sistema. Para isso, as seguintes tarefas devem ser feitas:

- Integração final;
- Testes de sistema;
- Documentação do usuário;
- Preparação de material de treinamento;
- Preparação de material de marketing.

3.4 Feature Driven Development

Feature Driven Development (FDD) é uma metodologia proposta por Peter Coad em seu livro *Java Modeling in Color with UML* [36]. O desenvolvimento é feito de forma iterativa e incremental, com iterações de curta duração, e baseado em modelos.

A equipe de desenvolvimento conta com a participação de especialistas no negócio (representantes do usuário, por exemplo) e é dividida em sub-equipes, chefiadas por um programador-chefe. A metodologia foi inicialmente concebida para grupos de 16 a 20 desenvolvedores, mas pode ser utilizada em grupos maiores, sendo limitada somente pelo número de programadores-chefe [36].

FDD é dividida em 5 atividades principais, baseadas em *features*, cujos objetivos são explicados a seguir. Uma *feature* é uma pequena funcionalidade do sistema que possui valor para o usuário. Assim, "estabelecer a infra-estrutura de persistência" não seria uma *feature*, pois não apresenta resultado visível para o usuário, enquanto "calcular o total do desconto" seria. As atividades de FDD são ilustradas na Figura 3-3.

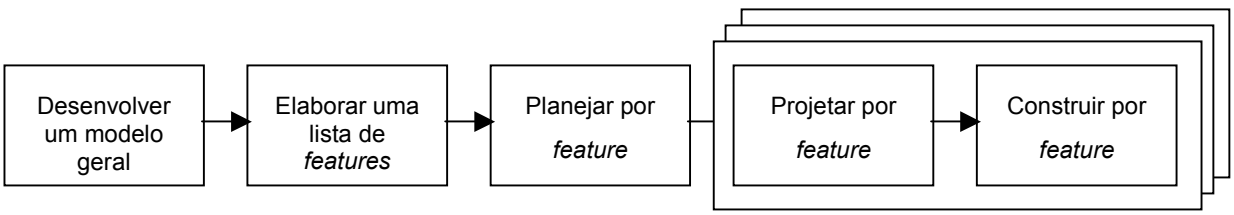


Figura 3-3 - Fases do FDD

Desenvolver um Modelo Geral

É atividade inicial e envolve a equipe de desenvolvimento e especialistas do domínio da aplicação. Os especialistas apresentam uma visão geral do escopo do sistema e do contexto onde este está inserido. Em seguida, eles, junto com a equipe de

desenvolvimento, produzem um modelo inicial do sistema. Os especialistas apresentam então uma visão mais detalhada e todos refinam o modelo produzido. Nestes passos, os especialistas e a equipe trabalham em pequenos grupos, apresentam os resultados do seu grupo e consolidam os resultados em um modelo comum.

Elaborar uma Lista de *Features*

A equipe constrói uma lista de *features*, com o conhecimento adquirido na modelagem inicial. Caso existam, documentos de requisitos ou diagramas de casos de uso podem ser usados como entrada para essa atividade. Quando eles não existem, a equipe pode ir identificando *features* informalmente durante a modelagem inicial. Em seguida, as *features* são agrupadas por funcionalidades. Caso o sistema seja de grande porte, esses conjuntos de *features* podem ser agrupados novamente. A priorização das *features* é feita pelos especialistas do domínio, que também definem qual o menor subconjunto das *features* que formam um sistema de valor para o negócio.

Planejar por *Feature*

Esta atividade consiste em fazer um plano de alto nível para os conjuntos de *features*, determinando sua ordem de construção. Esse plano é delegado a um programador-chefe e classes identificadas no modelo geral são atribuídas a desenvolvedores.

Projetar por *Feature*

Esta atividade, junto com a atividade Construir por feature, é responsável pelo desenvolvimento do sistema. O programador-chefe identifica um conjunto de *features* que possa ser desenvolvido em até 2 semanas, e forma a equipe que irá desenvolver estas *features*, identificando as classes envolvidas nestas *features* e os desenvolvedores responsáveis pelas classes. A equipe produz diagramas de seqüência detalhados, define as assinaturas dos métodos necessários e, ao final, faz uma inspeção no projeto.

Construir por *Feature*

Nesta atividade, os responsáveis pelas classes implementam o código de suas respectivas classes, fazem testes unitários, integram o código e fazem inspeção no código produzido. Quando o programador-chefe autorizar, as *features* já finalizadas são promovidas ao *build* principal do sistema. É importante notar que tanto um desenvolvedor pode fazer parte de, por exemplo, 2 ou 3 equipes de *features* diferentes, como um programador-chefe pode estar gerenciando várias equipes trabalhando em paralelo. Quando um conjunto de *features* é concluído, volta-se à atividade "Projetar por *feature*" para se escolher o próximo conjunto de *features* a ser construído.

3.5 Dynamic System Development Method

3.5.1 Visão Geral

O *Dynamic System Development Method* é um *framework* para processos de desenvolvimento de software baseados na técnica *Rapid Application Development* (RAD) [39], criado e desenvolvido por um consórcio, surgido em 1994, pela necessidade dos fabricantes de ferramentas RAD de criarem um método de desenvolvimento. Isto aconteceu devido à popularização da técnica RAD na comunidade de desenvolvimento de software. Atualmente, o consórcio conta com mais de 100 membros, como Hewllet-Packard, Pricewaterhouse Cooper, British Airways, Rational Software, entre outros [40].

A primeira versão do *framework* de desenvolvimento foi aprova em março de 1994 e, desde então, vem sendo evoluída pelos membros do consórcio. Atualmente, encontra-se na versão 4.1, que foi lançada em outubro de 2001.

Enquanto os métodos tradicionais tentam produzir um software que atenda a requisitos pré-acordados com o usuário, com pequenas variações no tempo e no custo do projeto, DSDM possui uma proposta inversa: o tempo é fixo, os recursos necessários são considerados o mais fixo possível e os requisitos que serão atendidos ao final serão determinados por esses dois fatores e pelas necessidades do negócio. A metodologia parte do princípio fundamental de que nada pode ser construído com perfeição da primeira vez, mas que 80% das principais funcionalidades do sistema podem ser desenvolvidas em 20% do tempo do projeto.

Por ser um *framework,* DSDM aponta **o que** deve ser feito, mas não **como** desempenhar estas atividades. Assim, ele pode ser usado em conjunto com outras metodologias, como RUP [42] e XP [43]. A estratégia de desenvolvimento é baseada em 9 princípios [40]:

1. Um envolvimento ativo do usuário é essencial;
2. As equipes[7] DSDM devem estar capacitadas a tomar decisões;
3. O foco é na entrega freqüente de produtos;
4. Adequação aos objetivos do negócio é o principal critério de aceitação do que for produzido;
5. O desenvolvimento iterativo e incremental é necessário para uma solução de negócio precisa;
6. Todas as mudanças durante o desenvolvimento são reversíveis;

[7] Uma equipe inclui tanto desenvolvedores como usuários

7. Os requisitos são definidos em um alto nível de abstração;
8. Testes são executados durante todo o ciclo de vida;
9. Uma abordagem colaborativa e cooperativa entre todos os *stakeholders* é essencial.

O desenvolvimento é iterativo, feito através de prototipação e conta com a participação do cliente. Segundo [38], isso traz os seguintes benefícios:

- Os usuários se sentem mais "donos" do sistema;
- O risco de se desenvolver o sistema errado é reduzido enormemente;
- O sistema final irá atender melhor às reais necessidades de negócio do cliente;
- Os usuários finais serão mais bem treinados, pois existem representantes deles envolvidos no desenvolvimento que conhecem bem suas necessidades e o sistema.
- A implementação enfrentará menos problemas, pois todas as partes envolvidas no projeto estão cooperando constantemente para o seu sucesso.

DSDM deve ser usado quando se conhece bem quem serão as pessoas afetadas pelo sistema, para que se possa escolher representantes desses grupos para participarem do projeto, representando as necessidades e o conhecimento do domínio que esses usuários possuem. Além disso, se o projeto for de grande porte, deve ser capaz de ser dividido em componentes menores, para que se possa fazer um entrega incremental ou para que o desenvolvimento seja feito de forma paralela.

3.5.2 Ciclo de Vida

Conforme dito anteriormente, o desenvolvimento em DSDM é feito de forma iterativa e incremental. Isso permite que necessidades urgentes do cliente sejam logo satisfeitas e que o cliente valide o que está sendo desenvolvido, ou seja, veja se o que está sendo feito realmente atende às suas necessidades e possui a qualidade desejada.

O ciclo de vida de um projeto DSDM é composto de cinco fases: Estudo da Viabilidade, Estudo do Negócio, Iteração de Modelagem Funcional, Iteração de Projeto e Construção, e Implementação. O *framework* deve ser instanciado em um processo para a organização, de acordo com seu ramo de negócio e suas restrições técnicas. A Figura 3-4 mostra as fases do DSDM.

As duas primeiras fases (Estudo de Viabilidade e do Negócio) devem ser feitas antes das demais e de forma seqüencial, pois irão delinear as características do projeto e os

procedimentos que ele deve seguir. A forma como as outras fases se relacionam e se combinam deve ser decidida pela equipe do projeto. A seguir, apresentamos cada fase em um nível maior de detalhe.

Estudo da Viabilidade

O objetivo dessa fase é determinar se o DSDM pode ser usado no projeto a ser desenvolvido. Esse estudo leva em conta questões técnicas e gerenciais, e deve durar, em geral, algumas semanas.

Estudo do Negócio

Nesta fase o escopo do projeto é estabelecido e os requisitos funcionais e não funcionais do sistema são definidos em um nível de abstração mais alto. Além disso, são elaborados um modelo inicial do negócio e um rascunho da arquitetura do sistema, e definidos os objetivos de manutenibilidade. Assim como o Estudo da Viabilidade, essa fase não deve durar mais do que um mês.

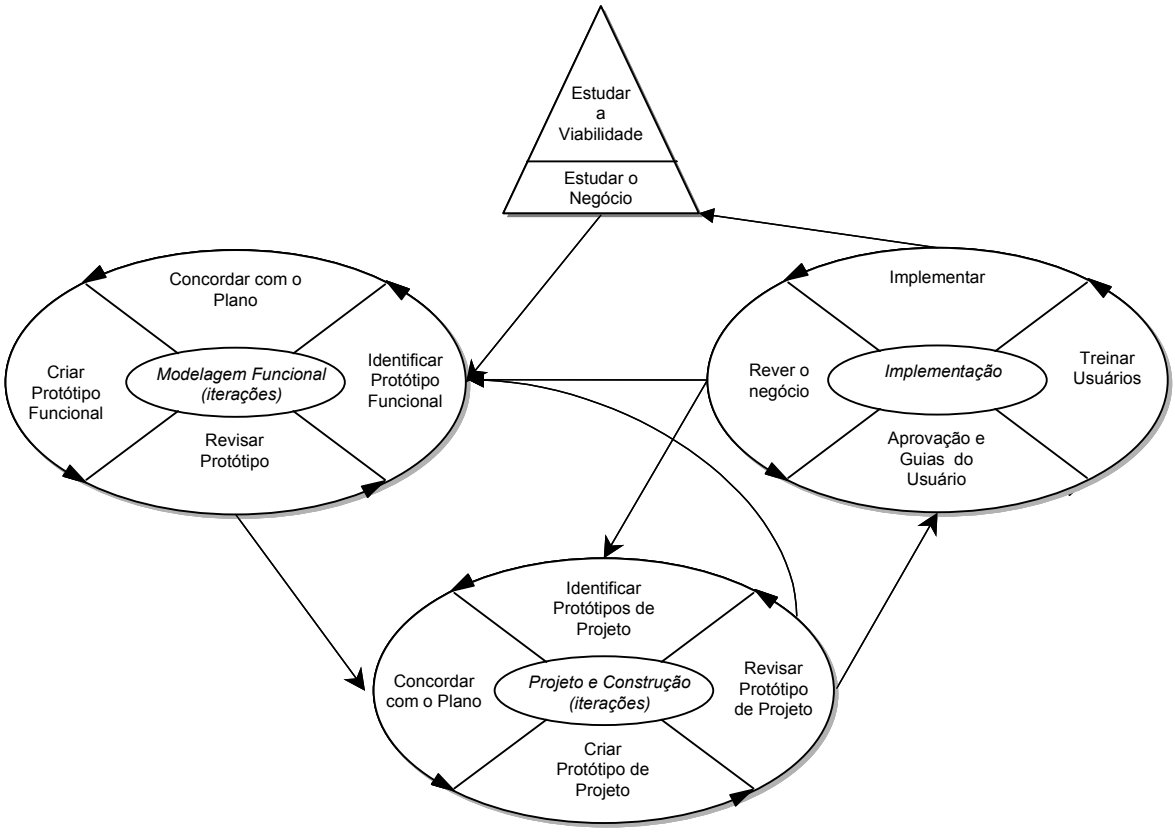


Figura 3-4 - Fases do DSDM

Iteração de Modelagem Funcional

O principal objetivo desta fase é refinar os aspectos de negócio do sistema, com base nos requisitos definidos no Estudo do Negócio. Tanto esta fase com a Iteração de Projeto e Construção consistem basicamente do seguinte ciclo de atividades:

- Identificar o que deve ser produzido;

- Definir como e quando fazê-lo;
- Criar o produto;
- Verificar se ele foi produzido corretamente, através da revisão de documentos, demonstração do protótipo e testes.

A maior parte do esforço do desenvolvimento concentra-se nesta fase e na Iteração de Projeto e Construção, pois é nelas que o protótipo inicial é continuamente incrementado até chegar ao produto final. No DSDM, todos os protótipos devem estar evoluindo rumo ao sistema desejado. Devem, portanto, ser funcionais e robustos o suficiente para serem utilizados pelo cliente e satisfazerem qualquer requisito não funcional considerado crítico, como desempenho.

Iteração de Projeto e Construção

Esta fase tem como principal foco garantir que o sistema está robusto o suficiente para ser entregue aos usuários. O principal produto da Iteração de Projeto e Construção é o sistema testado.

A atividade de teste é feita constantemente durante as fases de Iteração de Modelagem Funcional e Iteração de Projeto e Construção, apesar de não ser mencionada explicitamente. O sistema testado deve satisfazer todos os requisitos definidos, ou seja, tanto os essenciais quanto aqueles menos prioritários, que foram incluídos porque o tempo permitia.

Implementação

Ao contrário do que o nome possa sugerir, esta fase objetiva transferir o sistema do ambiente de desenvolvimento para o ambiente do cliente. Isso inclui, por exemplo, o treinamento dos usuários.

Um documento de revisão do projeto é elaborado. Este documento resume o que o projeto conseguiu alcançar, revendo todos os requisitos que foram identificados durante o desenvolvimento e informando a posição do sistema em relação a eles. Como DSDM é incremental, existem quatro possíveis posições:

- O sistema satisfaz completamente todos os requisitos elicitados, não havendo necessidade de um desenvolvimento posterior;
- Um novo conjunto de requisitos foi identificado durante o desenvolvimento. Nesse caso, é necessário retornar à fase de Estudo do Negócio;

- Uma parte menos importante das funcionalidades foi descartada devido a restrições de tempo. Deve-se retornar à Iteração de Modelagem Funcional, para que essas funcionalidades sejam incluídas;
- Algum aspecto técnico de menor importância não foi contemplado devido a pressões de prazo, mas agora pode ser tratado, retornando-se à Iteração de Projeto e Construção.

3.5.3 *Time Boxes*

O mecanismo usado pelo DSDM para permitir a flexibilidade dos requisitos é considerar períodos fixos de tempo para o cumprimento de um conjunto de atividades, chamado *time box*. Um *time box* pode ser composto por outros *time boxes* menores, que duram geralmente de 2 a 6 semanas.

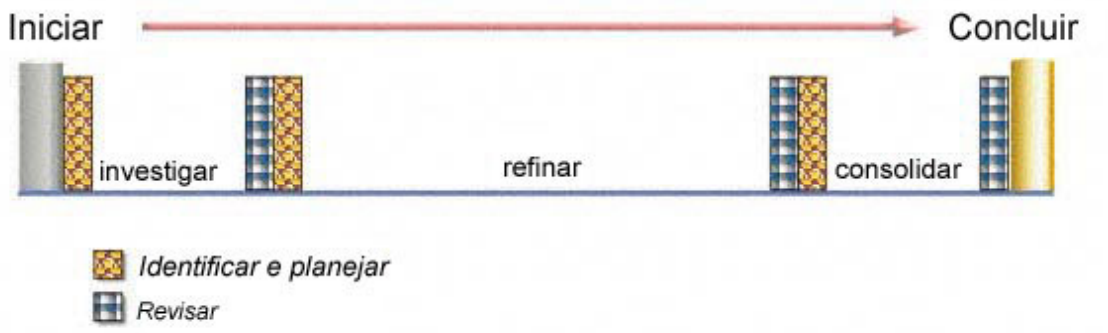


Figura 3-5 - Fases de um *time box*

Durante um *time box*, passa-se por três fases:

- Investigação: um passo para saber se a equipe está tomando a direção certa;
- Refinamento: aprimorar as ações com base nos resultados da revisão feita ao final da Investigação;
- Consolidação: ocorre ao final da *time box* e objetiva finalizar quaisquer atividades pendentes.

Cada *time box* possui uma data final, a qual não pode ser alterada, e um conjunto de requisitos a serem implementados. Este conjunto é formado por requisitos importantes, que são obrigatórios, e por requisitos de menor importância. A diversidade das diferentes prioridades dos requisitos é essencial para que se possa renegociar o escopo do que será entregue, caso as coisas não saiam exatamente conforme o planejado ou caso novos requisitos apareçam durante o desenvolvimento, pois a implementação de requisitos menos importantes pode ser postergada para iterações seguintes.

Os requisitos são priorizados de acordo com a regra chamada "MoSCoW". Esta regra é uma sigla para:

***M** ust Have, fundamental for projects success*

O

***S** hould Have, but the projects success don't rely on these*

***C** ould Have, can easily left out without impacting on the project*

O

***W** on't Have this time, can be left out this time and done in a latter date*

Ou seja, os requisitos são divididos nas seguintes categorias:

- Devem ser atendidos obrigatoriamente: são aqueles fundamentais para o sucesso do projeto;
- Devem ser atendidos preferencialmente: importantes, mas o sucesso do projeto não depende deles;
- Podem ser atendidos: requisitos que podem ser descartados sem impacto para o projeto;
- Não há tempo para atender: requisitos cuja implementação pode ser postergada para uma data posterior;

Segundo os autores, as letras "O" servem somente para compor a sigla, mas poderiam muito bem significar *On time* (no prazo) e *On budget* (no custo previsto), que são objetivos da metodologia.

Essa classificação serve como base para todas as tomadas de decisões durante o projeto. Isso é necessário porque a duração de uma *time box* é fixa e, por isso, o que será entregue ao final pode variar devido a restrições de tempo. Dessa forma, uma priorização sobre o que falta ser feito é necessária, para que as tarefas essenciais sejam cumpridas antes de outras menos críticas. As prioridades são revistas constantemente durante o projeto, para garantir que sempre refletem as necessidades do cliente.

3.6 Crystal Clear

Crystal é a mais simples de uma família de metodologias proposta por Alistair Cockburn [44]. Ele defende a idéia de que um projeto tem maiores chances de sucesso se utilizar uma metodologia adequada às suas características, tais como tipo de aplicação, tamanho da equipe, etc. Por isso, Cockburn propõe uma série de metodologias, cada uma voltada para um tipo diferente de projeto.

Crystal Clear, a mais leve das 4 metodologias da família, é voltada para projetos onde o software a ser desenvolvido não é crítico, a equipe é bastante pequena (em média 4 e, no máximo, 6 membros) e trabalha no mesmo ambiente, de preferência, na mesma sala. Na equipe deve haver pelo menos um desenvolvedor experiente, que atuará como um líder técnico. Crystal Clear baseia-se fortemente nos valores da comunicação, criatividade e no lado humano da equipe, levando em conta as qualidades e defeitos de cada membro.

3.6.1 Características Gerais

O desenvolvimento é iterativo e incremental. Os requisitos são levantados junto aos usuários e descritos na forma de pequenas histórias, que não precisam detalhar tudo que deve ser feito. O contato com o usuário deve ser facilitado, de modo que se possa esclarecer os detalhes dos requisitos sempre que necessário. Assim, as histórias funcionam apenas como um lembrete do que deve ser feito.

O conjunto de requisitos inicial é priorizado pelo usuário, enquanto o líder técnico deve definir a tecnologia e a arquitetura a serem utilizadas. É feito, então, um planejamento, com a participação de toda a equipe, das funcionalidades que farão parte do próximo *release*. Os *releases* são feitos a cada 2 ou 3 meses, de modo a se obter um *feedback* do usuário e se ter uma noção real do andamento do projeto.

Entre os *releases* oficiais, alguns protótipos podem ser mostrados aos usuários para esclarecer dúvidas sobre os requisitos. Estar sempre produzindo algo funcional e validando constantemente estes resultados com o usuário é um dos princípios básicos de Crystal Clear.

A fase de análise e projeto é feita de maneira simples e informal. Os modelos são feitos em quadros-negros e não necessitam de um nível de detalhes grande e nem precisam obedecer a uma notação. A equipe escolhe a forma de trabalhar que lhe é mais confortável. São feitas revisões constantes do projeto do sistema.

A implementação também não possui nenhuma restrição no modo de desempenhar as atividades. Entretanto, os testes devem ser constantes e, de preferência, automatizados. Quando o produto é finalizado, são elaborados o manual do usuário e o *build* oficial. O teste final de aceitação é feito por um grupo de usuários que deve ler o manual e tentar usar o sistema. Em alguns casos, um rascunho do manual do usuário é feito no início do desenvolvimento, de modo que se possa ter uma visão de todas as funcionalidades e, com isso, uma idéia geral do tamanho do sistema.

Segundo Cockburn, você pode dizer que está utilizando Crystal Clear se:

- A equipe se comunica constantemente;

- São feitos *releases* constantes, no máximo, a cada 3 meses. O cronograma e o acompanhamento do projeto são feitos através de marcos (*milestones*), que sejam, de preferência, versões executáveis do sistema em vez de documentos;
- Existe um usuário do sistema fazendo parte da equipe, mesmo que em tempo parcial. Ele é responsável por ajudar na definição da interface gráfica e a validá-la. Usuários reais do sistema devem validá-lo ao final de uma seqüência de iterações, antes de o sistema ser efetivamente implantado;
- O projeto possui testes de regressão e sua execução é automatizada. É feita alguma forma de revisão de código por pares (*peer review*) diariamente, semanalmente ou a cada iteração;
- A equipe conhece a missão do projeto. Os requisitos são orientados a funcionalidades (como casos de uso) e existe uma descrição do projeto do sistema.

3.6.2 Papéis e Artefatos

Existem quatro papéis principais, que são desempenhados por pessoas distintas:

- **Patrocinador (*sponsor*):** é a pessoa que está pagando pelo projeto. O patrocinador é responsável por definir a missão do projeto, ou seja, uma descrição breve do sistema e seu objetivo principal.
- **Analista sênior:** um analista experiente que atuará como líder técnico da equipe;
- **Usuário:** conhece os requisitos e o domínio da aplicação. Ajuda na definição dos casos de uso e dos rascunhos das telas;
- **Analistas-programadores:** são a equipe de desenvolvimento propriamente dita. Executam atividades de análise, projeto, implementação e definição dos casos de teste.

Além deles, existem papéis secundários, que podem ser desempenhados pelas mesmas pessoas acima ou por outras pessoas:

- **Especialista no negócio:** entende bem o domínio da aplicação que está sendo desenvolvida. Ele elabora os casos de uso do sistema, os rascunhos das telas e produz o documento de requisitos, contando com a ajuda do

Usuário. Pode ser desempenhado pelo patrocinador, pelo usuário ou pelo analista sênior;

- **Coordenador:** responsável por organizar os *releases*, definindo seus escopos e datas, e acompanhar o progresso do projeto, mantendo um *status* atualizado e uma lista de riscos. Este papel pode ser desempenhado pelo analista sênior;
- **Testador:** responsável por testar o sistema, elaborar os resultados de testes e os relatórios de defeitos. Pode ser desempenhado pelos analistas-programadores;
- **Escritor:** responsável por redigir o manual do usuário. Algumas equipes preferem redigir um rascunho do manual antes de iniciar o desenvolvimento. Dessa forma, elas pensam, de uma forma superficial, sobre o sistema inteiro e têm uma idéia da sua complexidade. Este papel pode ser desempenhado pelos analistas-programadores.

A metodologia recomenda a produção de uma série de artefatos. Entretanto, eles devem ser "leves", ou seja, devem ser artefatos simples e concisos, e não precisam ser apresentados de maneira rebuscada ou elaborada. O calendário do *release*, por exemplo, pode estar colado na parede ou ser enviado por email para todos. Abaixo, na Tabela 1, são apresentados os artefatos sugeridos, agrupados pelos respectivos papéis responsáveis pela sua produção.

Responsável	Artefato	Descrição do Artefato
Patrocinador	Missão do projeto	Descrição, em linhas gerais, dos objetivos do projeto
Coordenador	Seqüência do *release*	A ordem em que as funcionalidades serão entregues aos usuários
	Calendário do *release*	O planejamento das datas em que os *releases* serão feitos
	Lista de riscos	A relação de riscos levantados para o projeto
	Status do projeto	A situação atual do progresso do projeto
Analista Sênior	Estrutura da equipe (parte da metodologia)	Mapeamento das pessoas nos papéis da metodologia
	Metodologia	Adaptações da metodologia específicas para o projeto, como por exemplo, a linguagem de modelagem a ser utilizada
	Projeto do Sistema	Modelo de alto nível da organização do sistema
Especialista no negócio	Pares "ator-objetivo" (parte do documento de requisitos)	Visão geral dos atores que interagem com o sistema e seus respectivos objetivos
	Detalhamento dos casos de uso (parte do documento de requisitos)	Descrição dos casos de uso, com cenários principais e alternativos
	Documento de requisitos	Descrição dos requisitos que o sistema deve cumprir
Usuário	Ajuda nos casos de uso e rascunhos	Ajuda a definição dos casos de uso

	de telas	e das telas do sistema
Analistas-programadores	Rascunhos de telas	Esboços das telas do sistema
	Rascunhos de projeto	Esboços do projeto do sistema
	Modelo comum de objetos	Modelo detalhado do sistema, com as classes que o compõem
	Código-fonte	Implementação do sistema na linguagem escolhida.
	Sistema empacotado	Sistema compilado e pronto para entrega ao usuário
	Código de migração	Código necessário para a migração de dados de um eventual sistema legado.
	Casos de teste	Os casos testes utilizados para testar o sistema
Testador	Resultados de testes	Resultados da execução dos testes
	Relatórios de defeitos	Relação dos problemas que devem ser corrigidos, encontrados nos testes
Escritor	Manual do usuário	Documentação sobre como utilizar o sistema

Tabela 1 - Papéis e Responsabilidades da Crystal Clear

3.7 Considerações Finais

Neste capítulo apresentamos as metodologias ágeis, que propõem uma abordagem mais simples e objetiva para atacar os problemas do desenvolvimento de software. Além dos conceitos gerais deste tipo de metodologia, foram apresentados seus principais representantes. Apesar das diferenças, podemos identificar vários pontos em comum entre elas, como:

- Desenvolvimento iterativo e incremental, com iterações de curta duração;
- A elaboração de documentos é substituída pela ênfase na comunicação constante;
- Busca de um maior envolvimento do usuário no desenvolvimento, principalmente no planejamento das iterações e *releases*.

Estas metodologias são voltadas para equipes pequenas. Devido à simplicidade, ou seja, aos poucos artefatos e atividades, em comparação com metodologias pesadas, as metodologias ágeis são de fácil implantação, já que existem menos atividades a serem aprendidas e quase não há necessidade de instanciação.

Esta simplicidade, entretanto, pode ser um obstáculo caso a organização esteja querendo se alinhar a um modelo de qualidade, como o *Capability Maturity Model for Software* [10] ou as normas ISO 9001 [46] e ISO 15504 [47]. Estes modelos, em geral, pedem mais artefatos e atividades do que as metodologias ágeis recomendam. Além disto, a ausência de atividades detalhadas faz com que as metodologias ágeis dependam fortemente das competências pessoais dos membros da equipe.

Capítulo 4: Team Software Process

4.1 Introdução

Atualmente, a grande maioria da produção de software é feita em equipes, em virtude do enorme aumento da complexidade do software e dos cronogramas cada vez mais agressivos, resultado da alta competitividade. Para que estas equipes sejam realmente eficientes e construam um produto de qualidade, não basta somente ter pessoas altamente capacitadas. De acordo com Humphrey [10], elas precisam de liderança e guias para realizar seu trabalho, de modo a trabalhar como uma equipe efetiva. Uma equipe efetiva é definida como sendo aquela que possui as seguintes características:

- Os membros são capacitados;
- O objetivo da equipe é importante, definido, visível e realista;
- Os recursos da equipe são adequados ao seu trabalho;
- A equipe é motivada e comprometida em cumprir seu objetivo;
- Os membros da equipe cooperam e dão suporte uns aos outros;
- Os membros da equipe são disciplinados no seu trabalho.

Buscando fornecer essas informações de liderança e formação de equipes, Watts Humphrey, do Instituto de Engenharia de Software (*Software Engineering Institute* – SEI) da Universidade de Carnegie-Mellon propôs, em 1999, o *Team Software Process* (TSP) [10], um processo de desenvolvimento de software cujo objetivo é formar equipes de engenharia realmente efetivas e guiá-las no desenvolvimento de um sistema.

O TSP vem complementar os processos e modelos já propostos por Humphrey com o objetivo de disciplinar o processo de produção de software: o *Personal Software Process* (PSP) [45] e o *Capability Maturity Model for Software* (SW-CMM) [11]. O PSP descreve um

processo individual focado no desenvolvedor, ou seja, um conjunto de atividades que cada desenvolvedor deve realizar para aumentar sua produtividade e a qualidade do que está desenvolvendo. Entretanto, o PSP não trata de questões de trabalho em equipe e nem da organização onde o desenvolvedor está inserido. O TSP exige, como pré-requisito, que os desenvolvedores sejam treinados em PSP.

O SW-CMM é um modelo de maturidade para organizações que desenvolvem software. Ele define níveis de maturidade que a empresa deve atingir para melhorar seu processo de desenvolvimento, e descreve o que deve ser feito para atingir estes níveis. Por ser voltado para organizações, ele diz apenas **o quê** deve ser feito e não **como** fazer, ficando a cargo de cada organização implementar as práticas do CMM de acordo com sua realidade e cultura.

Por ser relativamente recente, a bibliografia de TSP ainda é escassa. Aqui, estaremos descrevendo a versão introdutória do TSP apresentada em [48], chamada TSPi. Segundo Humphrey, ela é uma simplificação do TSP voltada para o ensino do processo. Apesar de simplificada, esta é a versão mais detalhada disponível no *website* do SEI[8]. Os demais relatórios técnicos, que descrevem a estrutura do TSP, disponíveis, como [10] e [49], não entram em detalhes sobre as atividades que devem ser executadas, bem como suas entradas e saídas.

No TSP, o desenvolvimento do sistema é feito em ciclos, e cada ciclo passa por uma série de fases: Lançamento, Estratégia de Desenvolvimento, Planejamento do Desenvolvimento, Requisitos, Projeto, Implementação do Produto, Testes de Sistema e Integração, e *Postmortem*.

Cada fase possui um objetivo e um conjunto de atividades que devem ser executadas para cumpri-lo. Estas atividades são apresentadas na forma de *scripts*, que indicam a ordem das atividades e o que deve ser feito em cada uma. As atividades de desenvolvimento são fortemente baseadas em inspeções formais dos artefatos produzidos, coleta de métricas e planejamento rigoroso. A Figura 4-1 ilustra a estrutura do desenvolvimento de um projeto utilizando TSP.

[8] http://www.sei.cmu.edu/tsp

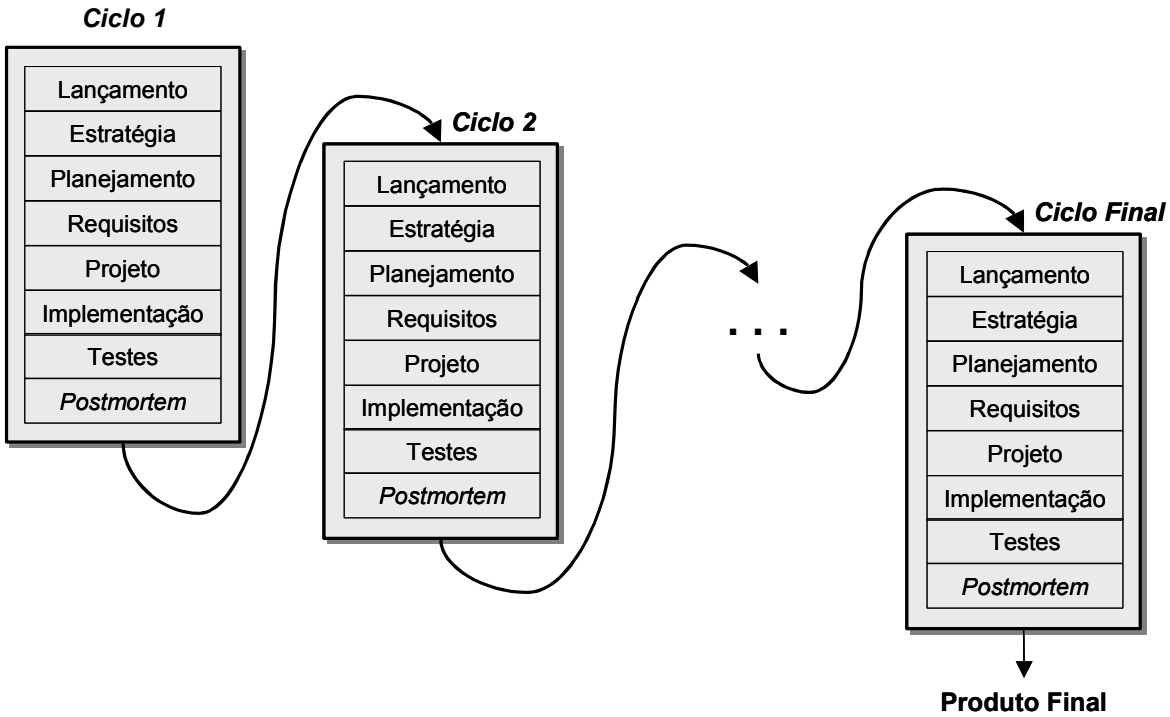


Figura 4-1 - Estrutura de um projeto utilizando o TSP

Além disto, o TSP define um conjunto de papéis que devem ser assumidos pelos membros da equipe. São eles: Líder da Equipe, Gerente de Desenvolvimento, Gerente de Planejamento, Gerente de Qualidade/Processo e Gerente de Suporte. Estes papéis determinam conjuntos de responsabilidades, que são necessários para o bom funcionamento da equipe.

Embora seja um processo novo na comunidade de engenharia de software, vários benefícios da adoção de práticas do TSP já são conhecidos. Destacam-se principalmente sua função como acelerador na implantação do SW-CMM, como em [55] e [57], e para melhorar o gerenciamento de projetos e até mesmo de empresas de pequeno porte, como pode ser visto em [50], [52] e [54]. Dentre os principais ganhos, destacam-se o aumento da produtividade e a menor quantidade de erros no software após sua liberação para o cliente.

As diversas fases que compõem o processo, com suas respectivas atividades e *scripts*, são apresentadas a seguir, na Seção 4.2. Os papéis, com suas respectivas responsabilidades, atividades e perfis necessários são apresentados na Seção 4.3.

4.2 Fases do *Team Software Process*

4.2.1 Lançamento

A primeira fase do TSP, chamada Lançamento (*Launch*) tem como objetivo iniciar o processo de formação da equipe. Para isso, são divididos os papéis e definidos metas da

equipe e de cada papel. Estas metas devem ser mensuráveis, para que seja possível determinar se foram ou não cumpridas. Assim, uma meta da equipe, como "terminar o projeto no prazo", pode ser considerada cumprida pela equipe se o produto for entregue com até 5 dias de atraso ou adiantamento, por exemplo.

Além das definições dos papéis e das metas, a equipe deve entrar em acordo sobre como serão conduzidas as reuniões semanais para o acompanhamento do projeto e sobre as métricas que devem ser reportadas nestas reuniões. A Tabela 2 apresenta o script LAU1, utilizado no primeiro lançamento.

Propósito		**Iniciar o time no primeiro ciclo de desenvolvimento**
Critério de Entrada		▪ Todos os estudantes concluíram satisfatoriamente um curso de PSP ▪ Os estudantes leram os Capítulos 1, 2 3 e Apêndice A do livro-texto
Geral		
Passo	**Atividades**	**Descrição**
1	Apresentar Visão Geral do Curso	O instrutor descreve os objetivos do curso de TSPi ▪ O que se espera que os estudantes cumpram ▪ Como seu trabalho será avaliado ▪ Os princípios básicos de trabalho em equipe ▪ O processo TSPi
2	Obter Informações do Estudante	O instrutor explica o critério para formar as equipes ▪ A informação necessária para fazer as atribuições corretas ▪ Os papéis na equipe, suas qualificações e suas responsabilidades O instrutor também pede que os estudantes ▪ Preencham e devolvam o formulário INFO antes do fim da aula ▪ Leiam o Capítulo 4 e o Apêndice B do livro-texto ▪ Leiam os capítulos sobre os papéis que os interessam
3	Descrever Objetivos do Produto	O instrutor descreve os objetivos do produto ▪ Os objetivos críticos do produto que precisam ser satisfeitos ▪ Os objetivos desejáveis e opcionais ▪ O critério para avaliar o produto finalizado
4	Realizar Atribuição da Equipe	O instrutor dá aos estudantes suas equipe e seus papéis
5	Definir Metas da Equipe	O instrutor descreve como definir metas ▪ Porque as metas são importantes e metas típicas para a equipe e para os papéis
6	Descrever Reuniões da Equipe	O instrutor explica as reuniões da equipe, seu propósito e como são conduzidas ▪ O propósito, calendário e forma de registro da reunião ▪ Dados necessários semanalmente
7	Realizar Primeira Reunião da Equipe	O líder da equipe coordena a primeira reunião da sua equipe ▪ Discute os papéis dos membros da equipe ▪ Discute e define as metas do ciclo 1 ▪ Estabelece uma hora padrão para o encontro semanal da equipe ▪ Define juntamente com a equipe uma hora específica na qual, semanalmente, todos os membros devem fornecer seus dados para o Gerente de Planejamento
8	Apresentar Dados Necessários	O Gerente de Planejamento revê para a equipe os ▪ Dados necessários semanalmente de cada membro da equipe ▪ Relatórios a serem gerados e providos para a equipe com estes dados
9	Iniciar Projeto	A equipe começa a trabalhar no projeto, utilizando o *script* STRAT1
Critério de Saída		▪ Cada estudante completou e submeteu o formulário INFO ▪ Os times de desenvolvimento estão formados e os papéis distribuídos ▪ O instrutor descreveu os objetivos gerais do produto ▪ O instrutor reviu e discutiu o TSPi e as metas das equipes e dos papéis ▪ A equipe definiu as metas do ciclo 1, horário das reuniões semanais e os dados a serem reportados semanalmente

Tabela 2 - Script LAU1

4.2.2 Estratégia de Desenvolvimento

Na fase Estratégia de Desenvolvimento, a equipe define a estratégia que será usada para executar o primeiro ciclo do projeto. Isto significa, na prática, definir que funcionalidades do sistema serão produzidas em cada ciclo.

Para esta definição, é feito um projeto conceitual do sistema, ou seja, são definidos os principais componentes que o sistema terá, com base no conhecimento que se tem até o momento, as principais funcionalidades de cada um e uma estimativa do tamanho de cada componente. Isto servirá de subsídio para o planejamento inicial do ciclo. Além disto, são feitas estimativas iniciais de tempo para cada funcionalidade. Toda a equipe participa deste processo.

Nesta fase, também é produzido (no primeiro ciclo) ou atualizado (nos ciclos posteriores) o Plano de Gerência de Configuração e são identificados os riscos do projeto, que são analisados em termos de probabilidade de ocorrência, impacto e ações de mitigação. Os passos da fase Estratégia de Desenvolvimento para o primeiro ciclo são apresentados na Tabela 3.

Propósito		**Guiar a equipe na produção de uma estratégia de desenvolvimento TSPi e nas estimativas iniciais de tamanho e tempo**
Critério de Entrada		▪ Os estudantes leram o Capítulo 4 do livro-texto ▪ O instrutor reviu e discutiu o processo TSPi ▪ O instrutor descreveu os objetivos gerais do produto ▪ As equipes de desenvolvimento estão formadas e com os papéis atribuídos ▪ As equipes concordaram em metas para o seu trabalho
Geral		A estratégia de desenvolvimento especifica a ordem na qual as funcionalidades do produto são definidas, projetadas, implementadas e testadas, definindo ▪ A maneira como o produto será melhorado nos ciclos futuros ▪ Como dividir o trabalho de desenvolvimento entre os membros da equipe A estratégia de desenvolvimento é produzida no início do processo para guiar as estimativas de tamanho e o planejamento de recursos. Esta estratégia deve estar sempre atualizada para refletir as decisões tomadas durante o projeto As estimativas iniciais de tamanho e tempo ▪ Cobrem o trabalho planejado para cada ciclo de desenvolvimento ▪ Fornecem a base para alocar o trabalho entre os membros da equipe
Passo	**Atividades**	**Descrição**
1	Apresentar Visão Geral da Estratégia	O instrutor descreve a estratégia de desenvolvimento ▪ O que é uma estratégia, como ela é produzida e utilizada ▪ Critérios para uma estratégia efetiva ▪ A necessidade e a maneira de se produzir estimativas de tamanho e de tempo
2	Estabelecer Critério para Estratégia	▪ O Gerente de Desenvolvimento lidera a discussão sobre o critério para a estratégia ▪ O registrador da reunião (Gerente de Qualidade/Processo) documenta estes critérios e provê cópias para os membros da equipe e para o instrutor
3	Produzir o Projeto Conceitual	O Gerente de Desenvolvimento lidera a equipe na produção do projeto conceitual para o produto inteiro

4	Selecionar Estratégia de Desenvolvimento	O Gerente de Desenvolvimento lidera a equipe na produção da estratégia de desenvolvimento. Isto envolve: ▪ Propor e avaliar estratégias alternativas ▪ Alocar funcionalidades do produto para cada ciclo de desenvolvimento ▪ Definir como subdividir e posteriormente integrar o produto
5	Produzir Estimativas Preliminares	O Gerente de Planejamento lidera a equipe na produção das estimativas preliminares de tamanho e tempo, que devem incluir: ▪ Estimativas de tamanho e tempo para todos os produtos do ciclo atual ▪ Estimativas grosseiras para produtos dos ciclos posteriores
6	Avaliar Riscos	Identificar e avaliar os riscos do projeto e colocá-los no formulário ITL (*issue tracking log*)
7	Documentar a Estratégia	O registrador da reunião documenta a estratégia selecionada
8	Produzir o Plano de Gerência de Configuração	O Gerente de Suporte produz o plano de gerência de configuração: ▪ Identifica o comitê de controle de configuração e seus procedimentos ▪ Especifica qualquer necessidade de ferramentas de suporte e infra-estrutura ▪ Revê os procedimentos com a equipe, buscando seu aceite
Critério de Saída		▪ Uma estratégia de desenvolvimento documentada e aprovada ▪ Estimativas de tamanho e tempo completadas e documentadas para todos os produtos a serem desenvolvidos no ciclo atual ▪ Estimativas completadas e documentadas para produtos a serem desenvolvidos nos ciclos subseqüentes ▪ Procedimento documentado de gerência de configuração ▪ Riscos e pendências registrados no formulário ITL ▪ Design conceitual e o formulário STRAT preenchido ▪ Documentação do projeto atualizada

Tabela 3 - Script STRAT1

4.2.3 Planejamento do Desenvolvimento

Após o planejamento macro dos ciclos do projeto, feito na fase de Estratégia de Desenvolvimento, a equipe deve elaborar o planejamento detalhado das atividades do ciclo atual. Isto é feito na fase de Planejamento do Desenvolvimento. Nesta fase, são produzidos o planejamento de atividades de cada membro da equipe e o Plano de Qualidade do projeto.

No planejamento das atividades do ciclo atual, a equipe inteira participa, coordenada pelo Gerente de Planejamento, da definição de que funcionalidades devem ser elaboradas neste ciclo e de suas respectivas estimativas de tamanho. A partir disto, são derivadas as atividades necessárias, como elaboração de documentos e codificação, e a equipe estima, em conjunto, o esforço necessário para realizar cada uma. Nesta etapa, o TSP aconselha que atividades individuais, cuja estimativa de duração sejam maiores que 10 horas, sejam quebradas em atividades menores. Isto traz maior visibilidade para o progresso das atividades, pois em uma atividade de grande duração é difícil precisar que porcentagem dela já foi concluída.

De posse das estimativas de esforço de cada atividade, o próximo passo é elaborar o cronograma das atividades e verificar se o planejamento é viável. Caso note-se que o trabalho a ser realizado é maior do que o tempo necessário, pode-se adotar duas estratégias: aumentar o número de horas a serem trabalhadas ou reduzir o escopo do ciclo. O que não

se deve fazer é reduzir as estimativas para que as atividades sejam encaixadas no cronograma, pois, segundo Humprey, as estimativas iniciais dos engenheiros geralmente são bastante otimistas. Assim, se ainda forem reduzidas para se adequar ao cronograma, o planejamento corre grande risco de ser impossível de ser cumprido.

Nesta fase também é produzido ou atualizado o Plano de Qualidade do projeto. Esta atividade é executada após a elaboração do cronograma, pois usa como entrada as horas planejadas para cada fase. O Plano de Qualidade define os critérios de qualidade que devem ser satisfeitos tanto pelos artefatos produzidos (ex: defeitos por página ou por linhas de código), quanto pela equipe (ex: quantidade de erros encontrados em inspeção ou de defeitos injetados por fase). A elaboração deste plano é de responsabilidade do Gerente de Qualidade/Processo.

Em seguida, é elaborado o plano de atividades de cada engenheiro. Aqui, deve-se levar em consideração as horas necessárias para se desempenhar os papéis do TSP. Também se deve fazer o balanceamento do trabalho entre a equipe, pois alguns podem acabar sobrecarregados na divisão das atividades. O ideal é que o total de horas trabalhadas de cada engenheiro seja acompanhado à medida que as atividades são distribuídas. Por fim, os planos são consolidados e distribuídos para a equipe. O *script* utilizado no planejamento é apresentado na Tabela 4.

A participação da equipe no planejamento das atividades é importante para garantir o comprometimento da equipe com as atividades e seus prazos. Caso não haja um planejamento das atividades, a equipe acaba se comprometendo com a data escolhida por outras pessoas, em geral, a alta gerência, sem saber se pode ou não alcançar esta meta. Sem crer que esta data é viável, dificilmente a equipe terá comprometimento em cumpri-la.

Propósito	**Guiar a equipe na produção de cronogramas e planos de tarefas e de qualidade, individuais e para toda a equipe, para o ciclo de desenvolvimento atual.**
Critério de Entrada	▪ A equipe possui uma estratégia de desenvolvimento e um projeto conceitual ▪ Os estudantes leram o Capítulo 5 do livro-texto
Geral	O plano de tarefas define: ▪ O tempo necessário para desempenhar cada tarefa do processo ▪ Uma ordem inicial na qual as tarefas devem se executadas ▪ O quanto cada tarefa representa em relação ao total de tarefas O cronograma fornece: ▪ O tempo planejado de cada engenheiro em cada semana do projeto ▪ O total de horas planejadas para a equipe por semana ▪ A semana prevista de conclusão de cada tarefa ▪ O valor planejado de cada semana, ou seja, o quanto a conclusão das atividades de cada semana representa em relação à conclusão do projeto como um todo Caso o plano de tarefas e o cronograma indiquem que o projeto não pode ser concluído a tempo, reajuste a estratégia e faça um replanejamento.

Passo	Atividades	
1	Apresentar Visão Geral do Planejamento	O instrutor descreve o processo de planejamento ▪ O plano de tarefas e o cronograma e como eles são produzidos ▪ O plano de qualidade e como ele é produzido
2	Registrar as Estimativas de Tamanho no Formulário STRAT	Baseado no projeto conceitual e no formulário STRAT, produzido na fase Estratégia de Desenvolvimento, o Gerente de Planejamento lidera a equipe ▪ Na identificação de outros artefatos a serem produzidos e suas estimativas de tamanho ▪ No registro destas informações no formulário STRAT e qualquer outro dado de tamanho no formulário SUMS
3	Produzir o Plano de Tarefas	O Gerente de Planejamento lidera a equipe ▪ Na produção de uma lista de tarefas com as estimativas de tempo da equipe e de cada engenheiro ▪ No registro destes dados no formulário TASK.
4	Produzir o Cronograma	O Gerente de Planejamento obtém o número estimado de horas que cada membro da equipe planeja se dedicar ao projeto a cada semana e: ▪ Registra as horas semanais no formulário SCHEDULE ▪ Produz os formulários TASK e SCHEDULE para a equipe ▪ Ajusta o plano caso as horas sejam inadequadas
5	Produzir o Plano de Qualidade	O Gerente de Qualidade/Processo lidera a equipe ▪ Na revisão dos objetivos de qualidade ▪ Na estimativa das metas de defeitos removidos e injetados ▪ Na geração e avaliação de rascunhos dos planos SUMP e SUMQ ▪ Nos ajustes necessários ao processo para obter um plano satisfatório
6	Produzir os Planos Individuais dos Engenheiros	O Gerente de Planejamento ajuda os engenheiros na elaboração de seus planos individuais ▪ Alocação das tarefas aos membros da equipe ▪ Estimativa do tempo para executar cada tarefa ▪ Registro dos dados nos formulários TASK e SCHEDULE. ▪ Produção do cronograma com valor planejado e datas de conclusão das tarefas
7	Balancear a Carga de Trabalho da Equipe	O Gerente de Planejamento lidera a equipe ▪ Na identificação de diferenças nas cargas de trabalho ▪ Na re-alocação das tarefas para minimizar o cronograma ▪ Na produção de planos balanceados para os engenheiros ▪ Na produção de um plano consolidado para a equipe (formulários TASK, SCHEDULE, SUMP e SUMQ)
Critério de Saída		▪ Formulários TASK e SCHEDULE para a equipe e para cada engenheiro preenchidos ▪ Formulários SUMP, SUMQ e SUMS preenchidos ▪ Documentação do projeto atualizada

Tabela 4 - Script PLAN1

4.2.4 Requisitos

A fase de Requisitos inicia o desenvolvimento do produto propriamente dito. Nesta fase, são levantadas as necessidades do cliente e, com base nisto, produzidos a Especificação de Requisitos de Software e o Plano de Testes de Sistema.

As atividades desta fase são basicamente voltadas para produção destes dois documentos e suas inspeções e revisões com o cliente. O TSP sugere que estes artefatos sejam produzidos na forma de documentos, cuja elaboração é dividida entre os membros da equipe. Ao final, o documento de requisitos deve ser revisado pelo usuário.

A Especificação de Requisitos de Software irá conter uma descrição do que o sistema deve fazer, do ponto de vista do usuário, e critérios para avaliar seu correto funcionamento. Apesar de ser sugerida uma estrutura para o documento de requisitos, não

são apresentadas sugestões de técnicas específicas para a elicitação ou documentação dos requisitos, como casos de uso ou cenários.

O Plano de Testes de Sistema irá descrever como cada requisito será testado e quais os eventuais recursos necessários para estes testes. Ele também pode conter um planejamento inicial das atividades necessárias para execução destes testes, como geração de massa de dados de teste, com as respectivas estimativas. Este artefato, assim como a Especificação de Requisitos de Software, deve ser inspecionado e, após aprovado, colocado na linha-base[9] do sistema, ou seja, sob gerência de configuração. Os passos desta fase estão descritos no *script* apresentado na Tabela 5.

Propósito		**Guiar a equipe no desenvolvimento e inspeção de requisitos para o ciclo atual de um projeto de desenvolvimento**
Critério de Entrada		▪ A equipe possui uma estratégia de desenvolvimento e um plano ▪ O estudantes leram o Capítulo 6, as seções de testes do Capítulo 9 e a Declaração de Necessidade[10]
Geral		O processo de desenvolvimento de requisitos produz uma Especificação de Requisitos de Software (ERS), que define: ▪ As funções que o produto deve prover ▪ Descrições de casos de uso para execução normal ou anormal de uma função A equipe deve ser cuidadosa no detalhamento dos requisitos ▪ Sem experiência com aplicações similares, funções aparentemente simples podem ser mais trabalhosas do que o esperado ▪ Em geral, é prudente adicionar funções em pequenos incrementos ▪ Caso haja tempo sobrando, adicione mais incrementos
Passo	**Atividades**	
1	Apresentar Visão Geral do Processo de Requisitos	O instrutor descreve o processo de requisitos e seus produtos finais ▪ Como o processo de requisitos é executado ▪ Como a inspeção dos requisitos é conduzida e registrada
2	Revisar a Declaração de Necessidade	O Gerente de Desenvolvimento lidera a equipe na revisão da Declaração de Necessidade do produto e na formulação de perguntas para o instrutor sobre ▪ As funcionalidades a serem executadas pelas várias versões do produto ▪ Como estas funcionalidades serão utilizadas
3	Esclarecer a Declaração de Necessidade	O Gerente de Desenvolvimento fornece as questões consolidadas ao instrutor, que discute as respostas com a equipe
4	Definir as Tarefas de Documentação de Requisitos	O Gerente de Desenvolvimento lidera a equipe ▪ Na estruturação do documento de ERS e no trabalho necessário para produzi-lo
5	Alocar Tarefas	O Líder de Equipe ajuda a alocar as tarefas entre os membros da equipe e obtém comprometimento de quando eles irão completar estas tarefas
6	Documentar Requisitos	Cada membro da equipe ▪ Produz e revisa suas porções do documento de ERS ▪ Entrega-as ao Gerente de Desenvolvimento O Gerente de Desenvolvimento produz o rascunho da ERS.
7	Elaborar Plano de Testes de Sistema	O Gerente de Desenvolvimento lidera a equipe na produção e revisão do Plano de Testes de Sistema.

[9] "Uma especificação ou produto que foi formalmente revisto e acordado, servindo então como base para o desenvolvimento futuro, só podendo ser alterado através de procedimentos formais de controle de mudanças" (IEEE-STD-610)

[10] Um documento produzido pelo usuário, descrevendo o que o sistema deve fazer.

8	Realizar Inspeção dos Requisitos e do Plano de Testes de Sistema	O Gerente de Qualidade/Processo lidera equipe na: ▪ Inspeção do rascunho da ERS e no Plano de Testes de Sistema ▪ Identificação de questões e problemas ▪ Definição de quem resolverá cada questão ou problema e quando ▪ Documentação da inspeção no formulário INS
9	Atualizar Documentação dos Requisitos	O Gerente de Desenvolvimento obtém as seções atualizadas da ERS e ▪ Combina-as na versão final da ERS ▪ Verifica a rastreabilidade com a Declaração de Necessidade e outras fontes
10	Revisar a ERS com o usuário	▪ O Gerente de Desenvolvimento provê uma cópia da ERS final para o usuário, para aprovação ▪ Após a aprovação, a equipe conserta quaisquer problemas identificados
11	Criar a Linha-base de Requisitos	▪ O Gerente de Suporte adiciona a ERS à linha-base do projeto.
Critério de Saída		▪ Um documento de ERS e um Plano de Testes de Sistema finalizados e inspecionados ▪ Um formulário INS preenchido para a inspeção de requisitos ▪ Dados de tempo, defeitos e tamanho registrados ▪ Documentação do projeto atualizada

Tabela 5 - Script REQ1

4.2.5 Projeto

No TSP, o projeto do sistema é feito em duas etapas. Na primeira, é elaborado o chamado projeto de alto nível que descreve os principais componentes do sistema, suas interfaces e como eles interagem entre si. Na segunda etapa, é elaborado o projeto detalhado destes componentes, onde cada um é dividido em unidades menores, que serão especificadas detalhadamente.

A fase Projeto do TSP trata somente do projeto de alto nível, ficando o projeto detalhado para a fase de Implementação. O objetivo da fase Projeto é elaborar uma descrição precisa de como os componentes do sistema serão organizados, para que não surjam muitas dúvidas quando estes componentes forem detalhados e para que sua integração seja pouco custosa, uma vez que as interfaces entre eles foram bem determinadas.

Nesta fase são produzidos dois artefatos: a Especificação de Projeto de Software e o Plano de Testes de Integração. Além deles, vários padrões a serem utilizados durante a fase, como convenções de nomes, mensagens de erro, padrões de representação do projeto (que modelos devem ser produzidos e em que nível de detalhe), formatos das interfaces, entre outros. Estas definições são necessárias pelo fato do processo de projeto do TSP não ser dependente de nenhuma técnica de modelagem ou linguagem específica, pois as técnicas e linguagens em uso evoluem rapidamente, o que tornaria o processo do TSP obsoleto.

Assim como na fase de Requisitos, os artefatos desta fase são produzidos em conjunto por toda a equipe. Ao final, o Gerente de Desenvolvimento é responsável por

consolidar as partes produzidas por cada membro da equipe em um só documento. As atividades desta fase são descritas pelo *script* DES1, apresentado na Tabela 6.

Propósito		**Guiar a equipe no desenvolvimento e inspeção das especificações de projeto para o ciclo atual de um projeto de desenvolvimento**
Critério de Entrada		▪ Uma estratégia de desenvolvimento e um plano ▪ Uma Especificação de Requisitos de Software finalizada e inspecionada ▪ Os estudantes leram o Capítulo 7 do livro-texto
Geral		O processo de projeto produz uma Especificação de Projeto de Software (EPS), que define a estrutura geral do produto para o ciclo ▪ Principais componentes do produto e a especificação de suas interfaces ▪ A alocação dos cenários de uso a estes componentes A EPS também especifica ▪ Padrões de arquivos e mensagens, definições e convenções de nomes ▪ Padrões e notação de projeto
Passo	**Atividades**	
1	Apresentar Visão Geral do Processo de Projeto	O instrutor descreve o processo de projeto e seus produtos finais ▪ Como o processo de projeto é executado e um exemplo de EPS ▪ Como a inspeção do projeto é conduzida e registrada
2	Elaborar Projeto de Alto Nível	O Gerente de Desenvolvimento lidera a equipe na: ▪ Definição da estrutura do produto no ciclo atual ▪ Nomeação dos componentes do produto ▪ Alocação dos cenários de uso a estes componentes ▪ Identificação das tarefas de projeto a serem completadas e documentadas
3	Definir Padrões de Projeto	O Gerente de Qualidade/Processo lidera o esforço na produção do glossário de nomes e padrões de projeto
4	Definir Tarefas de Projeto	O Gerente de Desenvolvimento lidera a equipe ▪ Na estruturação do documento de EPS e no trabalho necessário para produzi-lo
5	Alocar Tarefas	O Líder de Equipe ajuda a alocar as tarefas entre os membros da equipe e ▪ Obtém comprometimento de quando eles irão completar estas tarefas
6	Elaborar Especificação do Projeto	Cada membro da equipe ▪ Produz e revisa suas porções do documento de EPS ▪ Entrega-as ao Gerente de Desenvolvimento O Gerente de Desenvolvimento produz o rascunho da EPS.
7	Elaborar Plano de Testes de Integração	O Gerente de Desenvolvimento lidera a equipe na produção e revisão do Plano de Testes de Integração.
8	Realizar Inspeção do Projeto e do Plano de Testes de Integração	O Gerente de Qualidade/Processo lidera equipe na inspeção do rascunho da EPS e do Plano de Testes de Integração, para que: ▪ Cada caso de uso esteja coberto e referenciado no projeto ▪ O projeto esteja completo e correto ▪ O Plano de Integração de Testes esteja adequado ▪ Cada problema seja registrado e a responsabilidade por resolvê-lo esteja definida A inspeção é documentada no formulário INS e os defeitos registrados no formulário LOGD.
9	Atualizar o Projeto	O Gerente de Desenvolvimento obtém as seções atualizadas da EPS e ▪ Combina-as na versão final da EPS ▪ Verifica a rastreabilidade com a ERS
10	Criar a Linha-base de Projeto	O Gerente de Suporte adiciona a EPS à linha-base do projeto.
Critério de Saída		▪ Um documento de EPS e um Plano de Testes de Integração finalizados e inspecionados ▪ Padrões de projeto e o glossário de nomes definidos ▪ Formulários SUMP, SUMQ e INS atualizados ▪ Documentação do projeto atualizada

Tabela 6 - Script DES1

4.2.6 Implementação

Na fase de Implementação são desenvolvidos o projeto detalhado do sistema, sua implementação e seus testes unitários. Para estas atividades, são definidos uma série de padrões, como o padrão de codificação, de medição de tamanho dos artefatos, e de classificação dos defeitos encontrados, que complementam os padrões definidos na fase de Projeto. Esses padrões são utilizados para uniformizar a forma de trabalho da equipe. Podem ser definidos por um ou dois membros, mas devem ser revistos pela equipe inteira. O processo de definição dos padrões é liderado pelo Gerente de Qualidade/Processo.

O primeiro passo da fase de Implementação é produzir o projeto detalhado do sistema. Isto deve ser iniciado quando a fase de Projeto tiver sido concluída, ou seja, quando os componentes do sistema já tiverem sido divididos nas menores unidades de implementação possíveis, chamadas módulos. No projeto detalhado, são especificadas as funções e, caso exista, a máquina de estados de cada componente. Este projeto é registrado na Especificação de Projeto do Sistema.

Em seguida, é produzido o Plano de Testes Unitários, que irá conter os casos de testes a serem executados para cada componente. A definição dos casos de testes é feita antes da implementação, pois permite a identificação precoce de erros de projeto. Segundo Humphrey, desenvolver os casos de teste antes da codificação identifica mais erros de projeto do que a inspeção da Especificação do Projeto de Software e do que a execução dos testes unitários. Esta mesma estratégia é utilizada em Extreme Programming [8], onde é denominada *Test-first Design*.

O passo seguinte é a inspeção do projeto detalhado do sistema. Em seguida, é iniciado o processo de codificação dos módulos. Após a codificação, os engenheiros devem revisar o código, procurando por erros que estejam em seus *checklists* pessoais. Estes *checklists* contêm os erros mais comuns cometidos por cada engenheiro, e devem ser atualizados constantemente.

Uma vez concluída a implementação, o código é inspecionado e os testes unitários são executados, segundo os passos do script UT. Após a conclusão dos testes, cada componente é revisado pelo Gerente de Qualidade/Processo, para verificar se os critérios definidos no Plano de Qualidade do projeto estão satisfeitos. Esta análise irá determinar se o componente deve ser colocado sob gerência de configuração e liberado para integração, novamente inspecionado ou até mesmo descartado e desenvolvido novamente.

Estes passos estão descritos no *script* IMP1, apresentado na Tabela 7. A Tabela 8 apresenta o *script* UT, utilizado na execução dos testes unitários.

Propósito		**Guiar a equipe na implementação e inspeção do software para o ciclo atual de um projeto de desenvolvimento**
Critério de Entrada		A equipe possui uma estratégia de desenvolvimento e um plano ▪ ERS, EPS e o glossário de nomes ▪ Padrão de codificação e outros padrões documentados ▪ Os estudantes leram o Capítulo 8 do livro-texto
Geral		O processo de implementação produz um produto revisto, inspecionado e testado unitariamente, que deve: ▪ Cobrir completamente as funções e cenários de uso da ERS e da EPS ▪ Estar em conformidade com os padrões de projeto e codificação definidos ▪ Seguir os processos PSP2.1 ou PSP3[11]
Passo	**Atividades**	
1	Apresentar Visão Geral do Processo de Implementação	O instrutor descreve o processo de implementação, incluindo ▪ A importância da qualidade da implementação ▪ A necessidade e o conteúdo do padrão de codificação ▪ A estratégia para tratar componentes de baixa qualidade
2	Planejar Implementação	O Gerente de Desenvolvimento lidera o trabalho de ▪ Definir e planejar as tarefas de implementação (SUMP, SUMQ)
3	Alocar Tarefas	O Líder de Equipe ajuda na alocação das tarefas entre os membros da equipe e ▪ Obtém comprometimento de quando eles irão completar estas tarefas
4	Elaborar Projeto Detalhado	Os engenheiros produzem o projeto detalhado ▪ Fazem uma revisão do projeto usando métodos de revisão minuciosa de projeto ▪ Preenchem os formulários LOGD e LOGT
5	Elaborar Plano de Testes Unitários	Os engenheiros produzem planos de teste unitários
6	Desenvolver Testes Unitários	Os engenheiros seguem o *script* UT para desenvolver os casos de teste unitários, os procedimentos de teste e os dados de teste
7	Realizar Inspeção do Projeto Detalhado	O Gerente de Qualidade/Processo lidera a equipe em uma inspeção do projeto detalhado de cada componente (*script* INS e formulários INS e LOGD)
8	Codificar componente	Os engenheiros produzem o código-fonte do componente ▪ Fazem uma revisão de código utilizando um *checklist* pessoal ▪ Compilam e consertam o código até que ele compile sem erros ▪ Preenchem os formulários LOGD e LOGT
9	Realizar Inspeção do Código	O Gerente de Qualidade/Processo lidera a equipe na inspeção de código de cada componente (*script* INS e formulários INS e LOGD)
10	Realizar Testes Unitários	Os engenheiros, seguindo o *script* UT, conduzem os testes unitários e preenchem os formulários LOGD e LOGT
11	Revisar a Qualidade do Componente	O Gerente de Qualidade/Processo revê os dados de cada componente para determinar se a qualidade do componente atinge os critérios estabelecidos pela equipe ▪ Se sim, o componente é aceito para teste de integração ▪ Se não, o Gerente de Qualidade/Processo recomenda que o produto seja re-inspecionado e retrabalhado ou seja descartado e desenvolvido novamente
12	Realizar *Release* do Componente	▪ Quando os componentes são implementados e inspecionados satisfatoriamente, os engenheiros fazem sua liberação para o Gerente de Suporte ▪ O Gerente de Suporte insere os componentes do sistema de gerência de configuração

[11] O PSP é estruturado em níveis, que indicam o aumento da complexidade do processo de desenvolvimento de cada indivíduo. Nos níveis 2.1 e 3, os dois mais altos do PSP, o desenvolvedor utiliza um padrão de codificação e modelos de projeto, planeja tarefas e cronograma, estima e mede tamanho do seu trabalho, realizar revisões de projeto e de código, reporta resultados de testes, desenvolve o sistema ciclicamente e apresenta sugestões de melhoria para o processo. Maiores detalhes sobre a estrutura do PSP podem ser encontrados em [45].

Critério de Saída	▪ Componentes finalizados, inspecionados e sob gerência de configuração ▪ Formulários INS preenchidos para as inspeções de projeto e de código ▪ Planos de teste unitários e material de suporte ▪ Formulários SUMP, SUMQ, SUMS, LOGD e LOGT atualizados ▪ Documentação do projeto atualizada

Tabela 7 - Script IMP1

Propósito		**Guiar os engenheiros no testes unitários de seus componentes**
Critério de Entrada		O componente foi desenvolvido, revisado, compilado e inspecionado
Geral		▪ Apesar de todos os defeitos já terem sido encontrados nas fases anteriores, os testes unitários procuram identificar os poucos defeitos que ainda restarem ▪ É necessário cuidado, pois defeitos não encontrados em testes unitários podem levar de 5 a 40 horas cada para serem encontrados em etapas posteriores de teste ▪ Quando os defeitos de testes unitários são menores que 5 por KLOC[12], em geral não são encontrados defeitos nos componentes em testes ou uso posteriores ▪ Testes unitários cuidadosos geralmente encontram todos os defeitos de programas desenvolvidos adequadamente ▪ É importante rever e re-testar cuidadosamente cada conserto de defeito encontrado ▪ É mais provável acontecer erros durante o conserto de um defeito do que em um novo código
Passo	**Atividades**	
1	Elaborar Plano de Testes Unitários	Desenvolver um plano de testes para cobrir todos os passos dos *scripts* de teste com um mínimo de duplicação
2	Desenvolver Testes Unitários	▪ Desenvolver os casos de teste necessários e seus respectivos dados ▪ Rever o material de testes para garantir que eles não são defeituosos, pois na fase de Teste, defeitos nos casos teste são mais difíceis de encontrar do que defeitos no produto
3	Realizar Testes Unitários	Executar os casos de teste e consertar todos os defeitos encontrados
4	Realizar Teste dos Cenários	Testar todos os cenários de uso
5	Realizar Testes de Lógica	▪ Testar todos as ramificações lógicas ▪ Testar todas as condições ▪ Verificar o passo a passo e a terminação de cada laço ▪ Verificar alocação, liberação de ponteiros e ponteiros nulos
6	Realizar Testes de Interface	Verificar o comportamento apropriado de cada interface. Testar se ▪ Existem retornos apropriados para todos os casos de entrada ▪ Os tipos dos retornos e parâmetros estão corretos ▪ Condições de erro da interface são tratadas como especificado
7	Realizar Testes de Erros	Teste todas as condições de erro do programa para ▪ Valores e tipos impróprios ▪ Limite de representação de valores númericos (*overflow* e *underflow*) ▪ Problemas com inteiros, números reais, etc...
8	Realizar Testes de Variáveis	Verificar cada valor de variável e parâmetro ▪ No valor nominal, máximo e mínimo ▪ Acima do valor máximo especificado ▪ Abaixo do valor mínimo especificado ▪ Para operações apropriadas com 0, nenhuma entrada, tipo incorreto de dados, etc...
9	Realizar Testes de Dispositivos	Verificar a operação apropriada dos dispositivos ▪ Impressoras, telas, dispositivos de entrada, sensores, etc... ▪ Operação normal e incorreta: falta de papel, dispositivo desligado, etc...

[12] KLOC: *Kilo Lines Of Code*, mil linhas de código.

10	Realizar Outros Testes	Verificar outras informações importantes do produto ▪ Tamanho, limite e sobrecarga de *buffers* ▪ Performance e tempo de resposta precisos ▪ Limites e taxas de dados ▪ Datas e cálculos com datas ▪ Segurança, compatibilidade, conversão, instalação, recuperação
Critério de Saída		▪ Um teste unitário finalizado, com todos os defeitos consertados ▪ Dados de teste unitários preenchidos nos formulários LOGD, LOGT e LOGTEST

Tabela 8 - Script UT

4.2.7 Testes de Sistema e de Integração

A fase de Testes de Sistema e de Integração do TSP envolve atividades de *build* e integração, de testes de sistema, e de documentação do sistema. O principal objetivo desta fase é assegurar a qualidade do produto e não consertar defeitos, pois, apesar de ainda serem feitas algumas correções, a maior parte dos defeitos deve ter sido encontrada **antes** desta fase. Humphrey mostra em [48] que, quando sistemas de baixa qualidade são submetidos a testes de sistema e integração, as atividades de testes são bastante custosas e o produto final geralmente continua sendo de baixa qualidade.

O primeiro passo da fase de Testes de Sistema e de Integração é produzir os procedimentos e testes a serem efetuados. Isto inclui procedimentos para verificar se o *build* do sistema foi feito com sucesso e todas as partes necessárias estão presentes, e para testar todas as interfaces em condições normais e de erro. Além disto, pode ser produzida uma infra-estrutura para a execução automatizada tanto dos testes de integração quanto de sistema. Para isto, são produzidos os procedimentos de cada teste e o resultado final esperado.

Após a preparação dos materiais de testes necessários, o próximo passo é produzir o *build* do sistema. Nele os componentes necessários para compor a versão do sistema são obtidos e montados para formar o produto. Caso algum componente requerido ainda não tenha sido produzido, pode ser necessária a produção de *stubs* ou componentes que simulem o comportamento necessário. Se forem encontrados defeitos durante a produção do *build* do sistema, deve-se decidir se esta atividade deve continuar ou se os componentes defeituosos devem ser devolvidos aos desenvolvedores, para que sejam reparados.

O passo seguinte é a integração. Nela, verifica-se se todos os componentes estão presentes no *build* produzido, ou seja, se o *build* está completo. Em seguida, são executados os testes de integração. Assim como no passo anterior, a cada defeito encontrado, deve-se ponderar se as atividades de testes devem ou não continuar. Se possível, as correções dos erros encontrados devem ser efetuadas em paralelo com as atividades de testes. Entretanto,

se algum erro grave for encontrado, é melhor interromper as atividades de testes e inspecionar novamente os componentes defeituosos.

Uma vez concluídos os testes de integração, passa-se à execução dos testes de sistema. Assim como nos testes de integração, os defeitos encontrados devem ser registrados e deve-se julgar se as atividades devem prosseguir ou não. Em caso de componentes com muitos defeitos, é mais aconselhável continuar os testes de sistema após os componentes terem sido novamente inspecionados e os erros encontrados corrigidos.

Ao final, são executados testes de regressão, para garantir que as funcionalidades já existentes não foram afetadas. Para isto, são repetidos alguns testes de sistema dos ciclos anteriores. Após a conclusão com sucesso dos testes de sistema e de regressão, o sistema está pronto para o próximo ciclo de desenvolvimento ou para ser entregue.

Após a finalização dos testes, é produzida a documentação do usuário. Para isto, primeiramente é definida a sua estrutura geral. Em seguida, cada membro da equipe de documentação fica responsável pela produção de uma parte da documentação. Estas partes serão unidas pela Gerente de Desenvolvimento para a produção do primeiro rascunho da documentação, que será revisto pela equipe e devidamente corrigido, produzindo a documentação final. A Tabela 9 apresenta o *script* TEST1, que guia as atividades da fase de Testes de Sistema e de Integração.

Propósito		**Guiar a equipe na integração e teste dos componentes do produto no ciclo atual de desenvolvimento**
Critério de Entrada		A equipe possui uma estratégia de desenvolvimento e um planejamento de atividades ▪ Especificações ERS e EPS completadas e inspecionadas ▪ Componentes implementados, inspecionados, testados unitariamente e sob gerência de configuração. ▪ Os estudantes leram o Capítulo 9 do livro-texto
Geral		Quando defeitos são encontrados em *build*, integração ou testes de sistema, o Gerente de Qualidade/Processo determina se os testes devem continuar. Cada defeito encontrado em integração ou testes de sistema deve ser registrado no formulário LOGD e revisto por toda a equipe, para determinar ▪ Onde pode haver defeitos similares no produto ▪ Como e quando encontrar e consertar estes defeitos ▪ Mudanças no processo para prevenir defeitos semelhantes no futuro
Passo	**Atividades**	
1	Apresentar Visão Geral do Processo de Testes	O instrutor descreve o processo de testes de integração e sistema, explicando ▪ A necessidade de se possuir componentes de qualidade antes dos testes ▪ A necessidade e conteúdo dos padrões de teste ▪ A estratégia para lidar com componentes de baixa qualidade
2	Desenvolver Testes	O Gerente de Desenvolvimento lidera o desenvolvimento de testes. O Líder de Equipe ajuda na alocação das tarefas de desenvolvimento e de execução dos testes entre os membros da equipe Os membros da equipe de teste executam suas atividades de desenvolvimento de testes

		▪ Definindo quaisquer procedimentos e processos de *build* necessários ▪ Desenvolvendo procedimentos e infra-estrutura para testes de integração ▪ Desenvolvendo procedimentos e infra-estrutura para testes de sistema ▪ Medindo o tamanho e o tempo de execução de cada teste ▪ Revendo o material de teste e corrigindo erros
3	Elaborar *Build*	A equipe faz o *build* do produto e verifica sua completude ▪ Verifica se todas as partes necessárias estão à disposição ▪ Constroem o produto e o fornecem para testes de integração ▪ O responsável pelo produto (desenvolvedor) registra todos os defeitos encontrados no formulário LOGD
4	Realizar Integração	O Gerente de Desenvolvimento lidera as atividades de integração ▪ Verificar a completude e testar o produto de forma integrada ▪ Registrar todas as atividades de teste no formulário LOGTEST ▪ O responsável pelo produto (desenvolvedor) registra todos os defeitos encontrados no formulário LOGD
5	Realizar Testes de Sistema	O Gerente de Desenvolvimento lidera as atividades dos testes de sistema ▪ Testar o sistema para condições normais e de estresse ▪ Testar o produto para instalação, conversão e recuperação ▪ Registrar todas as atividades de teste no formulário LOGTEST ▪ O responsável pelo produto (desenvolvedor) registra todos os defeitos encontrados no formulário LOGD
6	Elaborar Documentação	O Gerente de Desenvolvimento lidera a equipe em ▪ Produzir a estrutura e as tarefas da documentação do usuário final ▪ Alocar estas tarefas à equipe de documentação ▪ Revisar a estrutura com a equipe de testes, para verificar se está correta ▪ Elaborar o rascunho da documentação para o ciclo atual de desenvolvimento ▪ Revisar, corrigir e produzir a documentação final do usuário
Critério de Saída		▪ Um produto integrado e testado para o ciclo de desenvolvimento ▪ Formulários LOGD e LOGTEST preenchidos para todos os testes ▪ Documentação do usuário revisada e finalizada ▪ Dados de tempo, tamanho e defeitos registrados

Tabela 9 - Script TEST1

4.2.8 *Postmortem*

Um ciclo de desenvolvimento do TSP encerra-se com a fase de *Postmortem*. Nela é feita uma avaliação geral dos resultados obtidos. Para isso, é realizada uma reunião com toda a equipe, onde serão analisados os dados colhidos pela equipe durante o ciclo, o desempenho das pessoas nos papéis, e oportunidades de melhoria do processo. Em virtude desta avaliação, podem ser efetuadas modificações no processo, nas metas definidas pela equipe ou nos responsáveis por cada papel.

Ao final, é produzido um relatório descrevendo o que foi produzido no ciclo, o desempenho dos membros da equipe nos papéis e as lições aprendidas no ciclo. Estas atividades são guiadas pelo *script* PM1, apresentado na Tabela 10.

Propósito	▪ **Reunir, analisar e registrar os dados do projeto** ▪ **Avaliar a performance da equipe e de cada papel** ▪ **Identificar maneiras de melhorar o processo no próximo ciclo** ▪ **Produzir o relatório do ciclo atual**
Critério de Entrada	▪ Os engenheiros possuem um produto finalizado e testado ▪ Eles reuniram todos os dados e preencheram todos os formulários ▪ Os estudantes leram os capítulos 10, 16, 17 e 18 do livro-texto

Geral		O relatório do ciclo contém uma análise do projeto na visão de cada papel ▪ Performance geral da equipe: Líder de Equipe ▪ Performance de planejado x realizado: Gerente de Planejamento ▪ Projeto geral e padrões do produto: Gerente de Desenvolvimento ▪ Gerenciamento de Mudança e suporte ao projeto: Gerente de Suporte ▪ Qualidade do processo e do produto: Gerente de Qualidade/Processo Este relatório deve ▪ Usar dados do processo para embasar afirmações dos engenheiros ▪ Considerar atentamente o significado dos resultados produzidos ▪ Ser curto e conciso
Passo	**Atividades**	
1	Apresentar Visão Geral do Processo de *Postmortem*	O instrutor descreve o processo de *postmortem* ▪ A necessidade de dados do processo completos e precisos ▪ O conteúdo do relatório do ciclo ▪ O processo e os formulários da avaliação por pares
2	Revisar dos Dados do Processo	O Gerente de Qualidade/Processo lidera a equipe na análise dos dados do projeto e na identificação de problemas e áreas de melhoria ▪ Liderança, planejamento, processo, qualidade ou suporte ▪ Ações e responsabilidades sugeridas pela equipe ▪ Áreas para melhoria do instrutor ou da infra-estrutura Os engenheiros preparam e submetem PIPs[13] sobre estas sugestões de melhoria
3	Avaliar Desempenho dos Papéis	O Líder de Equipe lidera a equipe na avaliação da efetividade dos papéis, ações do instrutor e infra-estrutura de suporte ▪ Onde eles foram efetivos ▪ Onde eles podem ser melhorados
4	Preparar Relatório do Ciclo Atual	O Líder de Equipe lidera a equipe na definição da estrutura do relatório do ciclo atual ▪ Alocando as tarefas de produção do relatório entre os membros da equipe ▪ Obtendo comprometimento sobre a finalização de seções do relatório ▪ Consolidando, revisando e corrigindo o relatório completo
5	Preparar Avaliação dos Papéis	Cada engenheiro completa a avaliação da equipe e de cada papel usando o formulário PEER, em porcentagem ▪ A dificuldade e a contribuição de cada papel ▪ Os percentuais de dificuldade e da contribuição de cada papel devem totalizar 100% ▪ A efetividade de cada papel numa escala de 1 (inadequado) a 5 (superior)
Critério de Saída		▪ O ciclo de desenvolvimento produziu um produto de alta qualidade com toda a documentação requerida ▪ O produto finalizado está sob gerência de configuração ▪ Todos os dados do processo foram analisados e PIPs submetidos ▪ As avaliações por pares estão prontas e foram submetidas (PEER) ▪ O relatório do ciclo atual foi finalizado e submetido ▪ Formulários SUMP e SUMQ foram preenchidos para o sistema e para todos os seus componentes ▪ A documentação do projeto está atualizada

Tabela 10 - Script PM1

4.3 Papéis do *Team Software Process*

O TSP define um conjunto de papéis que devem ser assumidos pelos membros da equipe de desenvolvimento. Estes papéis definem perfis e delimitam responsabilidades sobre atividades específicas do processo. A definição de responsabilidades claras para cada

[13] *Process Improvement Proposal*, proposta de melhoria do processo.

um contribui bastante para a formação de uma verdadeira equipe, pois sem ela a equipe pode perder bastante tempo até entender o que precisa ser feito e decidir quem deve executar que tarefa [48].

Além das responsabilidades definidas nos papéis, todos no TSP são considerados desenvolvedores, ou seja, espera-se que todos os membros da equipe, além das atividades pertinentes ao seu papel, participem ativamente das atividades de desenvolvimento, tais como projeto e implementação do sistema. Além disto, é importante que os papéis estejam distribuídos entre vários membros, e não concentrados somente em uma ou duas pessoas.

Os papéis são divididos entre a equipe na fase de Lançamento, conforme descrito na seção 4.2.1. É importante que o papel seja escolhido e não atribuído por alguém, para que cada membro da equipe escolha o papel com o qual tenha mais afinidade. Ao final de um ciclo, o desempenho de cada papel é avaliado e papéis cujo desempenho não tenha sido satisfatório podem ser trocados entre membros da equipe para o próximo ciclo. A seguir, apresentamos os papéis do TSP, com seus objetivos, principais atividades e o perfil desejado para desempenhá-lo.

4.3.1 Líder de Equipe

O Líder de Equipe é responsável por coordenar a equipe do projeto, alocando as tarefas entre os membros, motivando todos a executarem suas atividades da melhor maneira possível e resolvendo conflitos e questões que venham a atrapalhar as atividades do projeto. O Líder de Equipe está voltado para a gestão da parte "humana" do projeto, tratando de questões como motivação e desempenho geral da equipe.

As principais atividades do Líder de Equipe são:

- Motivar a equipe a cumprir suas tarefas, seguindo o processo e cumprindo o cronograma definido;
- Coordenar os encontros semanais da equipe, para verificar o que foi completado, se todos os membros forneceram os dados necessários, riscos e questões do projeto, e identificar as tarefas para a semana seguinte;
- Reportar o status da equipe para o instrutor;
- Ajudar a equipe na alocação de tarefas e resolução de questões e conflitos;
- Atuar como facilitador em todos os encontros da equipe e manter a documentação do projeto;
- Participar da produção do relatório do ciclo de desenvolvimento;

- Participar como engenheiro no desenvolvimento do produto.

Para desempenhar estas atividades, o membro da equipe deve se sentir confortável em uma posição de liderança, sendo naturalmente capaz de coordenar pessoas na execução de um trabalho comum e aproveitar da melhor maneira as características de cada membro da equipe. Além disto, deve ser capaz de identificar questões importantes e tomar decisões objetivas, sabendo ouvir a opinião de todos os envolvidos. Outra característica importante é ser capaz de tomar decisões algumas vezes impopulares e de pressionar as pessoas a completarem tarefas difíceis ou urgentes.

4.3.2 Gerente de Desenvolvimento

O Gerente de Desenvolvimento é responsável por liderar a equipe nas atividades de desenvolvimento propriamente ditas, garantindo que o produto está sendo produzido de acordo com os critérios de qualidade e com os requisitos definidos. Ele atua como um líder técnico, liderando as atividades de desenvolvimento do produto e tendo uma visão geral do que está sendo desenvolvido por cada membro da equipe.

As principais atividades de responsabilidade do Gerente de Desenvolvimento são:

- Liderar a equipe na produção da estratégia de desenvolvimento e nas estimativas iniciais de tamanho e tempo do produto a ser desenvolvido;
- Liderar a equipe na produção da Especificação de Requisitos de Software, no projeto de alto nível e na produção da Especificação de Projeto de Software;
- Liderar a equipe na implementação, *build*, integração e testes do sistema, além da produção da documentação do usuário;
- Participar da produção do relatório do ciclo de desenvolvimento;
- Participar como engenheiro no desenvolvimento do produto.

Para desempenhar este papel, o membro da equipe deve gostar de construir coisas e ser capaz de liderar o projeto e o desenvolvimento de um software. Além disto, deve ter bons conhecimentos de análise e projeto e dos métodos necessários. Outra característica importante é saber escutar as idéias de outros membros da equipe, para que possa objetivamente comparar as soluções propostas por eles com as suas.

4.3.3 Gerente de Planejamento

O Gerente de Planejamento é responsável por coordenar a produção de planos completos e precisos, tanto para a equipe como um todo, quanto para cada membro dela.

Além disto, é responsável por acompanhar o progresso em relação aos planos definidos, reportando semanalmente o progresso da equipe.

As principais atividades do Gerente de Planejamento são:

- Liderar a equipe na produção de um planejamento para o próximo ciclo de desenvolvimento, incluindo tarefas a serem executadas e cronograma;
- Balancear a carga de trabalho entre a equipe;
- Acompanhar o progresso em relação aos planos e reportá-lo semanalmente;
- Participar da produção do relatório do ciclo de desenvolvimento;
- Participar como engenheiro no desenvolvimento do produto.

Para desempenhar este papel, o membro da equipe deve ser uma pessoa organizada, que se sente confortável em ter um plano definido para executar seu trabalho, e tende a planejar suas atividades sempre que possível. Estar interessado em dados do processo para determinar os resultados do projeto, como por exemplo "Estamos atrasados?" ou "Por que o planejamento falhou?", também é uma característica importante. Por último, deve achar que planejamento é importante e estar disposto a pressionar os outros membros da equipe para que acompanhem e colham medições sobre suas atividades.

4.3.4 Gerente de Qualidade/Processo

O Gerente de Qualidade/Processo é o responsável pela definição dos padrões de qualidade adotados no projeto e pela elaboração do Plano de Qualidade. Além disto, ele deve garantir que a equipe está seguindo os padrões definidos e cumprindo as metas de qualidade que foram traçadas. Isto é feito coletando-se dados do processo e informando ao líder da equipe quaisquer problemas encontrados, e liderando atividades como inspeções e revisão de qualidade dos componentes.

As principais atividades do Gerente de Qualidade/Processo são:

- Liderar a equipe na produção e acompanhamento do plano de qualidade, bem como na definição e manutenção dos padrões de desenvolvimentos da equipe;
- Alertar a equipe e o Líder de Equipe quando problemas de qualidade forem detectados;
- Liderar a equipe na definição e documentação do processo e cuidar da sua melhoria contínua;

- Revisar e aprovar os produtos antes de sua submissão ao Comitê de Controle de Configuração;
- Atuar como moderador das inspeções e elaborar a ata das reuniões da equipe;
- Participar da produção do relatório do ciclo de desenvolvimento;
- Participar como engenheiro no desenvolvimento do produto.

O membro da equipe que desejar desempenhar este papel deve ter interesse em qualidade de software, processos e medições de processos. É preciso ter em mente que a qualidade do produto não é conseguida somente na fase de testes, mas desde o início do processo. Experiência com métodos de revisão e inspeção, e habilidade em conduzir revisões construtivas também são características desejáveis, apesar de não serem essenciais.

4.3.5 Gerente de Suporte

O Gerente de Suporte atua na definição e manutenção das ferramentas e métodos para suportar a equipe no desenvolvimento do produto. São de sua responsabilidade a gerência de configuração do projeto, o cumprimento das metas de reuso de componentes definidas e o estabelecimento de um sistema de acompanhamento de pendências, onde devem ser registrados os riscos e questões do projeto.

As principais atividades do Gerente de Suporte são:

- Liderar a equipe na definição das necessidades de suporte e na obtenção das ferramentas e infra-estrutura necessárias;
- Liderar o Comitê de Controle de Configuração e gerenciar os sistemas de controle de mudanças e de gerência de configuração;
- Manter o glossário do sistema e o sistema de acompanhamento de pendências;
- Atuar como um defensor do reuso na equipe;
- Participar da produção do relatório do ciclo de desenvolvimento;
- Participar como engenheiro no desenvolvimento do produto.

Para desempenhar estas atividades, o membro da equipe candidato a este papel deve gostar de ferramentas e métodos, e ser hábil no uso do computador, a ponto de poder prestar assistência à equipe em suas necessidades de suporte. Além disto, é importante que ele esteja ambientado com as ferramentas que provavelmente serão usadas no projeto.

4.4 Benefícios do *Team Software Process*

Existem vários relatos na literatura da Engenharia de Software sobre benefícios da adoção do TSP, como em [49] e [50]. Dentre estes benefícios, podemos destacar três principais: a formação de uma equipe coesa e integrada, a melhoria das práticas gerenciais, e o papel do TSP como facilitador da implantação do SW-CMM.

Na formação da equipe, a divisão das responsabilidades através da definição de papéis, assim como as reuniões semanais e a realização dos *postmortem* ao final de cada ciclo fazem com que o time trabalhe de forma alinhada com um objetivo comum e que a comunicação interna seja facilitada, como pode ser visto em [54]. Além disto, pelo fato do TSP ser focado desde o início na qualidade do produto final, através das inspeções e medições constantes, faz com que o produto chegue na fase de testes com poucos erros, o que diminui o custo desta fase em relação a processos que buscam encontrar os erros somente nos testes, diminuindo o custo total do projeto. Um exemplo disto é mostrado em [58].

O planejamento e acompanhamento das atividades é uma área muito forte do TSP. Além da participação de todos nas estimativas e planejamento das atividades, são coletadas várias métricas com o objetivo de se medir o progresso do projeto e melhorar as estimativas do ciclo seguinte. Além disto, a fase de Lançamento permite a definição de metas para o ciclo de desenvolvimento que se inicia, alinhando os objetivos de todos os membros da equipe. Estas práticas contribuem para a melhoria do gerenciamento do projeto e trazem tantos benefícios, que Oca relata em [54] o uso delas, com algumas adaptações, para a gestão de uma empresa de pequeno porte.

Todavia, o grande benefício do uso do TSP é, sem dúvida, seu papel como facilitador na implantação do SW-CMM [55]. Por ter sido concebido com o objetivo de definir como equipes de uma organização com grande maturidade deveriam trabalhar, o processo já atende a várias práticas do modelo SW-CMM.

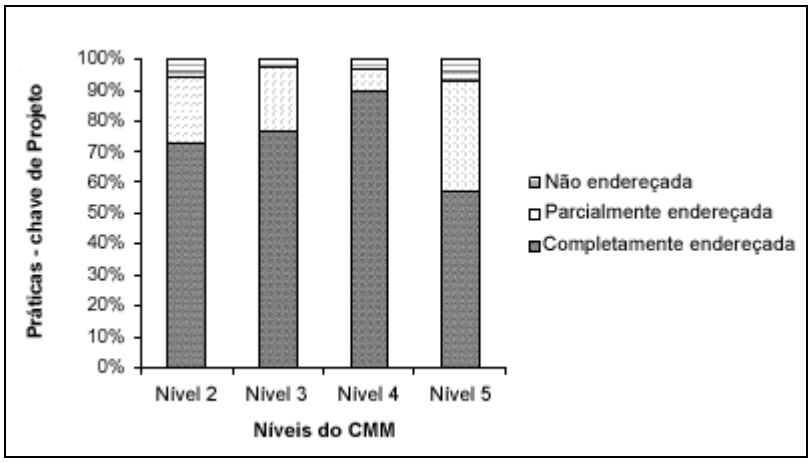


Figura 4-2 - Atendimento do TSP às práticas relacionadas aos projetos (traduzida de [56])

Davis mostra em [56] que o TSP atende 90% das práticas do SW-CMM que devem ser cumpridas nos projetos, em cada um dos cinco níveis de maturidade. Além disto, atende mais de 50% das práticas relacionadas à organização em cada um dos níveis, apesar de não ter sido concebido voltado para organizações, e sim para equipes. A Figura 4-2 e a Figura 4-3 apresentam, respectivamente, o quanto o TSP atende às práticas consideradas de projeto e da organização. Em seguida, a Tabela 11 apresenta, como exemplo, a maneira como o TSP atinge as metas das áreas-chave do nível 2 do SW-CMM. Em [56], é detalhado como o TSP atinge as demais práticas do nível 2 e as práticas dos demais níveis.

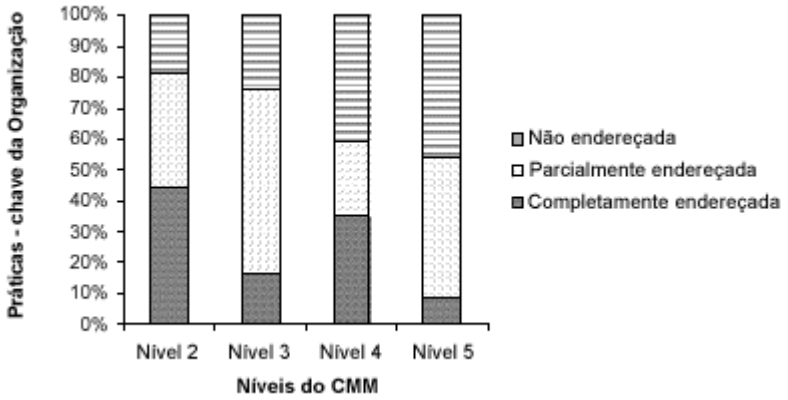


Figura 4-3 - Atendimento do TSP às práticas relacionadas à organização (traduzida de [56])

Este alinhamento com o SW-CMM apresenta duas grandes vantagens. A primeira é que o TSP serve como um guia de **como** as práticas podem ser implementadas, já que o SW-CMM só determina **o que** deve ser feito. A implementação das práticas é apontada por Davis como uma das grandes dificuldades das empresas que adotam SW-CMM [56].

A segunda vantagem é que a empresa atinge os níveis de maturidade em um espaço de tempo bem menor em relação ao usual. Atingir o nível 2, por exemplo, pode ser

até 40% mais rápido do que o normal, se o TSP for adotado como processo de desenvolvimento, como pode ser visto em [57]. Outro exemplo, é a redução em 33% do tempo necessário para passar do nível 3 para o nível 4, apresentada em [59].

Área-chave	**Meta**	**Como o TSP contempla**
Gestão de Requisitos	1. Os requisitos de sistema alocados ao software são Controlados para estabelecer uma linha mestra para o desenvolvimento de software e para o uso gerencial.	Uma equipe TSP define uma linha-mestra (*baseline)* para os requisitos. O TSP não inclui procedimentos para a gestão de requisitos. Esta meta é parcialmente coberta pelo TSP.
	2. Os planos, os produtos e as atividades de software são mantidos consistentes com os requisitos de sistema alocados ao software.	À medida que a equipe desenvolve os planos e produtos, ele são revisados em relação aos requisitos.Esta meta é parcialmente coberta pelo TSP.
Planejamento de Projeto de Software	1. As estimativas de software são documentadas para serem utilizadas no planejamento e no acompanhamento de projeto.	O TSP documenta e acompanha as estimativas de software utilizando um processo de estimativas definido. Esta meta é completamente coberta pelo TSP.
	2. As atividades e os compromissos do projeto de software são planejados e documentados.	Durante as fases de Lançamento, Estratégia e Planejamento, são planejados documentados as atividades e os compromissos do projeto. Esta meta é completamente coberta pelo TSP.
	3. Os grupos e as pessoas afetadas estão de acordo com os seus compromissos relacionados ao projeto de software.	Os gerentes de cada papel são responsáveis por tratar os compromissos com outros grupos ou indivíduos. A organização precisa determinar como grupos de engenharia de software participam de equipes de propostas, como incluem outros grupos no planejamento do projeto, e como acordam compromissos com grupos externos à organização. Esta meta é parcialmente coberta pelo TSP.
Acompanhamento e Supervisão de Projeto de Software	1. Os resultados e o desempenho reais são rastreados com relação aos planos de software.	Uma equipe TSP acompanha semanalmente os resultados reais e o desempenho em relação ao planejado. Membros da equipe acompanham o progresso em relação ao planejamento individual diariamente. Esta meta é completamente coberta pelo TSP.
	2. As ações corretivas são executadas e gerenciadas até sua conclusão, quando os resultados e o desempenho reais desviam significativamente dos planos de software.	Equipes TSP fazem pequenos ajustes nos planos, semanalmente, para minimizar o desvio entre os resultados atuais e o planejado. Quando ocorrem desvios significativos, um replanejamento completo é feito. Esta meta é completamente coberta pelo TSP.
	3. As alterações nos compromissos de software são consensadas pelos indivíduos e pelos grupos afetados	O responsável pelo papel do Líder de Equipe comunica mudanças nos compromissos aos membros da equipe, ao cliente, e ao demais envolvidos. Esta meta é completamente coberta pelo TSP.
Gestão de Subcontratação de Software	Por ser voltado para equipes, e não para organizações, o TSP não contempla esta área-chave.	
Garantia da Qualidade de Software	1. As atividades de Garantia da Qualidade de Software são planejadas.	A equipe cria um Plano de Qualidade de Software durante o Planejamento. Esta meta é completamente coberta pelo TSP.

	2. A aderência dos produtos de software e das atividades aos padrões, aos procedimentos e aos requisitos estabelecidos é verificada objetivamente.	Equipes TSP analisam dados do processo e do produto para verificar a qualidade de cada um deles. A aderência a padrões é verificada durante as inspeções. Esta meta é completamente coberta pelo TSP.
	3. As atividades e os resultados das atividades de Garantia da Qualidade de Software são informados às pessoas e aos grupos afetados.	E equipe planeja e acompanha coletivamente as atividades de garantia da qualidade. Ela coleta dados, analisa-os, e toma ações corretivas quando necessário. A equipe compartilha a situação das atividades com a gerência e com o cliente. Esta meta é completamente coberta pelo TSP.
	4. As questões de não conformidade, que não podem ser resolvidas internamente ao projeto, são levadas ao conhecimento da gerência superior.	As não conformidades são tratadas pelo papel do Gerente de Qualidade, em acordo com a equipe. Na pior das hipóteses, isto pode acontecer durante as reuniões semanais. Não existe uma política de escalonamento de questões não resolvidas no âmbito do projeto. Esta meta é parcialmente coberta pelo TSP.
Gestão de Configuração de Software	1. As atividades de gestão de configuração de software são planejadas.	A equipe planeja as atividades de gestão de configuração durante a fase Estratégia. Esta meta é completamente coberta pelo TSP.
	2. Os produtos de software selecionados são identificados, controlados e disponibilizados.	Os produtos selecionados são identificados durante a elaboração do plano de gerência de configuração, e são controlados durante as atividades de cada ciclo. Todavia, o TSP não inclui procedimentos para controle de mudanças. Esta meta é parcialmente coberta pelo TSP.
	3. As alterações realizadas nos produtos de software identificados são controladas.	O TSP requer que as mudanças nos produtos de trabalho de software identificados sejam controladas. Todavia, o TSP não inclui procedimentos para controle de mudanças. Esta meta é parcialmente coberta pelo TSP.
	4. As pessoas e os grupos afetados são informados sobre a situação e o conteúdo das configurações básicas (*baselines*) do software.	Esta meta não é coberta pelo TSP.

Tabela 11 - Mapeamento entre metas do nível 2 do SW-CMM e práticas do TSP

4.5 Considerações Finais

O TSP é um processo de engenharia de software que busca formar equipes efetivas de desenvolvimento, que trabalham de forma disciplinada, e preocupam-se com a qualidade desde o início do projeto. O desenvolvimento do produto é feito em ciclos, onde versões melhoradas do produto são implementadas. Cada ciclo é dividido em fases, cujas atividades são guiadas por *scripts*.

A adoção do TSP traz ganhos na qualidade do software produzido, no gerenciamento do projeto e, sobretudo, na formação de uma equipe motivada e que trabalha como um time de verdade. Além disto, o TSP está completamente alinhado com o SW-CMM, modelo de qualidade de reconhecida importância. O uso do TSP facilita atingir os níveis de maturidade definidos no modelo.

Entretanto, alguns obstáculos para a adoção do TSP ainda existem. Um deles é a necessidade de treinamento prévio em PSP, o qual é um pré-requisito para a implantação do TSP. Apesar do investimento neste treinamento ser recompensado em pouco tempo, pelos ganhos de produtividade e de qualidade, ainda existe uma resistência das empresas devido a questões culturais e pressões de mercado [50].

Outra desvantagem é a generalidade do processo. Nos *scripts* que guiam as atividades são apresentadas somente as etapas a serem cumpridas em cada fase, como elaborar especificação de requisitos, e não passos que guiem cada etapa específica, como no RUP. Além disto, apesar de o processo determinar a produção de uma extensiva documentação, não são fornecidos modelos destes documentos (*templates*), o que diminuiria bastante o esforço nesta atividade.

Capítulo 5: RUP-pe: Uma Metodologia Para Pequenas Equipes

5.1 Introdução

Nos capítulos anteriores, vimos que grupos de desenvolvimento com um número reduzido de integrantes necessitam de metodologias específicas, adaptadas às suas características, tais como comunicação facilitada e menor formalismo dos artefatos. Metodologias que não estejam adaptadas a estas características levam a execução das "atividades de *overhead*" do processo [51], assim chamadas por não resultarem em um progresso tangível para o projeto. Exemplos deste tipo de atividade são: preparação de planos e de documentação, avaliação da qualidade, testes, avaliação de riscos e acompanhamento das atividades.

Além disto, estar de acordo com um modelo de qualidade reconhecido é cada dia mais necessário, devido ao crescente aumento da competitividade e da exigência de qualidade do software desenvolvido. Caso se deseje obter o reconhecimento de um modelo de qualidade, como o SW-CMM, é necessário alinhar a metodologia em uso ao modelo desejado. Este alinhamento é, na maioria das vezes, bastante custoso, principalmente se a equipe utilizar uma metodologia com um maior grau de informalidade.

Neste capítulo apresentamos o RUP para Pequenas Equipes (RUP-pe), uma metodologia de desenvolvimento de software voltada para grupos de até 15 desenvolvedores e projetos de pequeno e médio porte, alinhada com o TSP. Seu propósito é ser uma metodologia de desenvolvimento simples e ágil, sendo um meio-termo entre as

Metodologias Ágeis e as metodologias pesadas, como o RUP. Além disto, por ser alinhado com o TSP, o RUP-pe está conseqüentemente alinhado com o SW-CMM, sendo uma metodologia que já possui uma garantia de qualidade do processo embutida em suas atividades.

O RUP-pe foi desenvolvido a partir de três elementos básicos:

- ***Rational Unified Process***: por ser um *framework* de processos bastante difundido e detalhado, o RUP foi escolhido para servir da base da metodologia. Suas atividades e papéis foram utilizados como ponto de partida para a proposta. Para isto, foi utilizada a versão 2002 do produto;
- **Metodologias Ágeis**: as práticas das Metodologias Ágeis têm se mostrado bastante efetivas na produção de software em equipes pequenas, levando ao desenvolvimento de software de qualidade e à satisfação tanto da equipe quanto do cliente [60] [61]. Estas práticas foram incorporadas de modo a tornar mais simples a execução das atividades;
- ***Team Software Process***: apesar de não entrar em detalhes sobre as atividades de desenvolvimento, o TSP fornece, em um nível de abstração mais elevado, um processo completamente alinhado com o SW-CMM. Possuir um processo como este é um diferencial importante, principalmente para empresas de pequeno porte, que precisam se sobressair no mercado. Por isso, o TSP foi utilizado como um guia para a definição das etapas do RUP-pe.

O RUP-pe possui o mesmo modelo de ciclo de vida do RUP, ou seja, é iterativo e incremental, e dividido em fases e disciplinas. Os objetivos e marcos de cada fase do RUP original foram mantidos. Por limitações de tempo, foram definidas apenas duas disciplinas: Gerenciamento de Projetos e Implementação. A escolha destas duas disciplinas deveu-se aos seguintes fatores:

- Importância para o sucesso do projeto: ambas as disciplinas são essenciais para o sucesso do projeto. A disciplina Implementação, por tratar do desenvolvimento propriamente dito do objetivo final do projeto: o software. A disciplina Gerenciamento de Projeto, pois estudos apontam que a falta de um planejamento de qualidade como causa de grande parte dos fracassos em projetos de software;
- Cobertura das disciplinas do RUP: ao definirmos uma disciplina de desenvolvimento e outra de suporte, mostramos como todas as atividades

do processo, estando ou não diretamente ligadas ao desenvolvimento do software, são afetadas pelo tamanho da equipe;

- Adoção de práticas "ágeis": as disciplinas escolhidas são as mais abordadas pelas metodologias ágeis. Assim, existe uma maior possibilidade de adoção de práticas destas metodologias.

Nas seções seguintes apresentaremos a definição das disciplinas Implementação e Gerenciamento de Projetos do RUP-pe, com seus respectivos fluxos, atividades, passos e artefatos. Esta definição foi feita da seguinte forma: primeiramente, o fluxo de atividades do RUP foi analisado criticamente, levando-se em consideração o cenário de pequenas equipes e projetos com as características apresentadas no Capítulo 1. Neste ponto foram identificados que atividades poderiam ser eliminadas ou modificadas, e quais não seriam afetadas pelo tamanho da equipe.

Em seguida, nas atividades que seriam modificadas, procurou-se incorporar práticas das Metodologias Ágeis, de modo a torná-las mais simples ou com menor grau de formalidade. Por fim, as atividades foram comparadas com o *script* correspondente do TSP, para que fosse verificado se o processo proposto estava de acordo com os passos do TSP e conseqüentemente, se o RUP-pe também estava alinhado com o SW-CMM.

Como resultado da definição destas disciplinas, foi elaborado um website [12] que apresenta o resultado obtido, proporcionando um guia para as equipes que desejarem utilizar a metodologia proposta. Lá estão detalhadas todas as atividades apresentadas neste capítulo, com seus respectivos passos, papéis e modelos de artefatos. Além disto, é apresentada uma visão geral das fases, do ciclo de vida da metodologia e dos papéis envolvidos nas atividades.

5.2 Disciplina de Implementação

Esta seção apresenta a adaptação feita na disciplina de Implementação do RUP. Esta adaptação tem por objetivos incluir práticas de XP às atividades de implementação e tornar o fluxo destas atividades o mais próximo possível do *script* IMP1 do TSP, utilizado para guiar as atividades de implementação neste último.

Em termos de correspondência, as atividades da disciplina de implementação do RUP estão distribuídas em duas etapas do TSP. As atividades de implementação dos componentes do RUP correspondem à fase de Implementação do TSP. Já as atividades de integração do RUP correspondem, no TSP, a algumas atividades da fase de Testes de Sistema e de Integração. Além disso, a fase de Implementação do TSP engloba algumas

atividades de Projeto, que pertencem à disciplina Análise e Projeto do RUP. Nesta adaptação, iremos tratar somente as atividades contempladas na disciplina Implementação do RUP, ou seja, relativas a implementação e integração.

A seguir, faremos uma análise crítica das atividades do RUP, em relação a pequenas equipes. Em seguida, apresentamos o novo fluxo de atividades proposto para o RUP-pe, com as atividades inseridas, e modificadas e suas respectivas justificativas. Ao final, é mostrada a correspondência do novo fluxo com o TSP e com XP.

5.2.1 Análise Crítica das Atividades do RUP

Estruturar o Modelo de Implementação

Esta atividade só é feita nas primeiras iterações da fase de elaboração. Consiste em definir de que forma os subsistemas e pacotes serão organizados, levando-se em consideração suas dependências e outras características, como o uso de componentes, e a atribuição da responsabilidade por componentes aos membros da equipe. A complexidade desta atividade está relacionada a fatores como o domínio do problema, a arquitetura escolhida e o tamanho do sistema, e não da equipe. Por isso, não será necessária nenhuma adaptação nela.

Implementar Componente

O objetivo desta atividade é produzir o código-fonte do sistema, de acordo com o Modelo de Projeto, produzido na disciplina Análise e Projeto. O tamanho da equipe não tem impacto na execução desta atividade.

Entretanto, serão incorporadas práticas de implementação das Metodologias Ágeis, tais como Programação em Pares, Refatoramento do Código, Propriedade Coletiva do Código e *Test-first Development*. Estas práticas, não sugeridas pelo RUP, têm se mostrado importantes na produção de um código de qualidade e de fácil manutenção.

Consertar Defeito

O propósito desta atividade é corrigir erros encontrados em um código já existente. Esta atividade não é influenciada pelo tamanho da equipe, não sendo necessário alterar seus passos.

Uma prática de XP será incorporada: escrever um caso de teste que mostre o defeito. Nesta prática, a maneira sugerida de se isolar um defeito é escrever um caso de teste que o revele. Isto facilita tanto a identificação da causa do defeito, como permite verificar se ele não será inserido novamente, já que os novos casos de testes o detectarão.

Implementar Componentes e Subsistemas de Testes

Esta atividade tem por objetivo implementar classes necessárias à execução automatizada dos testes unitários. Os passos do RUP são vagos e não apresentam comentários que auxiliem sua execução. Como o desenvolvimento dos componentes de teste se dá da mesma forma que a dos componentes do sistema propriamente dito, esta atividade será eliminada e os componentes de teste serão produzidos através da atividade Implementar Componente.

Será incorporada a prática de XP denominada *test-first development*. A idéia básica é os testes devem ser elaborados antes da implementação. Assim, o desenvolvimento se dá da seguinte maneira: escreve-se um teste, codifica-se o suficiente para o teste ser executado com sucesso, escreve-se outro teste, codifica-se novamente, e continua-se neste ciclo até que o componente seja desenvolvido Entre os benefícios do *test-first development*, podemos citar [17]:

- Os testes unitários são realmente elaborados. Quando a elaboração dos testes unitários é deixada para depois, geralmente são elaborados com menos cuidado ou apenas para uma parte das funcionalidades;
- Satisfação do programador ao ver os testes executarem com sucesso, o que traz maior motivação;
- Esclarecimento entre interface e comportamento. Ao se escrever testes para as diversas situações, exercita-se a definição das interfaces das classes e a maneira como os métodos devem se comportar, o que leva a percepção de possíveis falhas;
- Verificação comprovada, uma vez que possuir testes para todos os componentes do sistema traz alguma confiança na corretude do código produzido;
- Confiança para efetuar alterações. Já que existirão vários testes para praticamente todo o código, é mais fácil identificar erros ao se alterar o código já testado, realizando-se testes de regressão.

Para a adoção da técnica de *test-first development*, é necessário que os testes sejam desenvolvidos antes da implementação do componente ou de parte dele. Assim, esta atividade deve ser executada antes da atividade "Implementar Componente".

É importante salientar que os testes devem ser desenvolvidos para todo o código que tenha algum comportamento mais complexo, ou "tudo o que possa vir a falhar", como

em XP. Métodos simples, como exibição de mensagens ou *set()* e *get()* tradicionais não precisam, a princípio, ser testados.

O desenvolvimento dos casos de testes antes da codificação também é aconselhado no TSP. Segundo Humprey [48], o desenvolvimento dos testes detecta, em geral, mais erros de projeto do que a inspeção formal do projeto ou a execução dos testes unitários. Por isso, é importante desenvolver os casos de testes antes da implementação.

Sugere-se também o uso de *frameworks* de automação de execução de testes para a elaboração dos testes unitários, de forma que seja necessário desenvolver somente os casos de testes, não sendo preciso desenvolver a infra-estrutura necessária para a execução e coleta de resultados destes testes. A família xUnit, por exemplo, possui versões para várias linguagens [69] e é bastante difundida na comunidade de XP [33].

Executar Testes Unitários

Nesta atividade, os testes unitários são executados e o resultado da execução é analisado. Como esta atividade não está relacionada com o tamanho da equipe, não teve seus passos modificados. Entretanto, alguns princípios de XP serão adotados na execução dos testes unitários. O primeiro deles é que os testes devem ser automatizados. Uma vez que haverá testes para "tudo o que possa vir a falhar", a execução manual dos testes pode representar um ponto de gargalo na produtividade da equipe. Por se tratar de uma atividade meramente mecânica, ela pode e deve ser automatizada. Uma maneira de fazer isto é utilizando-se os *frameworks* de testes citados na seção anterior.

O segundo princípio a ser adotado é o de que a execução dos testes unitários só deve ser dada por encerrada quando todos os casos de teste forem executados com sucesso. Isto é muito importante para garantir a confiança no código e diminuir os erros de integração.

Além disto, um bom conjunto de testes que estão executando com sucesso traz mais confiança quando é necessário alterar o código, seja para incluir uma nova funcionalidade ou consertar um erro. Este princípio está alinhando com o TSP, onde um dos critérios de saída do *script* UT, utilizado para execução dos testes unitários, pede que "Um teste unitário deve ser concluído com todos os defeitos consertados".

Planejar Integração do Sistema

O objetivo desta atividade é determinar como será feita a integração do sistema a partir dos seus subsistemas. O RUP, segundo os conceitos apresentados na disciplina Implementação, adota uma estratégia de integração denominada Integração Incremental. Nela, o código é escrito e testado em pequenas partes, as quais são combinadas para formar

o todo, adicionando-se uma parte de cada vez. O RUP aconselha que esta integração seja feita, no mínimo, a cada iteração. Os benefícios da integração incremental são:

- Facilidade de localizar erros: quando um erro ocorre, ele está no componente adicionado ou modificado, ou na sua interação com a parte do sistema já integrada, pois o que já foi integrado deve estar funcionando perfeitamente;
- Componentes são mais testados: a cada componente integrado, o resultado é testado, para garantir que as funcionalidades não foram afetadas. Isto faz com que os componentes sejam testados mais freqüentemente do que se a integração fosse feita de uma só vez;
- Existe algo executando logo: os desenvolvedores vêem logo o resultado do seu trabalho. Isto aumenta a motivação e o comprometimento da equipe e torna mais fácil a obtenção de *feedback* junto aos usuários.

No que diz respeito à integração, o TSP apresenta diversas estratégias com seus prós e contras, deixando a cargo da equipe escolher a que mais se adequa à sua realidade. Das estratégias apresentadas, a escolhida pelo RUP equivale à denominada *One-at-a-time*. A integração é feita somente quando a implementação daquele ciclo é concluída, o que equivale, no RUP, ao final da iteração.

Os benefícios da integração incremental podem ser ampliados caso a prática de XP denominada Integração Contínua seja adotada. Na Integração Contínua, o sistema inteiro é integrado e testado várias vezes ao dia, sempre que uma nova parte do código é completada, seguindo o princípio da integração incremental. Isto aumenta os benefícios já citados, pois os erros são encontrados ainda mais cedo, os componentes são ainda mais testados e existe uma versão executável do sistema com uma freqüência maior [62].

Para o sucesso desta estratégia, é essencial dispor de ferramentas que automatizem o processo de *build* do sistema. Existem ferramentas grátis e de código aberto como o GNU make [63], o Apache Ant [64] e o Cruise Control [65], já bastante difundidas. Além disso, é extremamente aconselhável possuir um bom conjunto de testes de regressão automatizados, que possa garantir que o novo código integrado não afetou o funcionamento do sistema.

Com a adoção da Integração Contínua, os passos originais desta atividade devem ser reformulados, para refletir a prática em questão. Os novos passos serão descritos na seção 5.2.2.

Planejar Integração dos Subsistemas

Nesta atividade, é definido o modo como a integração dos componentes, a partir dos subsistemas, será feita, bem como a ordem em que estes componentes têm que ser integrados. Com a adoção da Integração Contínua, onde cada novo componente já é integrado diretamente ao sistema como um todo, esta atividade não é mais necessária. Por isso, ela será eliminada.

Integrar Subsistemas

O objetivo desta atividade é integrar os componentes para formar subsistemas, os quais serão integrados para formar o sistema final. Assim como a atividade anterior, com a adoção da Integração Contínua, esta atividade será eliminada, pois não é mais necessária.

Integrar Sistema

Na atividade Integrar Sistema, os subsistemas são integrados para formar o sistema final. O sistema será testado e, uma vez aprovado, incluído na linha-base do projeto. Com a adoção da estratégia da Integração Contínua, os passos desta atividade devem ser adaptados. Os responsáveis pela integração, por exemplo, devem ser os desenvolvedores que produziram o novo código a ser integrado, pois são as pessoas mais indicadas para resolver os conflitos que eventualmente surgirão. Os novos passos desta atividade serão detalhados na seção 5.2.2.

Revisar Código

A revisão de código tem por objetivos verificar a aderência aos padrões de codificação definidos pela equipe, disseminar conhecimento e boas práticas entre os membros do time, e identificar oportunidades de melhoria no código.

Com a adoção da Programação em Pares, esta atividade não é mais necessária, pois todos os objetivos citados são cobertos por esta prática. Williams mostra em [66] que a programação em pares promove uma revisão contínua do código, fazendo com que vários erros sejam encontrados mais cedo, disseminando conhecimento, promovendo um código melhor estruturado, entre outros. De acordo com um experimento realizado por Tomayko [67], com equipes utilizando XP e TSP, a programação em pares é mais eficiente e menos custosa do que as inspeções de código tradicionais.

5.2.2 A Disciplina Implementação no RUP-pe

Esta seção apresenta a disciplina Implementação proposta para o RUP-pe. A Figura 5-1 mostra o fluxo das atividades e, em seguida, cada uma delas é detalhada. As atividades originais do RUP que não foram adaptadas não serão comentadas em detalhes.

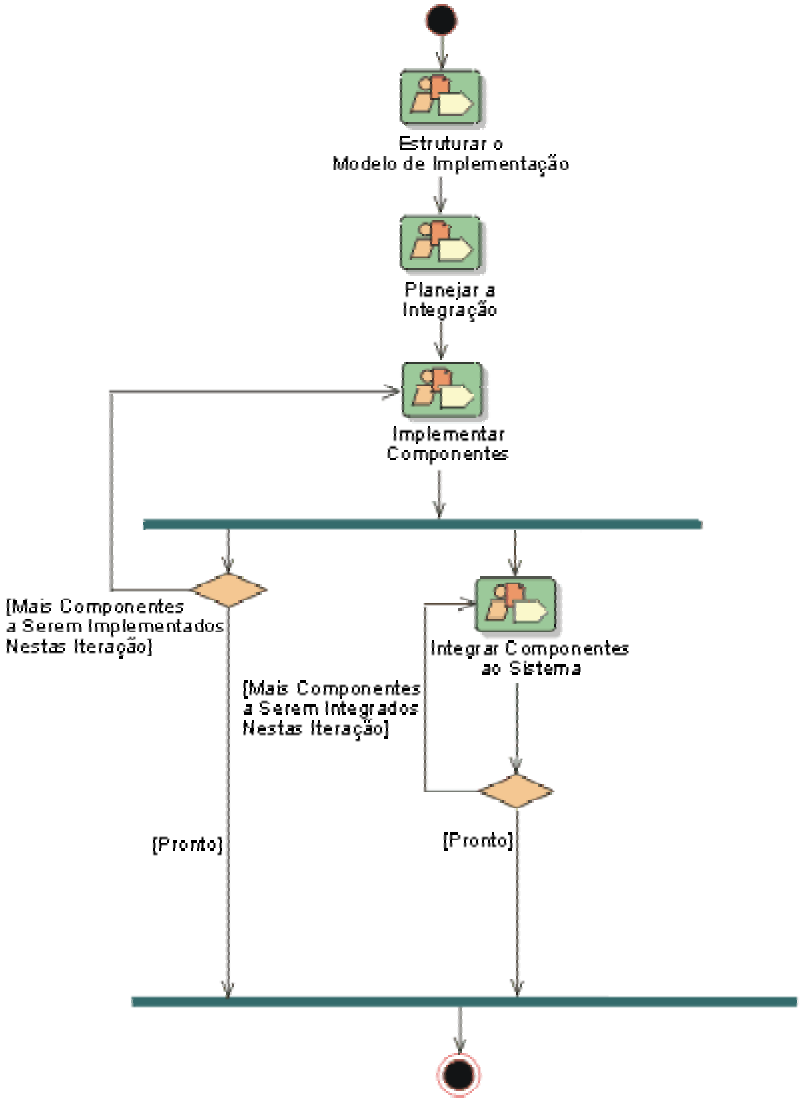


Figura 5-1 - Fluxo de Atividades da Disciplina Implementação do RUP-pe

Detalhes do Fluxo: Estruturar Modelo de Implementação

Este detalhe do fluxo, ilustrado na Figura 5-2, contém somente a atividade Estruturar Modelo de Implementação, a qual não necessitou de adaptações.

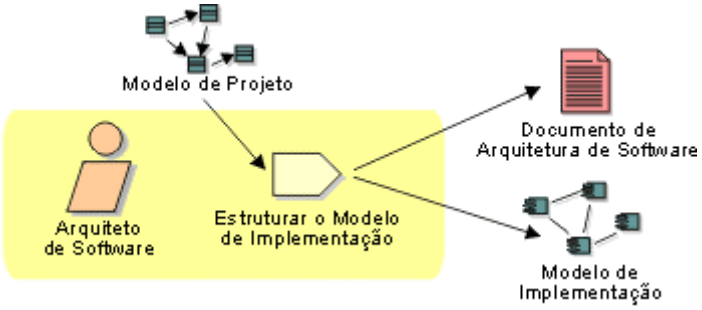


Figura 5-2 – Detalhe do Fluxo: Estruturar Modelo de Implementação

Detalhes do Fluxo: Planejar a Integração

Este detalhe do fluxo, ilustrado na Figura 5-3, contém somente a atividade Planejar Integração, que teve seus passos alterados, para a adoção da prática Integração Contínua.

Nesta atividade, devem ser definidos a freqüência dos *builds* e o conjunto de testes de regressão que serão executados. Quanto mais freqüentes forem estas duas atividades, melhores os resultados. Entretanto, para sistemas muito grandes, o processo de *build* pode ser muito demorado, sendo necessário diminuir a freqüência em detrimento da produtividade da equipe. Os novos passos desta atividade, apresentados em detalhes no website, são:

- Identificar testes de regressão: definir que casos de testes serão executados a cada *build*, para garantir que as funcionalidades existentes não foram afetadas;
- Definir freqüência de *builds*: definir o intervalo de tempo em que os *builds* serão gerados, como por exemplo, após a implementação de um componente ou duas vezes ao dia;
- Avaliar o Plano de Integração: efetuar uma revisão do Plano de Integração com a equipe com o intuito de identificar falhas e problemas.

O Plano de Integração, resultado desta atividade, deve documentar, de maneira simples e objetiva, as decisões tomadas. Ele pode, por exemplo, ser disponibilizado na forma de uma página HTML que pode ser consultada pela equipe sempre que necessário ou até mesmo o *script* de execução de uma ferramenta de automação de *builds*, como o Ant [64].

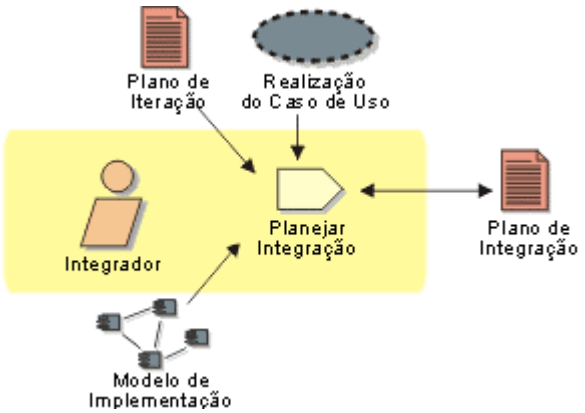


Figura 5-3 – Detalhes do Fluxo: Planejar Integração

Detalhes do Fluxo: Implementar Componentes

Este detalhe do fluxo, ilustrado na Figura 5-4, contém três atividades. Duas atividades originais do RUP foram removidas: Revisar Código e Planejar Integração dos Subsistemas.

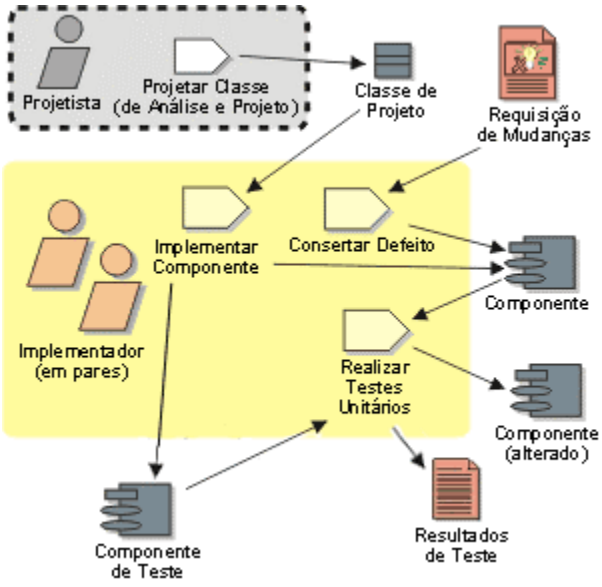


Figura 5-4 – Detalhes do Fluxo: Implementar Componentes

Para a atividade Implementar Componente, foram incorporadas quatro práticas de XP: programação em pares, *test-first development*, propriedade coletiva do código e refatoramento do código. Para a adoção da prática *test-first development*, será adicionado um novo passo à atividade, chamado Desenvolver Caso de Teste. Além disto, os seguintes pontos devem ser levados em consideração durante a implementação do componente:

- Todo o código deve ser produzido por duplas de programadores, como descrito na Seção 3.2.2;
- Este código deve seguir um padrão de codificação definido pela equipe e, sempre que houver oportunidades de melhorar o código, seja para simplificá-lo ou para deixá-lo mais estruturado e fácil de manter, deve-se aplicar regras de refatoramento, como as descritas em [68]. Entre os benefícios do refatoramento, podemos citar a melhoria do projeto do software, o fato de deixá-lo mais simples e fácil de entender e o aumento da facilidade de encontrar erros (decorrente da melhor estruturação do código) [68];
- As alterações não precisam ser feitas pela pessoa que desenvolveu inicialmente o componente, já que todos podem modificar qualquer parte do código sempre que for necessário, pois o código inteiro pertence a

todos. Por fim, todo o código produzido deve estar de acordo com os padrões definidos pela equipe. Isto facilita a comunicação entre a equipe e a manutenção futura do componente.

A atividade Implementar Componente deve ser usada tanto para os componentes do sistema quanto para componentes de teste, ou seja, os componentes que conterão a automação dos casos de testes unitários. Como já foi explicado, de acordo com a prática de *test-first development*, os componentes de teste devem ser desenvolvidos **antes** dos componentes do sistema.

A atividade Consertar Defeito não sofreu grandes alterações nos seus passos. A única modificação ocorreu no primeiro passo, Estabilizar o Defeito, onde se deve elaborar um caso de teste que revele o defeito.

A atividade Executar Testes unitários também não teve seus passos alterados. A única modificação foi o critério de conclusão dos testes, que é de 100% de sucesso, como em XP. Esta atividade deve ser executada alternadamente com a atividade Implementar Componente, já que a técnica de *test-first development* é adotada.

Detalhes do Fluxo: Integrar Componentes ao Sistema

Este detalhe do fluxo, apresentado na Figura 5-5, substitui o detalhe do fluxo Integrar Sistema. Assim como o original, ele contém apenas uma atividade, renomeada para Integrar Componentes ao Sistema.

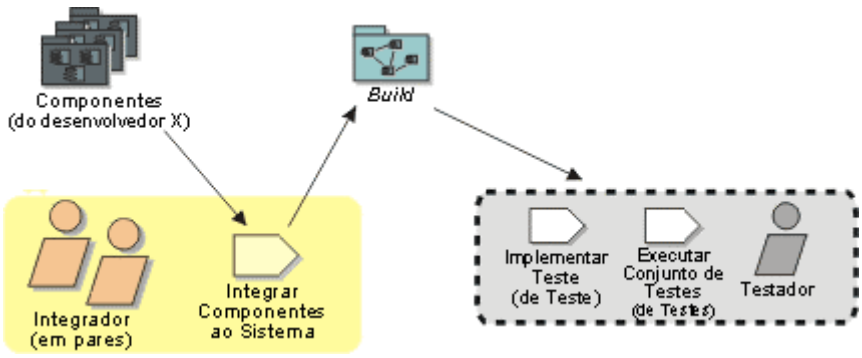


Figura 5-5 – Detalhes do Fluxo: Integrar Componentes ao Sistema

Esta atividade contempla a integração do componente recém-desenvolvido ao código já testado e integrado pela equipe. Conforme já descrito, isto deve ser feito logo que a implementação do componente for concluída e seus testes estiverem executando com sucesso. Preferencialmente, esta atividade deve ser automatizada.

A dupla que concluiu o componente é responsável por integrá-lo, devendo resolver quaisquer conflitos que ocorram durante a integração com o código já existente. Além disto, ela deve executar a bateria de testes de regressão para garantir que suas

alterações não alteraram a integridade do sistema. O novo código só deve ser oficialmente incluído no repositório do projeto quando todos os testes forem executados com sucesso. Para isso, propomos os seguintes passos para esta atividade:

- Obter o código já estável do repositório;
- Integrar o novo código e verificar erros de compilação e *build*;
- Uma vez gerado o *build* com sucesso, executar testes de regressão;
- Resolver erros dos testes. O código só pode ser atualizado quando todos os erros e conflitos forem resolvidos;

Esta atividade de integração corresponde, no TSP aos passos 11 e 12 do *script* IMP1. A única modificação é que o responsável por esses passos não é o Gerente de Qualidade/Processo, mas a dupla que desenvolveu o componente. Como a integração ocorre com freqüência, centralizar esta atividade em uma só pessoa pode comprometer o desempenho da equipe.

5.2.3 Correspondência com XP e TSP

A nova disciplina de Implementação aqui proposta contempla nove dos doze passos do *script* IMP1 do TSP, alguns com pequenas adaptações, e todos os passos do *script* TEST1 que dizem respeito à integração do sistema. Os passos não contemplados do *script* IMP1 são: Apresentar Visão Geral do Processo de Implementação, Elaborar Projeto Detalhado e Realizar Inspeção do Projeto Detalhado. O primeiro é mais voltado para o ensino do processo do que para as atividades de implementação propriamente ditas. Os dois últimos não correspondem a atividades da disciplina de Implementação do RUP, mas a atividades da disciplina de Análise e Projeto, estando fora do escopo desta proposta. Além disto, as atividades de integração são tratadas, no TSP, somente na fase de testes, mas foram incluídas por serem abordadas pela disciplina de implementação do RUP.

Adicionalmente, foram incorporadas cinco práticas de XP: programação em pares, integração contínua, refatoramento do código, testes unitários automatizados e propriedade coletiva do código. Esta correspondência é apresentada na Tabela 12.

Atividade do RUP-pe	TSP – passo/script	Prática de XP	Observação
Estruturar Modelo de Implementação	-	-	-
Planejar Integração	2, 3 e 4 (TEST1)	Integração contínua	-
Implementar Componente	5, 6, 8 e 9 (IMP1)	Testes unitários (*Test-first programming*), programação em pares, refatoramento do código, propriedade coletiva do código, padrão de codificação	Desenvolver os testes antes do código também é sugerido pelo TSP.
Consertar Defeito	8 e 9 (IMP1)	Programação em pares, refatoramento do código,	-

		propriedade coletiva do código, padrão de codificação	
Realizar Testes Unitários	10 (IMP1)	Testes unitários	-
Integrar Componentes ao Sistema	11 e 12 (IMP1)	Integração contínua	-

Tabela 12 - Correspondência entre a nova disciplina de Implementação, XP e TSP

5.3 Disciplina de Gerenciamento de Projeto

Esta seção apresenta a adaptação feita na disciplina de Gerenciamento de Projeto do RUP. Esta adaptação tem por objetivos incluir práticas de XP às atividades de Gerenciamento de Projeto e tornar o fluxo destas atividades o mais próximo possível dos *scripts* LAU1, STRAT1, PLAN1, PM1, e WEEK, utilizados, no TSP, para guiar as atividades de planejamento e acompanhamento do projeto.

A disciplina de Gerenciamento de Projeto do RUP está relacionada a quatro etapas do TSP: Lançamento, Estratégia, Planejamento e *Post-mortem*. As atividades de planejamento inicial correspondem às fases de Lançamento e Estratégia. Algumas destas atividades são revisitadas ao final das iterações, como a análise de riscos. O planejamento das iterações corresponde à fase de Planejamento, enquanto as avaliações de iterações e fases correspondem à fase de *Post-mortem*. O acompanhamento das atividades pode ser feito seguindo-se o script WEEK, que guia a realização da reunião semanal de acompanhamento do TSP.

A seguir apresentamos o novo fluxo de atividades da disciplina, com as atividades inseridas, e modificadas e suas respectivas justificativas. Ao final, apresentamos um resumo da correspondência do novo fluxo em relação ao TSP e a XP.

5.3.1 Análise Crítica das Atividades do RUP

Esta seção apresenta uma análise crítica das atividades da disciplina Gerenciamento de Projetos do RUP, levando-se em consideração equipes que possuem até 15 membros. As atividades são apresentadas em sua ordem de execução mais comum, sugerida pelo RUP.

Desenvolver Caso de Negócio (Estudo de Viabilidade)

O objetivo desta atividade é determinar se o projeto é viável do ponto de vista econômico. Para isto, é desenvolvido o Caso de Negócio, um documento que contém o contexto do negócio, previsões de retorno de investimento e restrições do projeto, entre outras coisas. A complexidade deste artefato está diretamente ligada à criticidade, ao custo e à importância do projeto para a organização, e não ao tamanho da equipe. Por isso, esta atividade não será modificada.

Identificar e Avaliar Riscos

Como o próprio nome diz, o objetivo desta atividade é levantar e analisar os riscos do projeto, ou seja, os fatores que podem fazer com que o projeto não obtenha sucesso. Estes riscos são capturados no artefato "Lista de Riscos", com suas respectivas probabilidades de ocorrência e o impacto que podem causar.

Apesar do tamanho da equipe poder ser um fator de risco para o projeto, a execução dos passos desta atividade não é influenciada pelo tamanho da equipe, e sim pela importância do projeto do ponto de vista técnico e financeiro. Assim, não será necessária nenhuma alteração nos seus passos.

Realizar Revisão de Aprovação do Projeto

Nesta atividade, a alta gerência da organização deve decidir se é viável ou não levar à frente o projeto em questão. Esta atividade não será modificada, pois não possui relação com o tamanho da equipe, e sim com a importância do projeto.

É recomendável realizá-la mesmo para projetos pequenos, para garantir que os níveis gerenciais superiores, a área de negócios da empresa e o cliente possuem um entendimento comum sobre a viabilidade do projeto. Pela sua importância, esta revisão deve produzir um Registro de Revisão.

Iniciar Projeto

O objetivo desta atividade é definir a equipe responsável pelo planejamento e gerenciamento do projeto, como por exemplo, o Gerente de Projeto, a Autoridade de Revisão do Projeto (ARP), figura da organização hierarquicamente superior ao gerente de projeto e o critério que será usado para julgar o sucesso do projeto.

Esta atividade não será modificada porque o modo de executá-la depende do tamanho e da cultura da organização, e não do tamanho da equipe do projeto. Em organizações de grande porte, por exemplo, pode-se ter um de projeto com uma equipe muito pequena, mas cuja aprovação e iniciação do projeto seja bastante burocrática devido a normas da empresa.

O fato de a equipe ser pequena pode apenas simplificar alguns passos, como "Designar equipe de planejamento do projeto", pois provavelmente poucas pessoas estarão envolvidas.

Desenvolver Plano de Garantia da Qualidade

Nesta atividade, é elaborado o plano que define os objetivos de qualidade e as atividades de Garantia da Qualidade do projeto. Esta atividade não é impactada pelo

tamanho da equipe, por isso não terá seus passos alterados. Entretanto, o responsável pela sua coordenação deve ser o Gerente de Qualidade e Processo, e não o Gerente de Projeto.

Desenvolver Plano de Medições

As medições são de extrema importância para o acompanhamento do progresso e da qualidade do projeto. Nesta atividade, o gerente de projeto deve determinar que medições serão feitas, a partir de que fontes de dados e quem são os responsáveis por coletar e interpretar estes dados. Esta atividade não é influenciada pelo tamanho da equipe, por isso seus passos não serão modificados. Em geral, este plano será simples e deve ser uma seção do Plano de Desenvolvimento de Software.

Entretanto, é imprescindível que sejam definidas métricas para o projeto. Caso a empresa não possua um conjunto de métricas pré-definidas, pode-se usar a abordagem de XP ou o conjunto de métricas sugerido por Campêlo [70].

Em XP são medidas apenas quatro variáveis: escopo, recursos, tempo e qualidade [33]. Segundo Beck [8], são estas variáveis que podem ser ajustadas para definir o rumo de um projeto. Por elas serem fortemente acopladas, ou seja, uma mudança em uma delas afeta as outras três, a solução para controlar o andamento do projeto é medi-las, para torná-las visíveis tanto para a equipe quanto para o cliente, de modo que todos os envolvidos possam achar o melhor balanceamento entre elas.

Campêlo sugere outro conjunto de métricas para o gerenciamento de projetos. Alguns exemplos de métricas sugeridas são: distribuição do esforço de desenvolvimento por fase e por disciplina, dias de atraso em relação aos marcos planejados, número de erros encontrados após o *release* e número de casos de uso implementados *versus* planejados por iteração. Sua proposta tem como principal vantagem utilizar métricas já voltadas para a o Nível 2 de Maturidade do *Software Capability Maturity Model* (SW-CMM 2).

Definir Organização e Equipe do Projeto

Esta atividade consiste em definir a estrutura organizacional do projeto e o perfil da equipe necessária em cada fase do ciclo de vida, ou seja, Concepção, Elaboração, Construção e Transição. A atividade não será modificada, mas sua execução será simplificada pelo fato de estarmos lidando com equipes pequenas. Afinal, definir uma equipe de dez pessoas, por exemplo, é bem mais simples do que definir uma equipe de cinqüenta.

Desenvolver Plano de Gerenciamento de Riscos

O Plano de Gerenciamento de Riscos descreve como serão tratados os riscos durante o projeto. Isto inclui a definição dos critérios usados para avaliação dos riscos, da

responsabilidade no projeto e na organização por cada risco, das tarefas de gerenciamento de riscos, e dos recursos e ferramentas necessários para este gerenciamento, entre outros.

Para projetos com equipes de pequeno porte, o gerenciamento dos riscos pode ser feito com um grau de formalidade menor, já que a comunicação da equipe é constante e facilitada. Assim, para o gerenciamento de riscos, será adotado somente o artefato "Lista de Riscos", incluindo-se os responsáveis pelas ações de mitigação e de contingência de cada risco. Não será necessária a produção do artefato "Plano de Gerenciamento de Riscos" e, por isso, esta atividade pode ser removida.

Definir Processo de Monitoração e Controle

O processo de monitoração e controle do projeto é apresentado na Seção 5.2.2. A monitoração do projeto será feita através dos encontros diários, das avaliações de iteração e da divulgação de métricas.

Para esta atividade, devem ser definidas somente as variáveis que serão monitoradas e os valores-limite para estas variáveis, a partir dos quais ações corretivas têm que ser tomadas. São exemplos de variáveis: densidade de defeitos, planejado *versus* realizado, casos de uso já implementados *versus* total de casos de uso. Assim, os passos de identificação dos indicadores do projeto e de seus valores serão incorporados à atividade "Desenvolver Plano de Medições", e esta atividade será removida, já que os outros passos tratam da definição do processo de monitoração, que já está definido.

Planejar Fases e Iterações

Nesta atividade, é elaborado o planejamento inicial do projeto. Isto envolve definir o escopo e as estimativas iniciais de esforço e custo, elaborar o cronograma e o orçamento iniciais do projeto, e definir os marcos das fases, entre outras atividades.

Os passos desta atividade não possuem relação direta com o tamanho da equipe, por isso não serão modificados. A única alteração necessária é a inclusão de marcos intermediários, para validação do trabalho realizado junto ao cliente, no passo "Definir Marcos do Projeto".

Desenvolver Plano de Aceitação do Produto

O Plano de Aceitação do Produto não é influenciado pelo tamanho da equipe, e sim por questões de contrato, por leis às quais o produto esteja submetido, ou pela criticidade do produto. Caso a maneira como a aceitação do produto será feita não esteja descrita no contrato do projeto, este artefato deve ser uma seção do Plano de Desenvolvimento de Software. Caso seja muito complexo, pode ser um artefato à parte.

Desenvolver Plano de Resolução de Problemas

Esta atividade será eliminada. Como a equipe é pequena, não há a necessidade de uma formalização do processo de resolução de problemas. Os problemas, assim como os responsáveis pela sua resolução, serão identificados durante encontros diários, como em Scrum, e durante as avaliações de iteração. Maiores detalhes sobre a identificação de problemas e acompanhamento de atividades são apresentados na Seção 5.2.2.

Compilar Plano de Desenvolvimento de Software

Nesta atividade, o Gerente de Projeto coordena a produção do Plano de Desenvolvimento de Software e dos diversos documentos de planejamento associados a ele. Apesar de não necessitar de modificações, pois seus passos não são afetados diretamente pelo tamanho da equipe, esta atividade teve sua execução simplificada.

Isto se deve ao fato de um menor grau de formalidade exigido no planejamento, devido à facilidade de comunicação da equipe. Por causa disso, alguns artefatos foram eliminados como o Plano de Resolução de Problemas e o de Gerenciamento de Riscos, e outros foram incluídos como seções do Plano de Desenvolvimento de Software, como o Plano de Aceitação do Produto, o Plano de Garantia da Qualidade e o Plano de Medições. O menor número de artefatos no planejamento torna menos custosa tanto a coordenação de seus respectivos desenvolvimentos como a compilação destes documentos.

Realizar Revisão de Planejamento do Projeto

O objetivo desta atividade é garantir que todos os envolvidos no projeto aprovam o planejamento inicialmente elaborado. Podem participar desta reunião grupos externos que têm uma participação rápida no projeto, como *designers* gráficos ou administradores de banco de dados.

Esta atividade não será modificada, pois seus passos não são afetados diretamente pelo tamanho da equipe. Entretanto, o fato de a equipe envolvida ser pequena facilita a realização da revisão, pois seu número de participantes será menor. Por ser uma revisão importante, deve produzir um Registro de Revisão.

Desenvolver Plano de Iteração

O planejamento da iteração envolve escolher que casos de uso serão desenvolvidos, determinar as atividades necessárias para este desenvolvimento, elaborar o cronograma destas atividades e distribuí-las entre a equipe. Esta atividade irá incorporar vários conceitos do planejamento de XP e do TSP, de modo a garantir um planejamento simples, mas efetivo, e que seja considerado realista pela equipe.

Isto será feito incorporando-se a prática Jogo do Planejamento, de XP, com algumas adaptações, além de alguns princípios do TSP, como o envolvimento de toda a equipe. As principais mudanças dizem respeito à definição do escopo da iteração, e elaboração de estimativas e divisão de tarefas. Além disto, propomos o uso de iterações curtas (entre 2 e 4 semanas), de duração fixa.

Realizar Revisão do Planejamento da Iteração

Nesta atividade, o Revisor do Projeto aprova o planejamento elaborado para a iteração. No cenário que estamos considerando, não é necessária, pois toda a equipe estará envolvida no processo de planejamento, seja na definição das atividades e estimativas, seja na atribuição de responsabilidades. *A priori*, toda a equipe já está tem conhecimento do planejamento da iteração e está comprometida com ele. Além disto, como as iterações são curtas e, por isso, freqüentes, aprovar um novo planejamento a cada 2 semanas pode trazer um gargalo desnecessário para o projeto.

Adquirir Pessoal

O objetivo desta atividade é adquirir os recursos humanos necessários para desenvolver o projeto, que serão mapeados nos papéis do RUP, de acordo com suas habilidades. Em seguida, o pessoal contratado para o projeto é sub-dividido em times e recebem os treinamentos que porventura sejam necessários. Para equipes pequenas, não é necessário agrupar os recursos em times menores, uma vez que a equipe em si já é pequena. Dessa forma, este passo pode ser removido, simplificando a atividade.

Iniciar Iteração

A atividade "Iniciar Iteração" consiste em dividir as atividades estabelecidas no planejamento entre a equipe, e adquirir recursos que sejam necessários para estas atividades. Devido à maneira como propomos o planejamento, esta atividade ficará reduzida somente ao passo "Adquirir Recursos Não-Pessoais", pois os passos "Atribuir Pessoal aos Pacotes de Trabalho" e "Emitir Ordens de Trabalho" podem sem dispensados. O primeiro não é necessário porque as responsabilidades já foram definidas no planejamento da iteração, onde cada um procura escolher as atividades que mais gostaria de realizar. Por esse mesmo motivo, o segundo passo será incorporado ao passo "Definir Tarefas e Responsabilidades" da atividade "Planejar iteração" Pelo fato da atividade ficar reduzida ao passo "Adquirir Recursos Não-pessoais", sugere-se que este passo seja incluído na atividade "Adquirir Pessoal", cujo nome deve ser mudado para "Adquirir Recursos". Dessa forma, a atividade "Iniciar iteração" seria eliminada.

Avaliar Iteração

O objetivo desta atividade é determinar se a iteração cumpriu seus objetivos e capturar lições aprendidas e oportunidades de melhoria do processo. Esta atividade é extremamente importante e deve ser mantida. No TSP, esta atividade corresponde à fase *Postmortem*.

Realizar Revisão do Critério de Avaliação da Iteração

Esta atividade tem por objetivo aprovar, de preferência com a participação do cliente, o critério que será utilizado para determinar se uma iteração foi concluída com sucesso ou não.

Esta atividade será removida, pois o critério de sucesso de uma iteração será sempre produzir uma versão executável do software com todos os casos de uso planejados inicialmente, que passe com sucesso pelos testes especificados. Como as iterações possuem uma duração muito curta, definir e aprovar um critério a cada 2 semanas trará um formalismo desnecessário ao processo, o que pode comprometer a produtividade da equipe.

Realizar Revisão de Aceitação da Iteração

Nesta atividade, o resultado da iteração recém-concluída é apresentado ao cliente para que seja formalmente aprovado. Como as iterações serão de curta duração, estas validações constantes podem ser custosas para ambos os lados. Para evitar este problema, sugere-se que as versões do sistema sejam mostradas para o cliente nos marcos de *releases* combinados, a cada 2 ou 3 meses. Estes intervalos não devem ser muito longos, pois o envolvimento do cliente é fundamentais para o sucesso do projeto [2] [3].

Assim, o nome da atividade foi modificado para "Realizar revisão de marco intermediário", para refletir melhor seu objetivo. Os passos da atividade não precisam ser alterados.

Agendar e Atribuir Trabalho

O objetivo desta atividade é acomodar novas tarefas que apareçam durante uma iteração. Como as iterações têm uma duração curta, esta probabilidade não é muito alta. Todavia, caso isto ocorra, a nova tarefa será comunicada a todos e encaixada no planejamento durante os encontros diários. Devido a esta estratégia, alguns passos devem ser modificados.

Monitorar Status do Projeto

O objetivo desta atividade é capturar o status atual do projeto e compará-lo com os planos, para detectar possíveis desvios e problemas de planejamento. Isto será feito

através de encontros diários com toda a equipe, como em XP e Scrum, sendo necessário alterar os passos desta atividade para suportar esta prática.

Reportar Status

O objetivo desta atividade é reportar o status do projeto e questões não resolvidas no âmbito do projeto para a Autoridade de Revisão do Projeto (ARP), a figura da organização hierarquicamente superior ao gerente de projeto. Neste aspecto, o único passo desta atividade, "Preparar Avaliação do Status", não precisa de modificações.

Todavia, achamos interessante que a equipe também seja informada do andamento das atividades e dos problemas encontrados. Isto será feito de duas formas: nas reuniões diárias do grupo e através da divulgação freqüente (a cada iteração) das métricas do projeto para todos.

O RUP não prevê este tipo de divulgação em suas atividades. O TSP, por sua vez, parte do princípio que cada um estará acompanhando o andamento de suas próprias atividades, o que é interessante para o gerenciamento individual, mas não dá visibilidade sobre andamento geral do projeto a todos. A estratégia de divulgação das atividades aqui sugerida, que é a adotada em XP, proporciona uma melhor visibilidade do status do projeto e uma melhor comunicação entre a equipe.

Tratar Problemas e Exceções

O objetivo desta atividade é tomar as ações corretivas apropriadas quando um problema é percebido. Segundo o RUP, o Gerente de Projeto é o responsável por iniciar esta tarefa e tomar as ações corretivas necessárias.

Sugerimos uma abordagem diferente para a resolução dos problemas: uma vez identificados nos encontros diários, os problemas serão tratados por todos que puderem contribuir para solucioná-los, e não somente pelo gerente de projeto, como sugerido pelo RUP. O fato de a equipe ser pequena facilita o tratamento dos problemas, pois exige um nível menor de formalidade e de comunicação.

Realizar Revisão do Projeto com a Autoridade de Revisão do Projeto

Esta atividade consiste em revisar o andamento do projeto com a Autoridade de Revisão do Projeto (ARP). É nesta revisão que o resultado da atividade "Reportar Status" é apresentado.

A forma de fazer esta revisão não tem relação direta com o tamanho da equipe, pois depende da estrutura da organização e dos seus processos gerenciais. Portanto, esta atividade não será modificada.

Preparar para Encerramento de Fase

O objetivo desta atividade é fazer os preparativos necessários para a atividade "Realizar revisão de marco do ciclo de vida". Seus passos não são afetados diretamente pelo tamanho da equipe. Entretanto, os passos "Verificar o status dos artefatos necessários" e "Distribuir os artefatos para os *stakeholders*" têm sua execução simplificada. Isto se deve ao fato do menor número de artefatos produzidos, uma vez que a comunicação entre a equipe é facilitada e o nível de formalidade, em geral, é menor.

Realizar Revisão de Marco do Ciclo de Vida

O objetivo desta atividade é revisar o status do projeto ao final de uma fase, para determinar se ele deve prosseguir ou não para a próxima fase. Os passos desta atividade são influenciados por fatores de negócios, financeiros e gerenciais, não sendo influenciados pelo tamanho da equipe. Dessa forma, esta atividade não será modificada.

Preparar para Encerramento do Projeto

O objetivo desta atividade é fazer os preparativos necessários para a atividade "Realizar revisão de aceitação do projeto". Seus passos não são afetados diretamente pelo tamanho da equipe, por isso esta atividade não será modificada.

Realizar Revisão de Aceitação do Projeto

O objetivo desta atividade é obter a aceitação formal por parte do cliente dos produtos entregues a ele. Esta atividade não será modificada, pois seus passos não são afetados diretamente pelo tamanho da equipe. Mesmo que exista um representante do cliente trabalhando juntamente com a equipe durante o projeto, é importante a execução desta atividade, por resguardar a equipe/organização de futuras reclamações do cliente sobre o produto.

5.3.2 A Disciplina Gerenciamento de Projeto do RUP-pe

Esta seção apresenta a disciplina Gerenciamento de Projeto proposta para o RUP-pe. A Figura 5-6 mostra o fluxo das atividades e, em seguida, cada uma delas é detalhada. Este fluxo de atividades é o mesmo do RUP original. Não foi necessário alterar nenhum dos subfluxos que compõem este fluxo nem a ordem de execução destes. As atividades originais do RUP que não foram adaptadas não serão comentadas em detalhes.

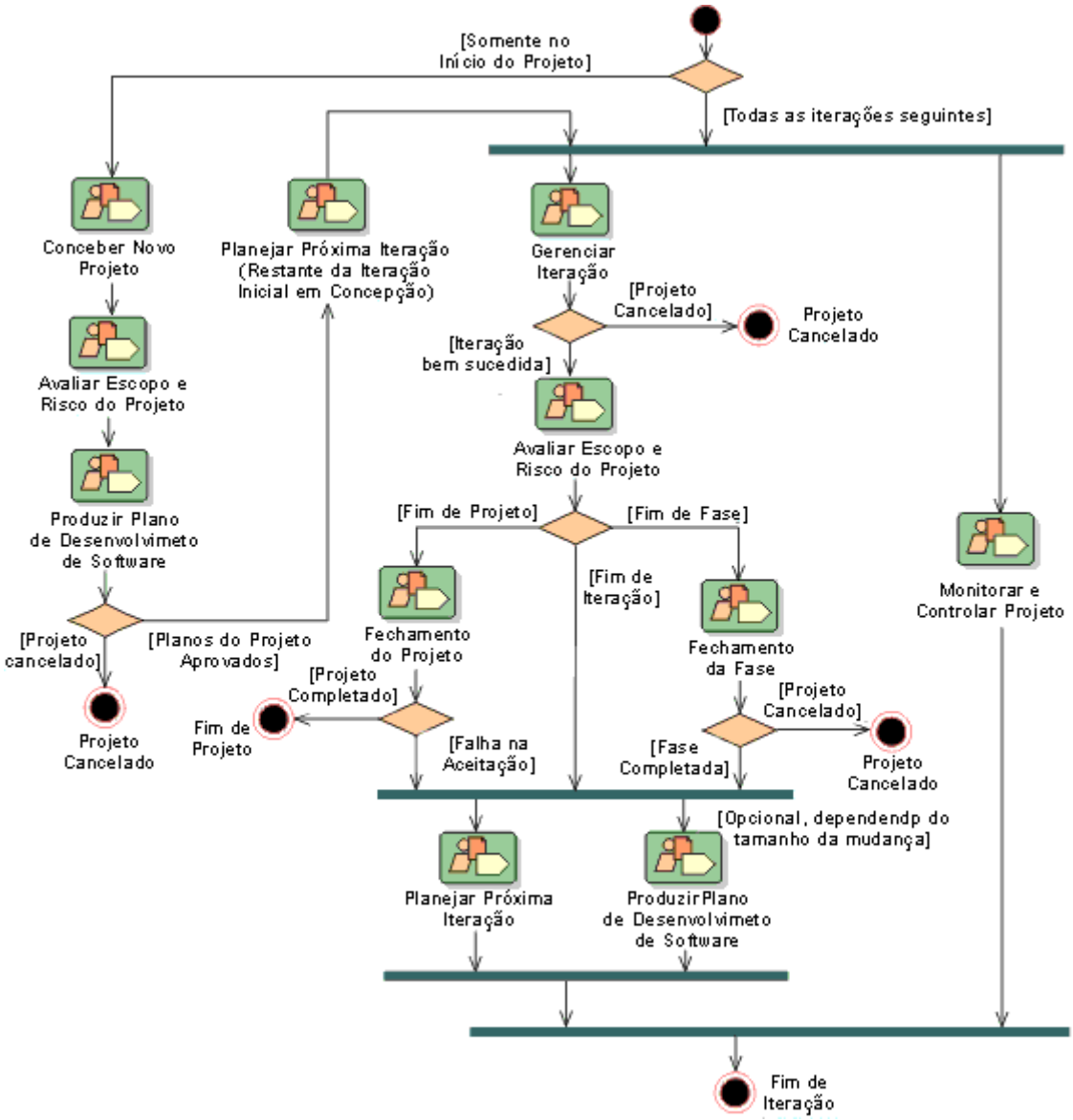


Figura 5-6 - Fluxo de Atividades da Disciplina Gerenciamento de Projeto do RUP-pe (traduzido de [16])

Detalhes do Fluxo: Conceber Novo Projeto

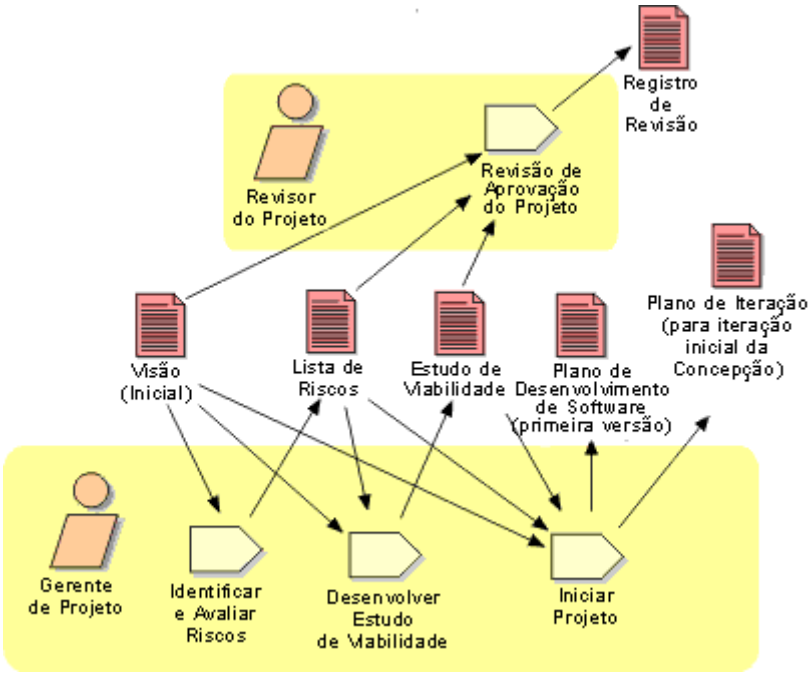


Figura 5-7 - Detalhes do Fluxo: Conceber Novo Projeto

Nenhuma das atividades deste subfluxo, apresentadas na Figura 5-7, necessitou de alterações. Entretanto, todas tiveram alterações nos artefatos de entrada e saída, devido à exclusão de algumas atividades originais do RUP.

Na atividade "Identificar e Avaliar Riscos", o artefato de entrada original, Plano de Gerenciamento de Riscos, foi substituído pelo artefato Lista de Riscos. As atividades Desenvolver Estudo de Viabilidade e Iniciar projeto tiveram o artefato Lista de Riscos incluído nos seus respectivos conjuntos de artefatos de entrada.

Na descrição das atividades originais do RUP "Desenvolver Estudo Viabilidade" e "Iniciar Projeto", não é citado nenhum artefato que contenha os riscos do projeto. Esta falha do RUP pode fazer com que estas atividades sejam executadas de forma errada pelos usuários do processo. Por exemplo, ao se fazer o estudo da viabilidade sem levar em conta os riscos ou ao se alocar de um gerente pouco experiente para um projeto de alta complexidade, pode-se causar o fracasso de um projeto já nos seus estágios iniciais.

Detalhes do Fluxo: Avaliar Escopo e Risco do Projeto

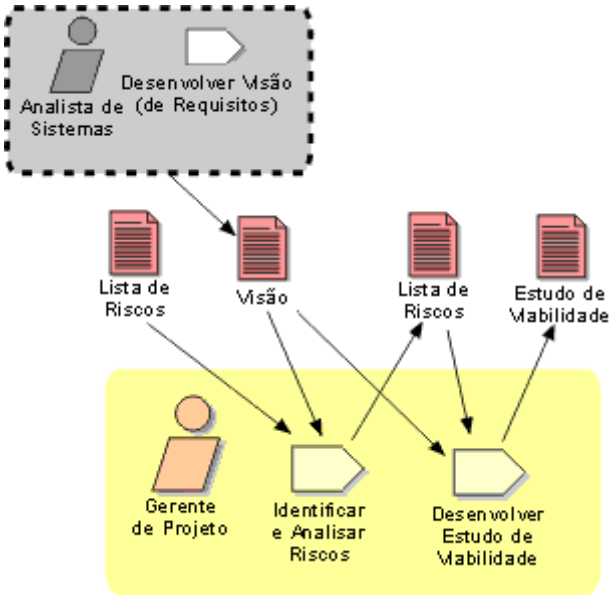


Figura 5-8 - Detalhes do Fluxo: Avaliar Escopo e Risco do Projeto

As duas atividades deste subfluxo, apresentadas na Figura 5-8, já foram comentadas na seção anterior.

Detalhes do Fluxo: Produzir Plano de Desenvolvimento de Software

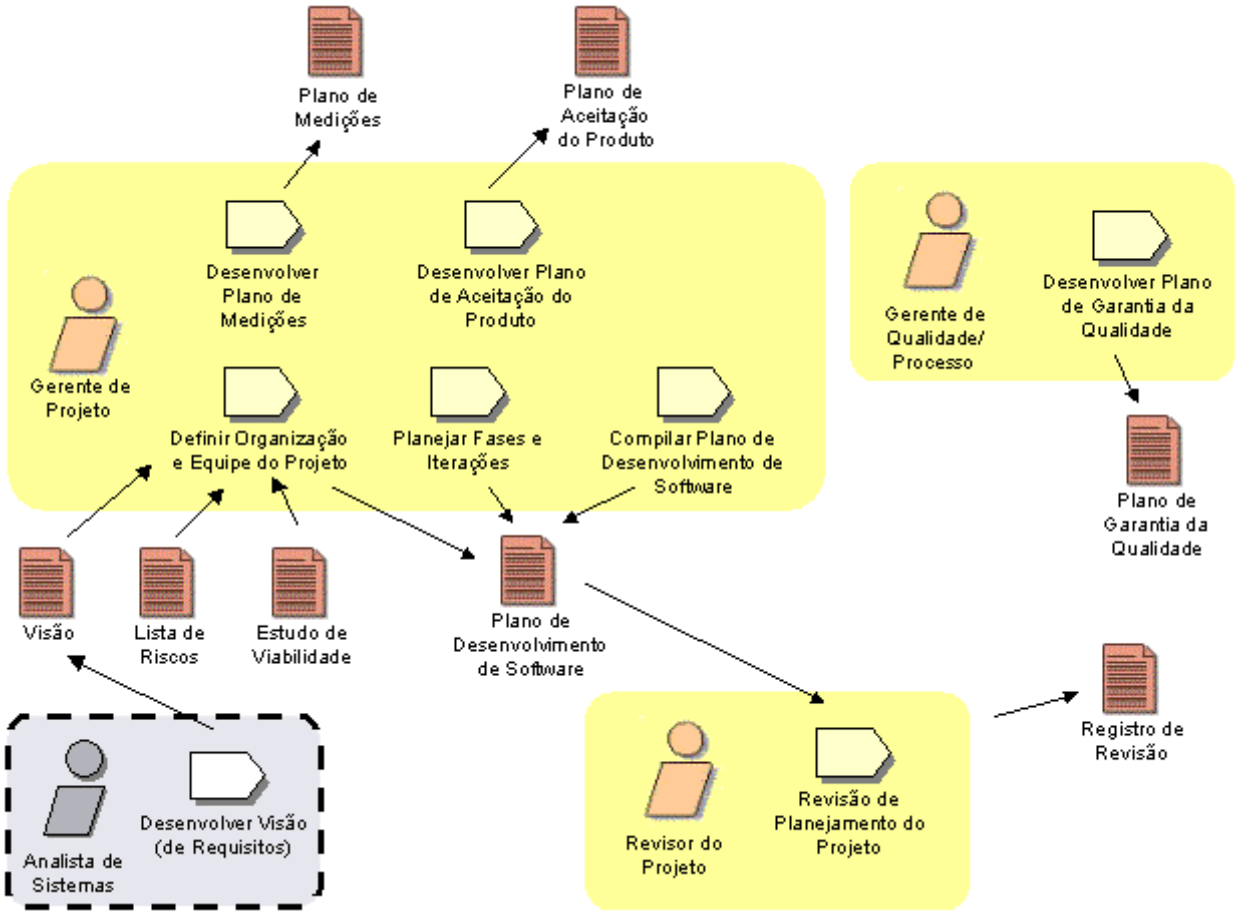


Figura 5-9 - Detalhes do Fluxo: Produzir Plano de Desenvolvimento de Software

As atividades deste subfluxo, apresentadas na Figura 5-9, sofreram algumas modificações. Na atividade Desenvolver Plano de Medições foi incluído o passo "Definir Indicadores do Projeto", vindo da atividade do RUP Definir Processos de Monitoração e Controle. Esta atividade, como vimos na Seção 5.3.1, foi removida.

A atividade Desenvolver Plano de Garantia da Qualidade deve ser executada pelo Gerente de Qualidade e Processo, e não pelo Gerente de Projeto, como no RUP. Nesta atividade estão descritas algumas das responsabilidades do Gerente de Qualidade e Processo.

Caso o Plano de Garantia da Qualidade seja simples, deve ser uma seção do Plano de Desenvolvimento de Software. Caso contrário, é interessante que seja um artefato em separado, de modo a deixar o Plano de Desenvolvimento de Software mais conciso.

Na atividade Planejar Fases e Iterações, apesar de não ter havido mudanças nos passos, deve-se ter em mente que o planejamento inicial do projeto não deve ser muito detalhado, pois ainda não se têm informações que permitam fazer um planejamento confiável. Além disto, pelo fato do desenvolvimento de software ser uma atividade bastante dinâmica, planos de longo prazo geralmente não são realistas. Boehm mostra em [71] que gastar um tempo excessivo no planejamento gera uma alta probabilidade de fracasso por causa do tempo gasto nessa atividade e porque mudanças tornarão os planos obsoletos, causando atrasos.

Devem ser definidos os marcos iniciais para cada fase do projeto, os marcos intermediários para validação do trabalho com o cliente, seus respectivos objetivos, e o número inicial de iterações. Como será adotada a estratégia de iterações de tamanho fixo, o número de iterações será derivado a partir da duração de cada fase. Os objetivos de cada iteração devem ser determinados em alto nível, pois o planejamento detalhado deve ser feito somente a cada iteração. Deve-se planejar detalhadamente, no máximo, para a iteração seguinte.

A atividade Compilar Plano de Desenvolvimento de Software foi simplificada, como já comentado na Seção 5.3.1. Ela teve alguns passos reformulados e artefatos de entrada removidos, pois alguns planos que formam o Plano de Desenvolvimento de Software, como o Plano de Resolução de Problemas foram eliminados.

Não foram modificadas as atividades Desenvolver Plano de Aceitação do Produto, Definir Organização e Equipe do Projeto e Revisão de Planejamento do Projeto.

Detalhes do Fluxo: Planejar Próxima Iteração

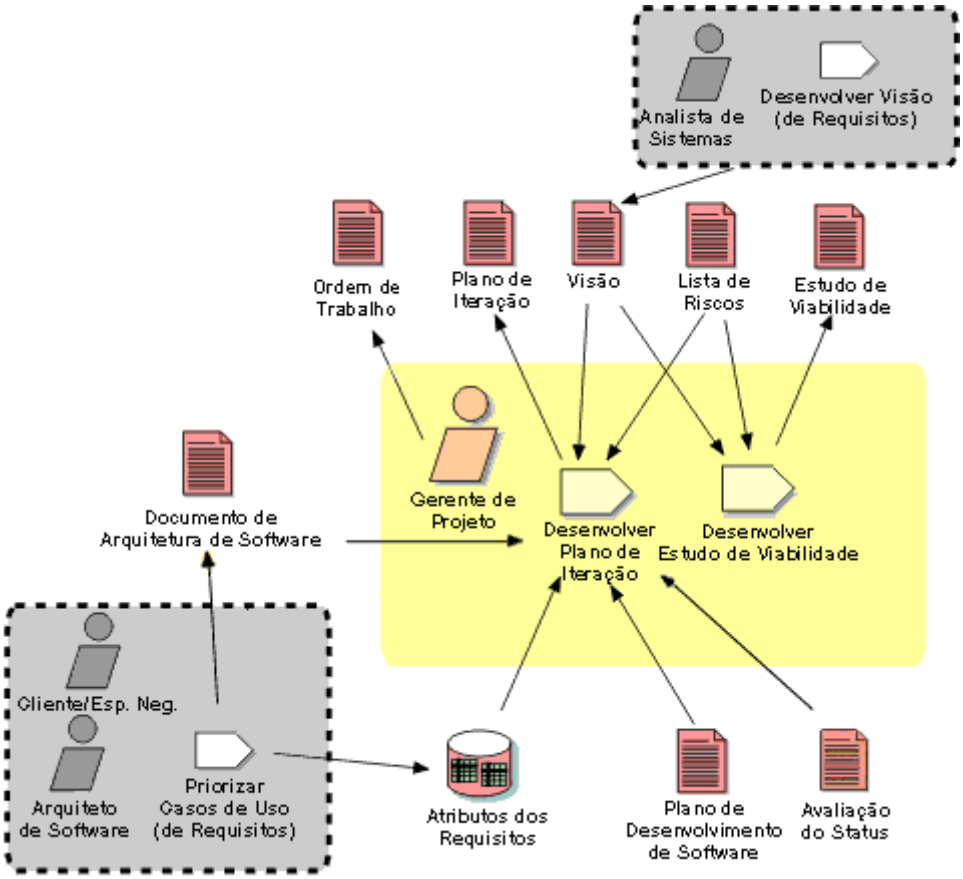


Figura 5-10 - Detalhes do Fluxo: Planejar Próxima Iteração

As atividades deste subfluxo são apresentadas na Figura 5-10. A atividade Desenvolver Estudo de Viabilidade já foi comentada anteriormente. Na atividade Desenvolver Plano de Iteração, foram incorporados vários conceitos do TSP e de XP para a realização das atividades de planejamento.

O primeiro passo do planejamento da iteração é determinar o seu escopo, ou seja, que casos de uso serão desenvolvidos. No RUP original, a escolha dos casos de uso é feita pelo Arquiteto de Software, o que pode fazer com que casos de uso importantes para o negócio, mas considerados de baixo risco do ponto de vista técnico, sejam deixados em segundo plano. Desta forma, as versões intermediárias do sistema podem não prover tanto valor ao cliente quanto poderiam.

Em XP, o escopo da iteração é determinado pelo cliente, que decide com base nas suas necessidades, o que deve ser implementado em primeiro lugar. Apesar da equipe poder alertar ao cliente sobre possíveis riscos técnicos em funcionalidades consideradas menos importantes para o negócio, a palavra final é do cliente. Isto pode fazer com que alguns riscos sejam atacados muito tardiamente.

A abordagem escolhida para o RUP-pe é a seguinte: a equipe deve priorizar os casos de uso em conjunto com um representante do cliente ou alguém que entenda bastante do negócio. Esta escolha deve ser feita após o consenso entre ambas as partes, baseando-se nos seguintes fatores:

- Riscos: os riscos devem ser atacados o quanto antes. É importante implementar casos de uso que estejam ligados a um ou mais riscos o mais cedo possível. Assim, caso seja percebida a inviabilidade do projeto, os recursos investidos terão sido menores do que se os riscos fossem deixados para o final. Esta é uma das grandes vantagens do desenvolvimento iterativo e incremental;
- Criticidade: além dos riscos, outro fator importante na escolha de um caso de uso é a importância dele para o projeto, ou seja, o valor que a sua implementação traz para o negócio do cliente. Este é o principal fator de escolha utilizado por XP. Isto, entretanto, possui o grande problema de deixar os riscos em segundo plano, o que pode trazer grandes dificuldades para o sucesso do projeto. A determinação da importância do caso de uso para o negócio deve ser feita, preferencialmente, pelo próprio cliente ou usuário do sistema. Além de ser a melhor pessoa para determinar o que é mais importante, este envolvimento faz com que o cliente se sinta "ouvido" pela equipe e note que o sucesso do projeto também depende dele. Caso este envolvimento não seja possível, a criticidade dos casos de uso deve ser determinada por alguém que entenda bastante do negócio.
- Cobertura: deve-se garantir que pelo menos um cenário de cada caso de uso já foi implementado. Isto é importante para exercitar todas as funcionalidades do sistema, o que pode levar à detecção de novos riscos ou à descoberta de fatores que não foram identificados durante a análise de requisitos.
- Desenvolvimento de habilidades: é importante preocupar-se em aprender o que não se conhece o mais cedo possível, como uma nova ferramenta ou testar uma nova tecnologia. Isto pode ser incluído no tópico "Riscos", mas é importante ter em mente que, mesmo que este fator não seja considerado um risco, deve ser desenvolvido o mais cedo possível.

Uma vez determinados os casos de uso candidatos para a iteração atual, deve-se verificar quantos poderão ser feitos no espaço de tempo de uma iteração. A abordagem

sugerida é utilizar iterações de tamanho fixo, também chamadas de iterações *time-boxed*. Este conceito foi recentemente incorporado ao RUP, em sua versão 2002, apesar de já ser uma prática comum em várias metodologias ágeis, como Scrum [9], FDD [36], XP [8], entre outras. Larman [17] aponta os seguintes benefícios desta estratégia:

- Lei de Parkinson: segundo Parkinson [72], o trabalho se expande para ocupar todo o tempo disponível para sua execução. Datas finais distantes ou não muito claras, como por exemplo, "daqui a 6 meses", pioram este efeito. Ter marcos constantes e bem definidos ajuda a equipe a manter o foco e progredir no trabalho;
- Priorização e decisão: iterações curtas e com data fixa para acabar forçam o time a tomar decisões mais objetivas na priorização do trabalho e dos riscos, escolhendo o que possui mais valor para o negócio e ajuda a mitigar os riscos;
- Satisfação da equipe: iterações curtas levam a uma sensação constante de trabalho cumprido. A equipe vê algo concluído em intervalos de tempo bem definidos, o que aumenta a sua satisfação e a sua confiança;
- Confiança do cliente: quando a equipe se compromete em entregar um conjunto de funcionalidades em um curto espaço de tempo e cumpre isto freqüentemente, o cliente passa a ter uma confiança cada vez maior no time.

Cada iteração deve durar entre 2 e 4 semanas. Iterações de duração curta permitem que o progresso seja acompanhado mais de perto e as estimativas refinadas mais rapidamente, já que estes dois fatores estarão sendo reavaliados em curtos períodos de tempo.

Para determinar o que poderá ser feito, é necessário saber as atividades necessárias para a implementação dos casos de uso e suas respectivas durações. Toda a equipe deve estar envolvida nisto, pois a participação das pessoas que efetivamente irão realizar as atividades torna mais remota a possibilidade de que uma atividade seja esquecida.

Após a descoberta das atividades necessárias, cada membro da equipe deve escolher as atividades que deseja realizar. Ao escolher uma atividade, o membro do time deve informar quanto tempo levará para realizá-la. O TSP sugere que atividades com duração maior do que 10 horas devem ser divididas em atividades menores, pois traz uma maior precisão às estimativas. Tarefas que serão realizadas em grupo devem ter um responsável, que se preocupará em garantir o seu cumprimento.

Deixar que todos os membros da equipe participem ativamente do planejamento e escolham as tarefas que desejam realizar faz com que todos criem um comprometimento com o plano e busquem cumpri-lo. Isto é apontado por Humprey [48] como um fator de extrema importância para o sucesso do planejamento.

No cenário inverso, onde as atividades e o cronograma são determinados somente pelo gerente de projeto e meramente repassados à equipe, é bastante provável que o esforço gasto no planejamento tenha sido em vão, pois a equipe não estará comprometida com o plano e dificilmente irá cumpri-lo.

Caso a duração total das tarefas seja maior que a duração da iteração, o escopo deve ser revisto e casos de uso menos importantes devem ser deixados para iterações posteriores. A redefinição do escopo da iteração, em detrimento do aumento de sua duração, é aconselhada por XP e pelo próprio RUP, no passo Determinar Atividades da iteração, da atividade Desenvolver Plano de Iteração.

Ao final, cada membro da equipe terá sua Ordem de Trabalho, que conterá as tarefas escolhidas e as respectivas estimativas feitas durante o planejamento. Isto permite que o membro da equipe acompanhe seu próprio progresso.

Caso seja necessário, o membro da equipe pode adicionar informações complementares sobre suas tarefas, como dependências de outras pessoas, recursos necessário, entre outros. Todavia, este artefato deve ser mantido o mais simples possível, pois é totalmente descartado ao final da iteração.

O RUP, apesar de não apresentar um modelo para este artefato, sugere seu conteúdo. Todavia, este conteúdo é bastante detalhado e formal, não sendo considerado adequado pelo próprio RUP para projetos de menor porte. Assim, sugerimos que seu conteúdo se limite ao citado acima.

Detalhes do Fluxo: Gerenciar Iteração

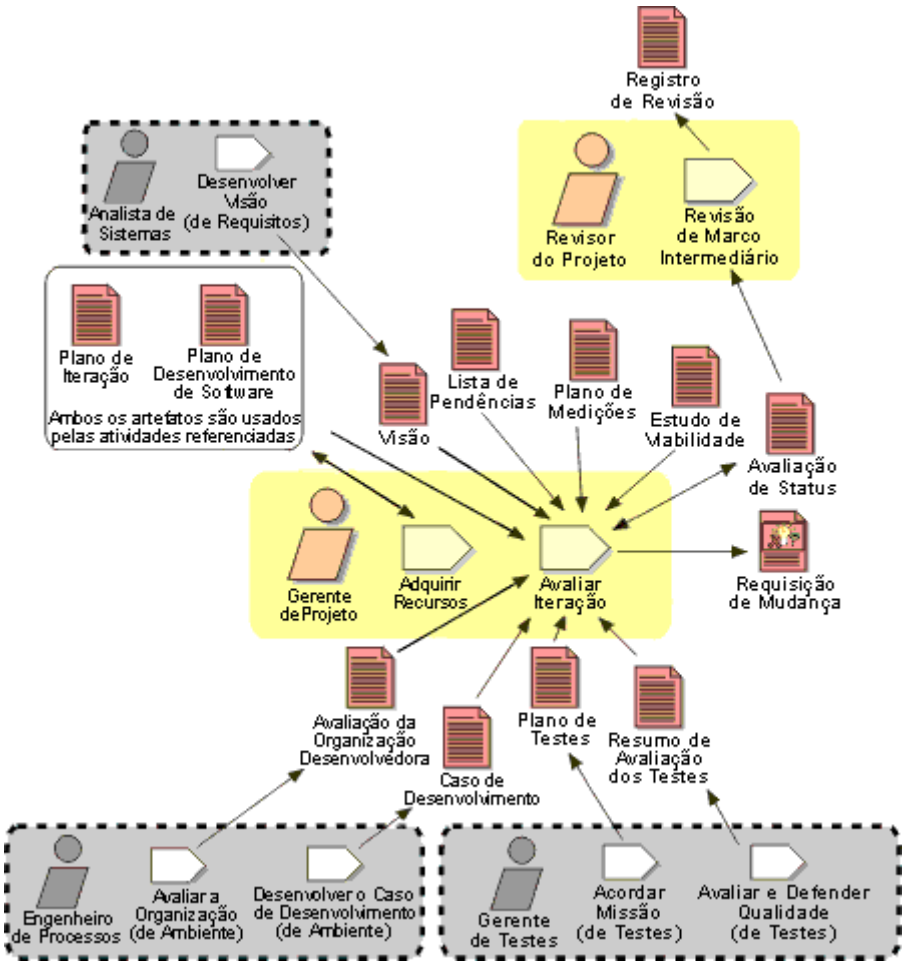


Figura 5-11 - Detalhes do Fluxo: Gerenciar Iteração

Neste Detalhe do Fluxo, apresentado na Figura 5-11, foi inserida uma nova atividade: Adquirir Recursos. Ela é o resultado da junção da atividade Adquirir Pessoal com o passo Adquirir Recursos Não-pessoais, da atividade Iniciar Iteração, o único passo que se aplica ao cenário das pequenas equipes. Os passos da atividade Adquirir Recursos foram mantidos, com exceção do passo Formar Equipes, cuja exclusão foi explicada na Seção 5.3.1. Vale observar que, nesta atividade, pelo menos os papéis propostos pelo TSP devem ser mapeados nos membros da equipe.

No passo "Mapear Habilidades da Equipe para Papéis", o Gerente de Projeto deve atribuir os papéis de suporte definidos no TSP aos membros da equipe. São eles: Gerente de Suporte, Gerente de Qualidade e Processo, Gerente de Desenvolvimento, Líder de Equipe e Gerente de Planejamento. Em geral, o Gerente de Projeto irá desempenhar os dois, em muitos casos até os três últimos papéis. Por causa do número reduzido de participantes da equipe, isto não deve ser uma atividade complexa.

O passo "Treinar Equipe do Projeto" merece uma atenção especial do Gerente de Projeto. O treinamento da equipe seja no domínio de aplicação do projeto, seja em

aspectos técnicos específicos, não é levado em conta nas metodologias ágeis nem no TSP, apesar de ser extremamente importante.

Na atividade Avaliar Iteração deve-se analisar as métricas e comparar o que fora planejado com o efetivamente realizado, tanto para as atividades de construção do produto como para as atividades dos papéis específicos, como Gerência de Configuração. Caso seja necessário, responsáveis pelos papéis podem ser trocados. No TSP, esta avaliação corresponde à fase *Postmortem*.

Esta avaliação é um complemento aos encontros diários, pois eles servem para identificar problemas nos andamentos das atividades, que possam comprometer o andamento do projeto naquela iteração. A avaliação da iteração possui um objetivo mais abrangente, pois busca identificar lições aprendidas e oportunidades de melhoria no processo, para que os erros passados não sejam cometidos na próxima iteração. Além disto, os riscos do projeto devem ser revistos, para saber quais foram mitigados, o status dos demais, e se há algum novo risco.

A atividade Realizar Revisão de Marco Intermediário possui os mesmos passos da atividade Realizar Revisão de Aceitação da Iteração, mas deve ser executada somente a cada entrega para o usuário, a cada 2 ou 3 meses, e não ao final de cada iteração. O objetivo desta atividade é validar o progresso do projeto constantemente com o cliente. A atividade Avaliar Iteração sofreu mudanças somente em seus artefatos de saída, tendo mantido seu modo de execução.

Detalhes do Fluxo: Monitorar e Controlar Projeto

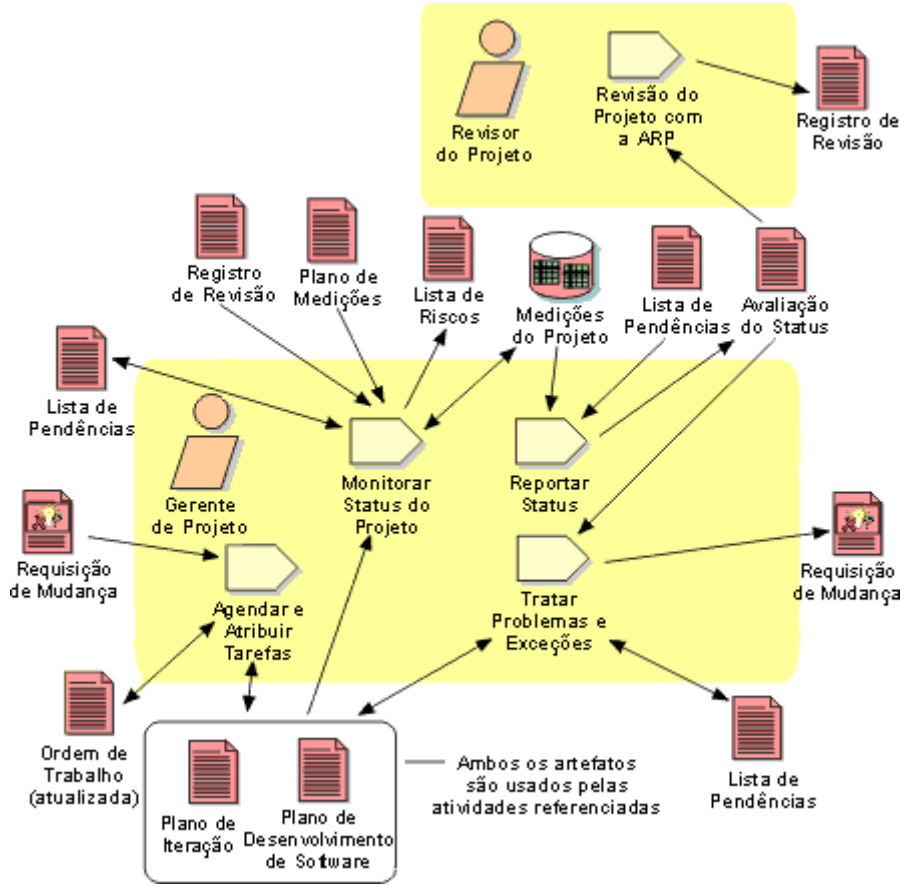


Figura 5-12 - Detalhes do Fluxo: Monitorar e Controlar Projeto

Este detalhe do fluxo, apresentado na Figura 5-12, sofreu modificações. As atividades Agendar e Atribuir Tarefas, Tratar Problemas e Exceções, e Monitorar Status foram reformuladas para incorporar a prática de encontros diários com a equipe, descrita em XP e Scrum.

A atividade Monitorar Status deve ser realizada através de encontros diários com toda a equipe. Estes encontros devem possuir duração de 15 a 30 minutos, onde cada membro informa que atividades já foram completadas, em que está trabalhando e eventuais dificuldades e problemas encontrados. Estas reuniões também permitem que a carga de trabalho entre a equipe seja constantemente balanceada, uma vez que todos possuem visibilidade sobre o a conclusão das atividades e sobre os problemas que cada um está encontrando. Como a equipe é pequena, a comunicação constante é facilitada e deve ser extremamente encorajada.

O foco destas reuniões é capturar a situação do projeto e identificar problemas, mas não resolvê-los. As pessoas que puderem contribuir para solucionar o problema devem conversar em outro momento, pois o envolvimento de todos nesta discussão é quase sempre desnecessário e improdutivo. Para isto, será utilizada a atividade Tratar Problemas e

Exceções, onde todos os membros envolvidos devem discutir a solução do problema. Os passos originais do RUP foram mantidos, com exceção da necessidade de se emitir Ordens de Trabalho para a resolução dos problemas, uma vez que este artefato foi eliminado, como apresentado na Seção 5.3.1.

A atividade Agendar e Atribuir Tarefas tem como objetivo tratar novas tarefas, não contempladas durante o planejamento da iteração. Isto é pouco provável de ocorrer, devido à curta duração das iterações. Todavia, caso isto ocorra, a nova tarefa deve substituir uma tarefa já planejada cujo esforço estimado seja semelhante. Caso não exista uma atividade semelhante que possa ser substituída, deve ser feito um replanejamento da iteração, para se decidir que tarefas serão trocadas pela nova tarefa. Deve-se evitar ao máximo mudar a data final da iteração, para que os benefícios das iterações de duração fixa não sejam comprometidos. A tarefa original deve ser replanejada para uma iteração seguinte, de acordo com a prioridade determinada pelo cliente ou especialista do negócio.

A decisão sobre a urgência da nova tarefa deve ser tomada pelo cliente ou pelo especialista do negócio, pois algo que havia sido previamente escolhido como importante não será desenvolvido, em detrimento da nova tarefa. Estas decisões devem ser informadas à equipe nos encontros diários, onde a nova tarefa deve ser escolhida por um dos membros da equipe. Assim como no planejamento normal, o membro da equipe que se candidatar à nova tarefa deve informar quanto tempo levará para executá-la.

Na atividade Reportar Status, os passos originais foram mantidos. Todavia, o status do projeto também deve ser reportado à equipe durante os encontros diários, ao invés de ser reportado somente ao nível gerencial acima do Gerente de Projeto. A atividade Revisão do Projeto com a ARP não necessitou de alterações.

Detalhes do Fluxo: Fechamento da Fase

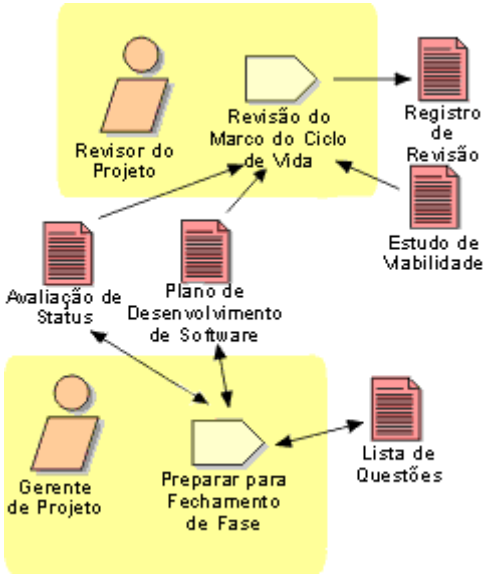


Figura 5-13 - Detalhes do Fluxo: Fechamento da Fase

Conforme apresentado na Seção 5.3.1, nenhuma das atividades deste subfluxo teve seus passos alterados. Estas atividades são apresentadas na Figura 5-13. A única alteração necessária foi a retirada de alguns artefatos de entrada da atividade Revisão do Marco do Ciclo de Vida, e a retirada de alguns artefatos de entrada e de saída da atividade Preparar para o Fechamento de Fase, devido à exclusão de outras atividades da Disciplina.

Detalhes do Fluxo: Fechamento do Projeto

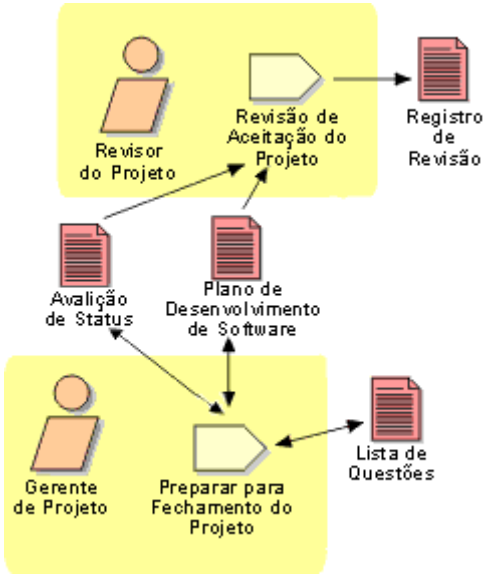


Figura 5-14 - Detalhes do Fluxo: Fechamento do Projeto

Conforme apresentado na Seção 5.3.1, nenhuma das atividades deste subfluxo teve seus passos alterados. Estas atividades são apresentadas na Figura 5-14. A única alteração necessária foi nos artefatos de entrada de ambas as atividades, que foram diminuídos devido à exclusão de outras atividades da Disciplina.

5.3.3 Correspondência com XP e TSP

A nova disciplina de Gerenciamento de Projeto aqui proposta contempla vinte dos trinta e três passos existentes nos scripts LAU1, STRAT1, PLAN1, PM1 e WEEK do TSP. Do script LAU1, não são atendidos os passos Apresentar Visão Geral do Curso, Obter Informações do Estudante, Definir Metas das Equipes e Descrever Reuniões das Equipes, por serem voltados para o ensino do TSP e não para atividades de gerenciamento do projeto. O passo Descrever Objetivos do Produto equivaleria no RUP à atividade Desenvolver Visão, que não é produzido nesta disciplina. O passo Realizar Primeira Reunião da Equipe não acontece obrigatoriamente, mas nada impede sua realização.

Do script STRAT1, o passo Apresentar Visão Geral da Estratégia também não é atendido por ser voltado para o ensino do TSP. Os passos Estabelecer Critério para Estratégia e Documentar a Estratégia não são necessários, pois a estratégia de desenvolvimento já está determinada devido às características da própria metodologia: o desenvolvimento será iterativo e incremental, com ênfase na arquitetura e nos riscos. O passo Produzir Projeto Conceitual não pertence à disciplina Gerenciamento de Projeto, e sim à disciplina Análise e Projeto. O mesmo acontece com o passo Produzir Plano de Gerência de Configuração, pertencente à disciplina Gerência de Configuração e Mudanças.

Os scripts PLAN1, PM1 e WEEK são completamente atendidos, com exceção do passo 1 dos scripts PLAN1 e PM1, Apresentar Visão Geral do Planejamento e Apresentar Visão Geral do Processo de *Postmortem*, respectivamente. Estes passos são voltados para o ensino do TSP e não para as atividades relacionadas aos *scripts*.

Adicionalmente, foram incorporadas duas práticas de XP: jogo do planejamento e encontros diários. Esta correspondência é apresentada na Tabela 13.

Atividade do RUP-pe	TSP – passo/script	Prática de XP	Observação
Identificar e analisar riscos	6 (STRAT1)	–	–
Desenvolver Estudo de Viabilidade	–	–	–
Iniciar Projeto	9 (LAU1)	–	–
Realizar Revisão de Aprovação do Projeto	–	–	–
Desenvolver Plano de Medições	8 (LAU1)	–	Esta atividade foi simplificada. O Plano de Medições é parte do Plano de Desenvolvimento de Software.
Desenvolver Plano de Aceitação do Produto	–	–	Esta atividade foi simplificada. O Plano de Aceitação do Produto é parte do Plano de

			Desenvolvimento de Software.
Desenvolver Plano de Garantia da Qualidade	5 (PLAN1)	–	–
Definir organização e equipe do projeto	4 (LAU1)	–	–
Planejar fases e iterações	4, 5 (STRAT1)	–	–
Compilar Plano de Desenvolvimento de Software	Todo o script PLAN1, exceto o passo 1	–	–
Revisão do planejamento do projeto	–	–	–
Agendar e atribuir tarefas	6 (PLAN1)	Encontros Diários, Jogo do Planejamento (se necessário)	–
Monitorar status do projeto	Todo o script WEEK	Encontros Diários	–
Reportar status	Todo o script WEEK	Encontros Diários	A prática de XP é utilizada somente ao se reportar o status para a equipe.
Tratar problemas e exceções	–	–	–
Revisão de projeto com Autoridade de Revisão do Projeto (ARP)	–	–	–
Desenvolver Plano de Iteração	2, 3, 4, 6 e 7 (PLAN1)	Jogo do Planejamento	–
Adquirir recursos necessários	4 (LAU1)	–	O TSP trata somente da aquisição de pessoal. Esta atividade foi simplificada.
Avaliar iteração	Todo o script PM1, exceto o passo 1	–	–
Revisão de marco intermediário	–	–	–
Preparar para fechamento de fase	Critério de entrada do script PM1	–	–
Revisão do marco do ciclo de vida	Todo o script PM1, exceto o passo 1	–	–
Preparar para fechamento do projeto	–	–	–
Revisão de aceitação do projeto	–	–	–

Tabela 13 - Correspondência entre a nova disciplina de Gerenciamento de Projetos, XP e o TSP.

5.4 Considerações Finais

Neste capítulo apresentamos a metodologia RUP-pe, voltada para equipes de até 15 desenvolvedores. Ela é uma metodologia baseada no RUP, que incorpora princípios das metodologias ágeis e está alinhada com o TSP, e, por conseqüência, com o SW-CMM.

Foram descritas as disciplinas Implementação e Gerenciamento de Projeto. Ambas são simplificações das disciplinas originais do RUP, onde as atividades e a maneira

de executá-las foi revista, buscando simplificá-las e incorporar os princípios de formação de equipe do TSP. A correspondência das disciplinas propostas com o TSP e com XP também foi apresentada.

O RUP-pe conta também com um *website* [12], onde todas as atividades aqui discutidas estão descritas em detalhes, bem como modelos dos artefatos e guias gerais da metodologia, com o objetivo de orientar as equipes que queiram utilizar a proposta.

Capítulo 6: Análise Crítica das Práticas do RUP-pe

6.1 Introdução

Ao definir uma metodologia de desenvolvimento de software, é necessário verificar se a mesma é realmente viável, do ponto de vista de quem irá utilizá-la na prática: as equipes de desenvolvimento de software. Em um cenário ideal, a viabilidade do RUP-pe poderia ser verificada aplicando-se a metodologia em um projeto real, cuja equipe possuísse até 15 desenvolvedores. Todavia, a adoção desta abordagem possui uma série de dificuldades:

1. Encontrar uma empresa disposta a implantar uma nova metodologia: devido à pequena margem de manobra que as empresas possuem atualmente, elas não querem correr o risco de testar algo novo, que possa afetar o custo ou o prazo de conclusão dos seus projetos, mesmo que as práticas atuais de desenvolvimento não sejam suficientes.
2. A necessidade de se acompanhar um projeto durante toda sua duração para se validar a metodologia: seria necessário acompanhar um projeto desde o início, até sua conclusão, para verificar se todas as atividades propostas realmente são consideradas úteis pela equipe do projeto. Isto demanda um tempo considerável, pois estes projetos em geral duram vários meses.
3. Características específicas de um projeto: a adoção do RUP-pe com sucesso em um projeto não necessariamente implica que a metodologia se aplique a projetos com outras características (ex: equipes geograficamente distribuídas, sistemas críticos). Assim, o ideal seria avaliar seu uso em mais de um projeto, o que é extremamente difícil pelas questões já apresentadas acima.

4. O RUP-pe ainda não é uma metodologia completa: como só foram definidas duas disciplinas, utilizar o RUP-pe em conjunto com a maneira como a organização desenvolvedora pratica as outras disciplinas, como por exemplo, Requisitos e Análise e Projeto, não permite avaliar seguramente o RUP-pe. Em caso de fracasso do projeto, por exemplo, pode não ser possível determinar se o que mais contribuiu para isto foram atividades do RUP-pe ou as outras atividades do desenvolvimento.

Assim, para validar nossa proposta, utilizamos uma abordagem alternativa. Foram elaborados questionários com perguntas sobre as principais práticas incorporadas à metodologia e sobre os artefatos que foram afetados (eliminados ou modificados), divididos em dois perfis: Gerente de Projeto/Líder de Equipe, e Arquiteto/Engenheiro de Software. No primeiro perfil, foram abordadas práticas relativas ao gerenciamento do projeto, como o uso de métricas e a forma de acompanhamento de atividades. No segundo, além das práticas de gerenciamento de projeto, foram abordadas práticas de implementação, como a adoção de um padrão de codificação e o uso de programação em pares. Com exceção das perguntas sobre o perfil do entrevistado, todas as demais perguntas eram restritas a projetos cujas equipes possuíssem, no máximo, 15 integrantes. Os modelos de questionários aplicados são apresentados no Apêndice B – Questionários. Estes questionários tinham dois objetivos principais:

- Verificar quais das práticas propostas já tinham sido utilizadas pelos entrevistados, e que benefícios ou problemas elas trouxeram. Caso a prática não tenha sido utilizada, buscou-se saber se ela seria viável, com base na experiência dos entrevistados;
- Verificar se os artefatos excluídos ou modificados são considerados importantes ou não pelos entrevistados, com base em sua experiência.

Os questionários foram distribuídos para desenvolvedores de 3 empresas de desenvolvimento de software do Ecossistema de Recife, as quais adotavam o RUP como processo padrão de desenvolvimento. Estes desenvolvedores não possuíam nenhum conhecimento prévio em relação ao RUP-pe e foram escolhidos ao acaso. A única restrição na escolha dos entrevistados foi tentar distribuir o mesmo número de questionários dos dois perfis. De um total de 38 questionários distribuídos, sendo 20 do perfil Gerente de Projeto/Líder de Equipe e 18 do perfil Arquiteto/Engenheiro de Software, foram devolvidos 6 do primeiro perfil e 16 do segundo.

Esta abordagem está longe de ser a ideal, tanto pelo número de respostas obtidas, como por não ser uma aplicação prática da metodologia. Entretanto, ela nos permite ter uma idéia da viabilidade das principais mudanças incorporadas à nossa metodologia, além de possuir algumas vantagens. A primeira é que, por contar com uma diversidade de tipos de projetos e profissionais com experiências diferentes, a análise da metodologia é feita de forma mais abrangente, pois leva em conta um maior número de diferentes cenários. Além disso, é possível avaliar somente as mudanças que estão sendo propostas, sem necessidade de se examinar práticas de outras disciplinas que não puderam ser definidas.

6.2 Análise Crítica dos Resultados Obtidos

Nesta seção, apresentamos uma análise crítica dos resultados obtidos com a aplicação dos questionários. Inicialmente, é descrito o perfil dos entrevistados. Em seguida, procuramos encontrar evidências do uso e de benefícios das práticas incorporadas ao RUP-pe. Além disto, buscarmos uma relação entre o conhecimento sobre XP e a adoção de tais práticas, como maneira de verificar se mesmo quem não conhece XP faz uso e obtém benefícios da maioria das práticas. Por fim, são analisados os artefatos originais do RUP que foram impactados na definição do RUP-pe e são feitas as considerações finais sobre os resultados obtidos.

6.2.1 Perfil dos Entrevistados

A primeira parte dos questionários teve como objetivo traçar um perfil dos entrevistados, do ponto de vista da experiência em desenvolvimento de software e do uso de práticas de XP. Para isso, foram elaboradas perguntas sobre a experiência técnica, experiência no domínio da aplicação e no processo de desenvolvimento, e a respeito do conhecimento prévio sobre XP.

A Figura 6-1 mostra a experiência técnica dos entrevistados, ou seja, de quantos projetos já participaram antes do projeto que estão trabalhando atualmente. Não foi levado em consideração o tempo de duração destes projetos. Como podemos observar, 54% dos entrevistados possui uma experiência de mais de 4 projetos. Este número sobe para 86%, se considerarmos aqueles que já participaram de pelo menos 2 projetos antes do projeto atual. Os entrevistados possuem, em sua maioria, uma boa experiência em projetos de software.

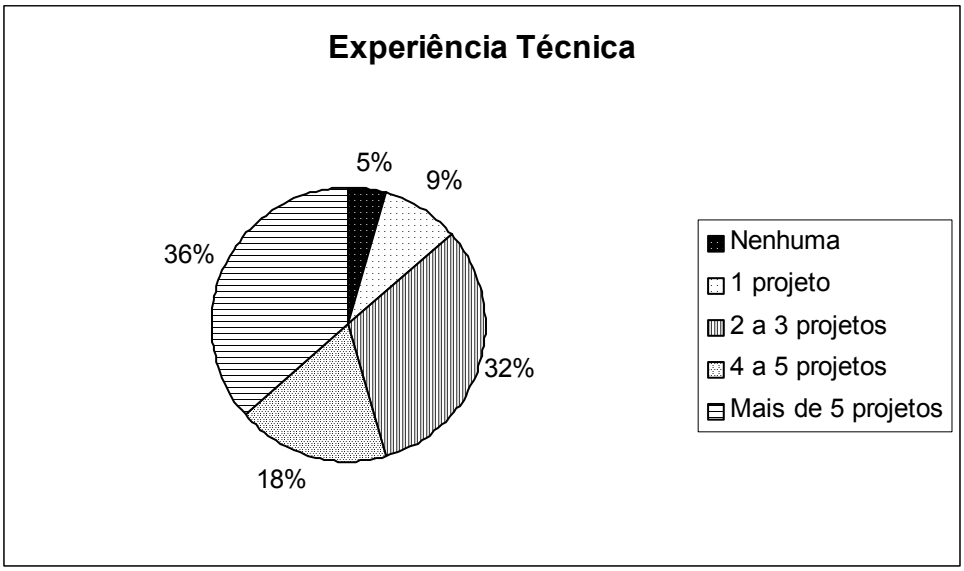


Figura 6-1 - Experiência técnica dos entrevistados

Na Figura 6-2, temos o resultado da experiência no domínio da aplicação, ou seja, o número de projetos em um mesmo domínio, que o entrevistado participou antes do projeto atual. Exemplos de possíveis domínios são sistemas para a área de saúde ou jogos. O objetivo desta pergunta era tentar levantar a diversidade de projetos já vivenciada pelos entrevistados. Quanto maior a diversidade de projetos, maior a importância da opinião sobre as práticas propostas, pois podemos verificar sua viabilidade em um número maior de situações diferentes.

Observamos que 40% dos entrevistados já participou de 2 a 3 projetos no mesmo domínio de aplicação, enquanto 55% deles possui, no máximo, 1 projeto de experiência no domínio de aplicação. Nenhum dos entrevistados possui mais de 5 projetos de experiência no mesmo domínio.

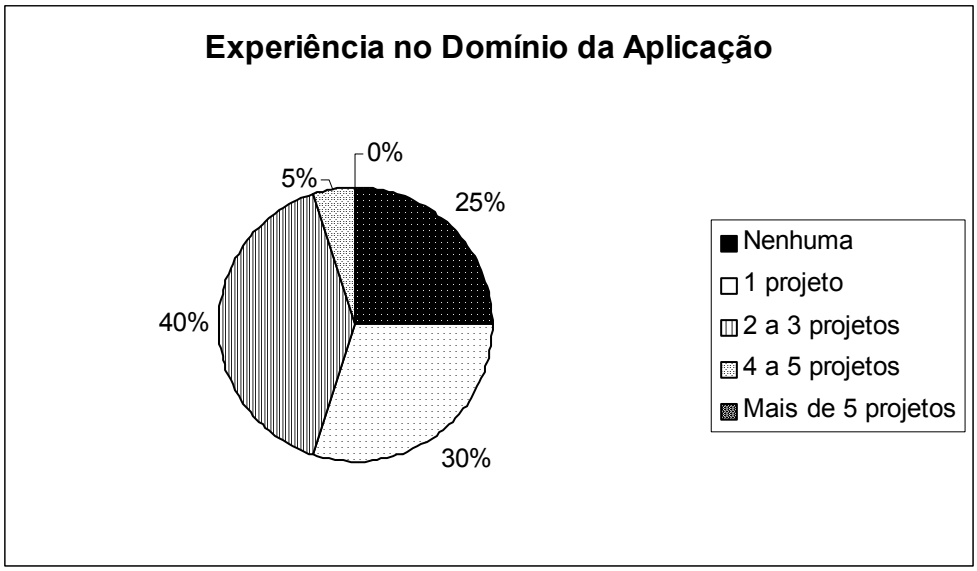


Figura 6-2 - Experiência no domínio da aplicação

Outro fator examinado foi a experiência dos entrevistados no processo utilizado atualmente, a qual é apresentada na Figura 6-3. Por experiência no processo estamos considerando o número de projetos dos quais o entrevistado participou que utilizava o mesmo processo utilizado no projeto atual. Não foram levados em consideração que processos eram utilizados nem seus respectivos níveis de maturidade.

A experiência no processo é importante no momento de se julgar as análises sobre a utilidade dos artefatos, uma vez que este julgamento pode ser influenciado pelo pouco conhecimento sobre o processo atual ou pela existência de antigos vícios na execução das atividades.

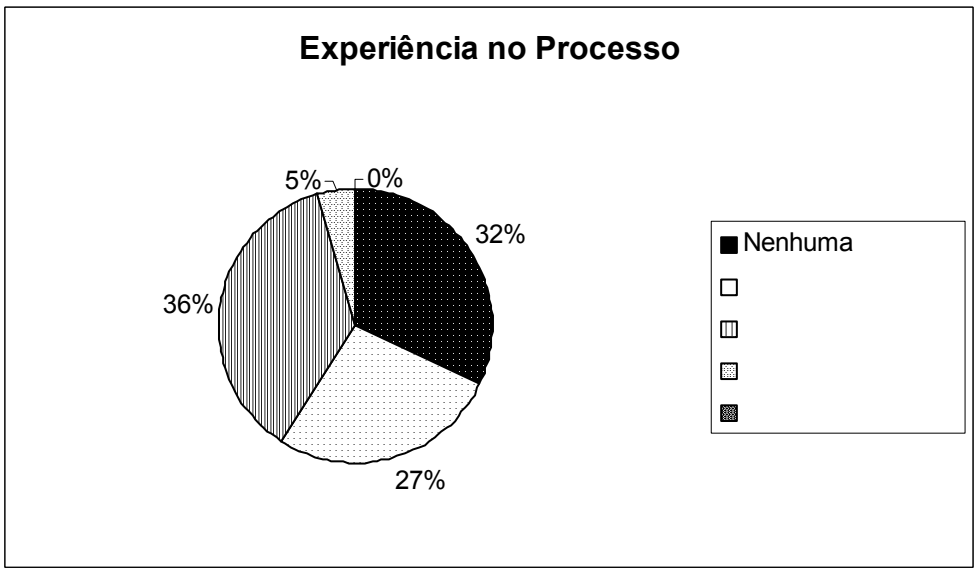


Figura 6-3 - Experiência no processo de desenvolvimento

Observamos que 32% dos entrevistados não possuía nenhuma experiência no processo utilizado atualmente, enquanto 59% possuía experiência de, no máximo, 1 projeto. A grande maioria, ou seja, 95% dos entrevistados, possui experiência de, no máximo, 3 projetos. Nenhum dos entrevistados possui mais de 5 projetos de experiência no mesmo processo.

O último ponto analisado sobre o perfil dos entrevistados foi o conhecimento prévio de XP, a metodologia ágil que contribuiu com o maior número de práticas para o RUP-pe. Esta questão tinha por objetivo buscar uma relação entre a adoção de práticas ágeis e o conhecimento sobre XP, ou seja, se estas práticas estavam sendo utilizadas somente pelo fato de XP estar em evidência na literatura ou se realmente eram as chamadas "boas práticas", que têm se mostrado efetivas e, por isso, são repetidas nos projetos seguintes. Os resultados são apresentados na Figura 6-4.

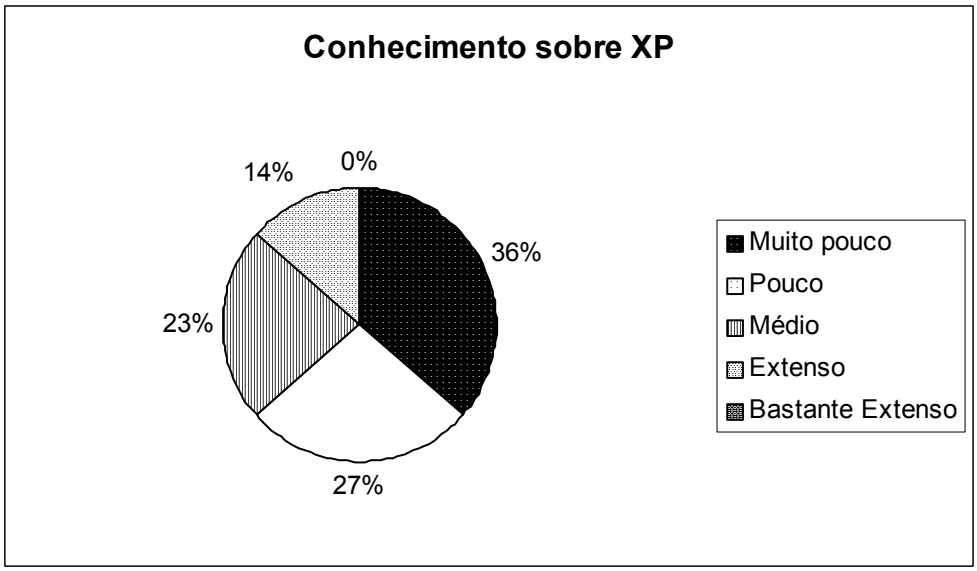


Figura 6-4 - Conhecimento sobre XP

A maior parte, isto é, 63% dos entrevistados avaliou como "muito pouco" ou "pouco" seu conhecimento sobre XP. Este número sobe para 86% quando consideramos o conhecimento avaliado como, no máximo, médio. Nenhum dos entrevistados considerou seu conhecimento sobre XP bastante extenso. Podemos observar que são desenvolvedores que possuem alguma informação a respeito de XP, mas não podem ser considerados especialistas no assunto.

Ao considerarmos estes dados em conjunto, podemos perceber que as pessoas entrevistadas possuem uma experiência relativamente grande e bastante diversificada. Este dado é importante na hora de analisarmos o julgamento das práticas propostas. A grande homogeneidade dos tipos de projeto e/ou a pouca experiência dos entrevistados

certamente seriam fatores que comprometeriam a credibilidade do julgamento, já que não mostrariam os benefícios ou problemas das práticas sugeridas em diferentes situações.

6.2.2 Práticas de Gerenciamento de Projeto

Nesta seção apresentamos a análise das práticas propostas para a disciplina Gerenciamento de Projeto do RUP-pe. Através destas questões, que constavam em ambos os perfis de questionários, procuramos identificar o uso das práticas pelos entrevistados e, no caso de não terem sido utilizadas, a opinião dos entrevistados sobre a viabilidade de prática com base em suas experiências.

Iterações de curta duração (entre 2 e 4 semanas)

A primeira prática questionada foi o uso de iterações de curta duração, ou seja, entre 2 e 4 semanas. Como se pode observar na Figura 6-5, 68% dos entrevistados afirmou já ter utilizado esta prática alguma vez, sendo que 59% já utilizou em até 3 projetos, enquanto 9% já utilizou esta prática em mais de 3 projetos. Nenhum dos entrevistados que não utilizou achou que a prática não funcionaria.

Dentre as pessoas que afirmaram já ter utilizado-a pelo menos uma vez, 36% delas classificou seu conhecimento em XP como pouco ou muito pouco. Isto pode ser um indicativo de que as pessoas têm optado por esta estratégia após tomar conhecimento da abordagem de desenvolvimento proposta pelas metodologias ágeis.

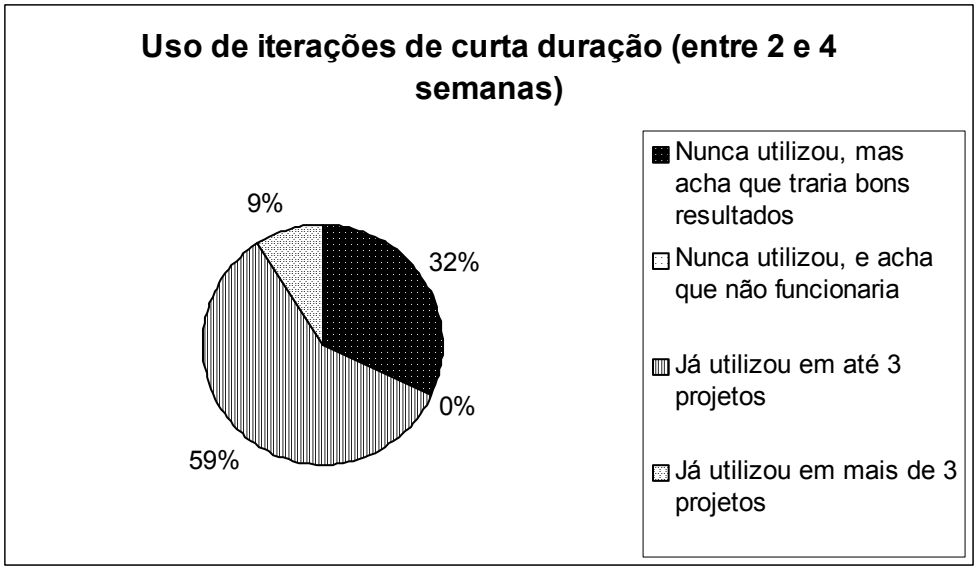


Figura 6-5 - Uso de iterações de curta duração

Entre os benefícios citados, podemos destacar: melhoria das estimativas, melhor acompanhamento do projeto e *feedback* do cliente. Entretanto, foram citados como preocupações a sobrecarga causada pela necessidade de planejamentos constantes e a

necessidade de se estimar bem o esforço necessário, para que não haja uma "correria" ao final da iteração de modo a se terminar a tempo o trabalho necessário. O primeiro problema é tratado no RUP-pe através de um planejamento de iteração feito de forma simples. Este planejamento não deve consumir muito tempo da equipe e, pelo fato desta atividade ser repetida constantemente, as estimativas conseqüentemente tendem a melhorar cada vez mais. Isto é apoiado pela coleta e divulgação das métricas, o que permite à equipe ter noção de sua real capacidade de trabalho, resolvendo o segundo ponto.

Participação de toda a equipe no planejamento (estimativas e divisão das tarefas)

Nesta pergunta procurou-se investigar a participação de toda a equipe do projeto nas atividades de planejamento, em especial, nas estimativas e na divisão de tarefas. O resultado, apresentado na Figura 6-6, mostra que 72% dos entrevistados já utilizou esta prática em até 3 projetos, um número bastante expressivo. Entre os 77% que já utilizaram esta prática em pelo menos um projeto, 45% classificou seu conhecimento em XP como muito pouco ou pouco. Entre os que não utilizaram esta prática, 23% dos entrevistados, ou seja, a totalidade, acha que ela traria bons resultados em um projeto real.

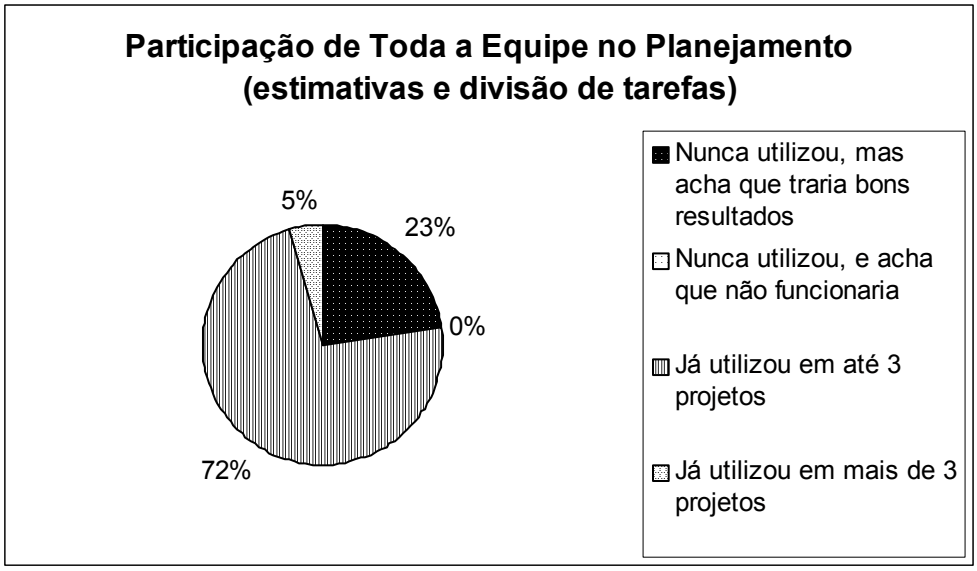


Figura 6-6 - Participação de toda a equipe no planejamento

As principais vantagens citadas de se ter toda a equipe participando do planejamento foram o aumento do comprometimento com o planejamento e a melhoria das estimativas. Os problemas apontados foram: erros nas estimativas por desconhecimento do negócio e dispersão do grupo durante as reuniões de planejamento. Mais uma vez, os erros podem ser corrigidos através das iterações de curta duração, pois as estimativas vão sendo refinadas. O problema de dispersão do grupo deve ser contornado

pela própria equipe, principalmente pelo Gerente de Projeto, o qual deve atuar como moderador destas reuniões.

Balanceamento constante da carga de trabalho entre a equipe

Ao serem questionados sobre o balanceamento constante da carga de trabalho entre a equipe, 63% dos entrevistados afirmou já ter utilizado esta prática em até 3 projetos, como podemos observar na Figura 6-7. Deste total, 36% classificou seu conhecimento em XP como muito pouco ou pouco. Os 32% que não utilizaram julgaram que esta prática traria bons resultados. O principal benefício citado foi a possibilidade de se cumprir o planejamento da iteração, já que as tarefas vão sendo realocadas à medida que os recursos vão se tornando disponíveis. Entretanto, foi notado que é essencial que toda a equipe possua conhecimento para a realização da maioria das tarefas, senão de todas.

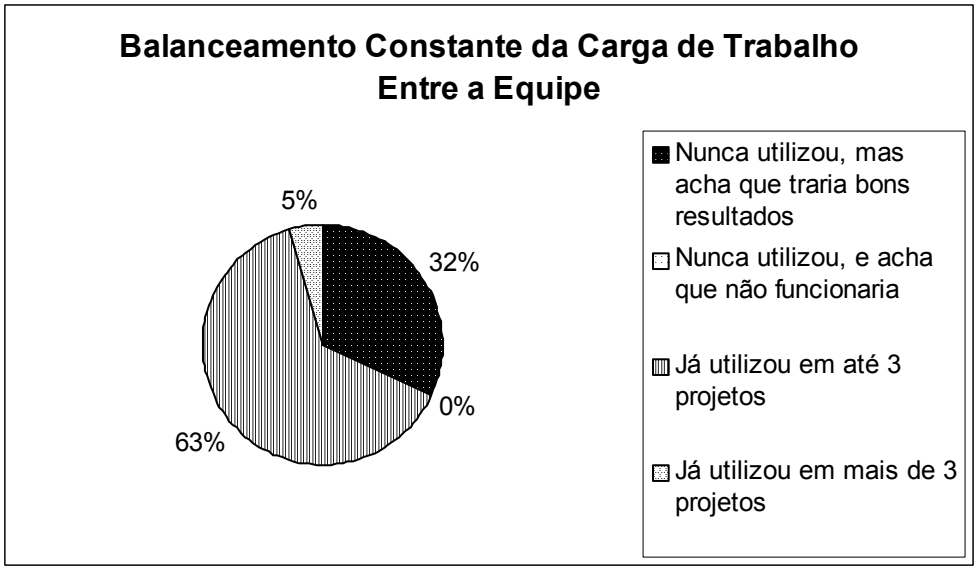


Figura 6-7 - Balanceamento constante da carga de trabalho entre a equipe

No RUP-pe, através dos encontros diários, da programação em pares e da propriedade coletiva do código, promove-se a comunicação entre a equipe e a disseminação do conhecimento sobre o sistema a ser desenvolvido. Além disto, os encontros diários possibilitam uma identificação precoce de pessoas que estejam super-alocadas ou com dificuldades para a realização de suas atividades.

Acompanhamento do progresso e dos problemas do projeto através de encontros diários com a equipe

Nesta pergunta, foi investigado o uso de encontros diários para o acompanhamento do projeto. A Figura 6-8 nos mostra que 37% dos entrevistados já utilizou esta prática em pelo menos um projeto, enquanto 31% nunca utilizou, mas acha

que traria bons resultados ao projeto. Entre os benefícios, foram citados a identificação de problemas e o fato de todos ficarem a par da situação do projeto.

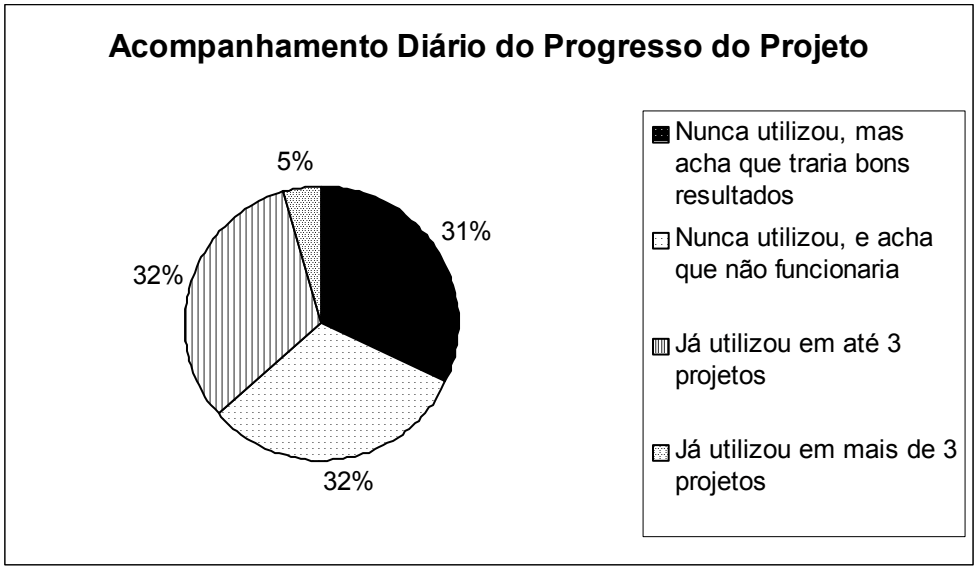


Figura 6-8 - Acompanhamento diário do progresso do projeto

Dos 32% que afirmaram não ter utilizado e achar que a prática não funcionaria, os principais problemas indicados foram o excesso de reuniões e a improdutividade destes encontros. Foram bastante sugeridos encontros semanais ao invés de diários.

É importante notar que o questionário não descrevia a forma como estes encontros deveriam ser realizados, o que pode ter induzido os entrevistados a pensarem que se tratava de uma reunião formal, e não um encontro de meia hora, no máximo. Um indicativo disto é o fato de que dentre os que utilizaram, 18% classificou seu conhecimento em XP como muito pouco ou pouco, ou seja, 82% dos entrevistados que utilizam encontros diários classificou seu conhecimento em XP pelo menos como mediano.

Uso de métricas para acompanhar o progresso do projeto e melhorar estimativas e re-planejamento

Outro ponto investigado foi o uso de métricas para o planejamento e acompanhamento do projeto, em especial, acompanhamento de progresso e melhoria das estimativas, levando, conseqüentemente, a um melhor replanejamento. A Figura 6-9 nos mostra que esta prática ainda não é muito difundida, uma vez que apenas 36% dos entrevistados afirmou fazer uso de métricas em até 3 projetos. O restante, ou seja, 64% dos participantes, afirmou que nunca utilizou, mas acredita que traz bons resultados. Entre os poucos comentários dos entrevistados, destacam-se a maior visibilidade sobre o que está acontecendo com o projeto. Foi apontado como dificuldade a obtenção dos dados para a geração das métricas e a análise das mesmas.

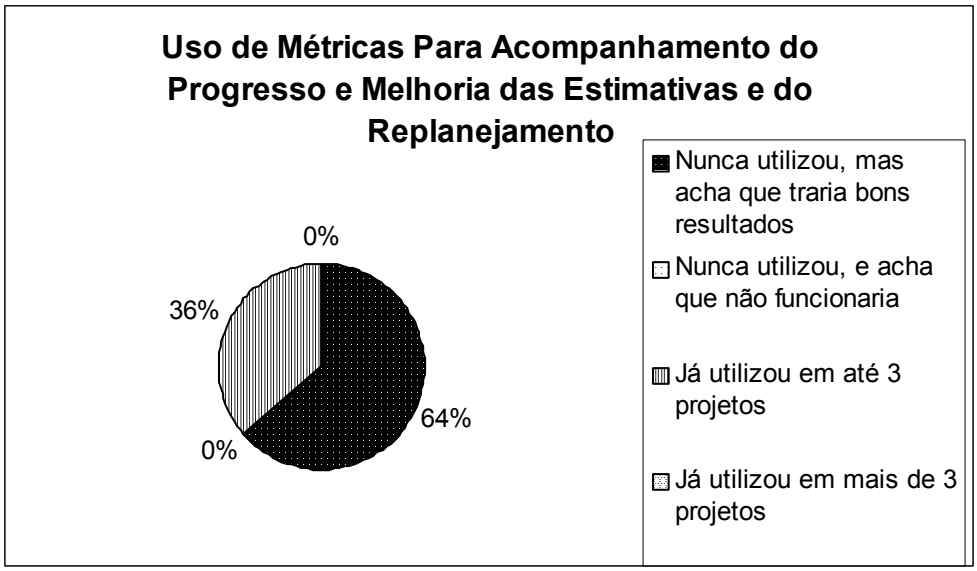


Figura 6-9 - Uso de métricas para acompanhamento do progresso e melhoria das estimativas e do replanejamento

O RUP-pe reconhece a importância de se utilizar métricas para apoiar o gerenciamento do projeto e aborda estes problemas através da definição de métricas simples, que possam ser facilmente coletadas e analisadas. Além disto, são dadas sugestões de métricas que podem ser coletadas, como mostramos na Seção 5.3.1.

Um ponto que merece ser observado é que entre os 36% que utilizaram métricas, 14% classificou seu conhecimento de XP como muito pouco ou pouco, ou seja, a grande maioria possui um conhecimento, no mínimo, médio sobre XP. Isto é um indício de que as métricas propostas por XP realmente são úteis e sua adoção não traz uma sobrecarga ao gerenciamento do projeto.

Cliente ou especialista do negócio participando da definição do escopo das iterações

No que diz respeito à participação do cliente ou de um especialista do negócio na definição do escopo das iterações, 41% dos entrevistados já utilizou em até 3 projetos, enquanto 54% nunca utilizou, mas acha que traria bons resultados. Entre os que utilizaram, 77% classificou seu conhecimento em XP como, no mínimo, médio.

Foram apontados como principais benefícios da adoção desta prática a certeza de se estar trabalhando em algo importante para o cliente e o envolvimento do cliente no planejamento, deixando-o ciente das atividades, prazos e limitações da equipe.

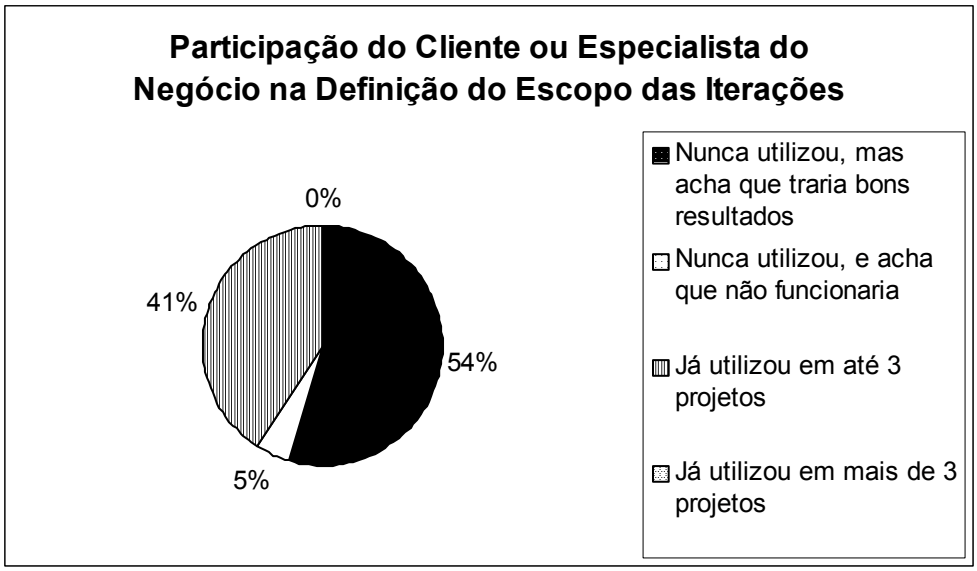


Figura 6-10 - Participação do cliente ou do especialista do negócio na definição do escopo das iterações

Entre os 5% que nunca utilizaram e afirmaram que a prática não traria um bom resultado, foram apontados como problemas o fato do cliente muitas vezes não saber bem o que quer e querer as coisas realizadas em um prazo irreal. Entretanto, acreditamos que a participação do cliente ou de um especialista supera estas dificuldades por aumentar seu envolvimento e comprometimento com as atividades de planejamento. Dessa forma, fica claro para o cliente/especialista por que determinadas atividades devem ser adiadas ou que riscos estão sendo assumidos.

6.2.3 Práticas de Implementação

Nesta seção apresentamos a análise das práticas propostas para a disciplina Implementação do RUP-pe. Assim como nas questões da seção anterior, procuramos identificar o uso das práticas pelos entrevistados e, no caso de não terem sido utilizadas, a opinião dos entrevistados sobre a viabilidade de prática com base em suas experiências. As questões sobre as práticas de implementação faziam parte somente do questionário aplicado para Arquitetos e Engenheiros de Software.

Adoção de um padrão de codificação

Entre os desenvolvedores entrevistados, a imensa maioria, ou seja, 87% já adotou um padrão de codificação em pelo menos um projeto, onde 74% do total de entrevistados já utilizou esta prática em mais de três projetos. Estes resultados são mostrados na Figura 6-11.Certamente, esta é uma das práticas mais difundidas. Entre os que já adotaram um padrão de codificação, 63% classificou seu conhecimento de XP como muito pouco ou

pouco, indicando que esta prática já é bem estabelecida entre a comunidade de desenvolvimento entrevistada.

Entre os benefícios citados, estão a maior facilidade de entendimento do código e, como conseqüência, a maior facilidade de manutenção. Nenhum dos entrevistados que não utilizaram esta prática achou que ela não funcionaria.

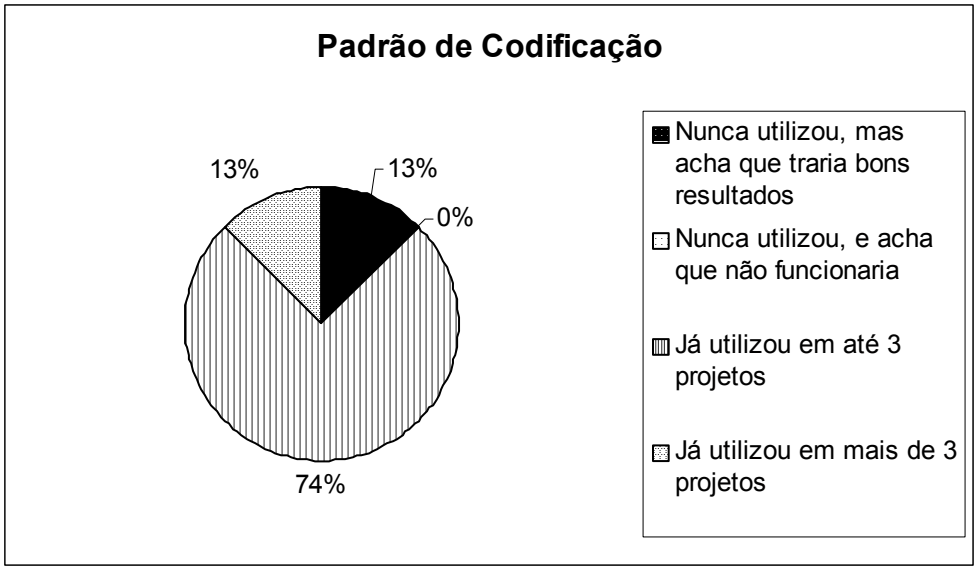


Figura 6-11 – Adoção de um padrão de codificação

Integração contínua

A Figura 6-12 nos mostra os resultados a respeito do uso da integração contínua. A maioria dos desenvolvedores afirmou já ter utilizado em até 3 projetos, ou seja, 56% dos entrevistados. Dentre os que adotaram, apenas 38% classificou seu conhecimento de XP como muito pouco ou pouco, o que indica uma relação entre o conhecimento desta metodologia e a adoção dessa estratégia de integração. Entre os que nunca utilizaram esta prática, 31% afirmou acreditar que ela seria benéfica, enquanto 13% não achou que ela fosse trazer benefícios.

Entre as vantagens da integração contínua, foram citadas a diminuição dos problemas de integração e a identificação de problemas mais precocemente, evitando surpresas ao final da iteração. Como problema, foi apontada somente a preocupação com o tempo gasto nesta atividade, que poderia causar perda de produtividade. Isto, entretanto, não representa um grande problema caso a estratégia do RUP-pe, apresentada na Seção 5.2.2, for seguida.

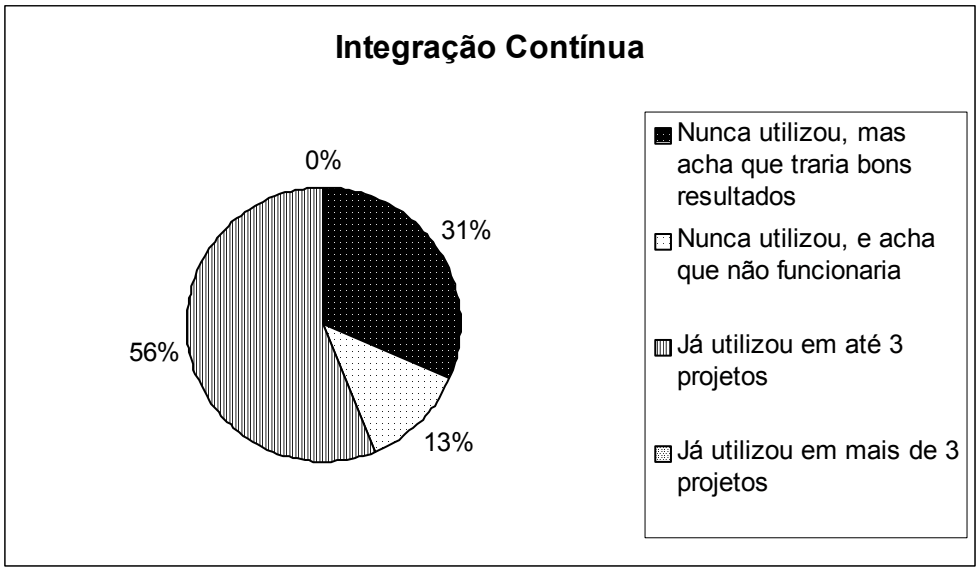


Figura 6-12 - Uso de integração contínua

Como sugerimos que a integração seja feita com o uso de ferramentas, a intervenção do desenvolvedor só será necessária caso haja erros. Certamente, é melhor identificar estes erros durante a integração somente do seu próprio código, do que ao final da iteração, caso todos os códigos sejam integrados de uma só vez. Além disto, como os trechos de código integrados são pequenos, pois a integração é feita em curtos espaços de tempo, a resolução de conflitos se torna mais simples.

Programação em pares

A programação em pares mostrou-se uma prática com um bom índice de adoção, como mostra a Figura 6-13. A metade dos entrevistados afirmou já ter utilizado esta prática em até 3 projetos. Este número sobe para 56% se considerarmos também os que utilizaram em mais de 3 projetos. Deste total, 31% classificou seu conhecimento em XP como muito pouco ou pouco.

Entre os benefícios foram destacadas a melhoria da comunicação, a disseminação do conhecimento e a melhor adaptação de novatos, além da maior qualidade do código e a solução mais rápida de problemas.

Esta foi uma das práticas com maior índice de rejeição por parte das pessoas que nunca tinham utilizado. 31% das pessoas que não utilizaram-na acham que ela não funcionaria. Os problemas apontados foram perda de produtividade, desperdício de recursos e problemas de relacionamento entre os membros. Estudos sobre a programação em pares mostram que os dois primeiros problemas não são verdade. Williams [66] mostra que utilizando-se a programação em pares, o tempo de desenvolvimento é apenas cerca de 15% maior do que codificando-se sozinho, e não o dobro do tempo, como em geral é

imaginado. Em compensação, a quantidade de defeitos do código é bem menor, com a necessidade de menos esforço gasto nas correções de erros.

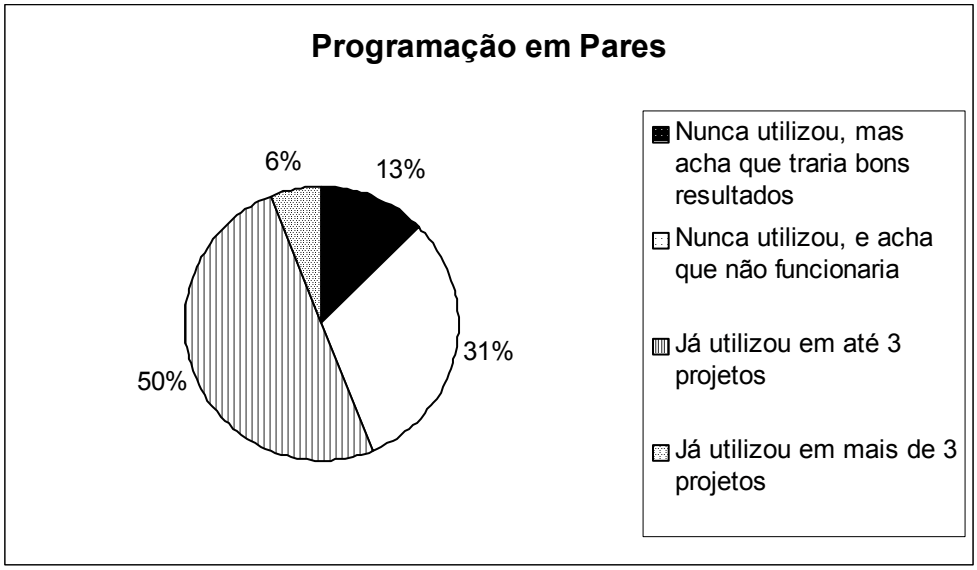


Figura 6-13 - Adoção de programação em pares

No que diz respeito ao relacionamento entre os membros, Williams mostra em um experimento que por volta de 80% dos desenvolvedores afirmam sentir mais satisfação ao programar em pares. Além disto, é importante lembrar que os pares devem ser trocados constantemente, e os papéis de um par também, como explicamos na Seção 3.2.2. Outros estudos sobre benefícios da programação em pares podem ser vistos em [79], [80] e [81].

Desenvolvimento de testes unitários para todo o sistema, com execução automatizada

O desenvolvimento de testes unitários com a execução automatizada é mais uma prática considerada benéfica pelos entrevistados. 56% deles afirmaram ter desenvolvido testes unitários com execução automatizada em até 3 projetos. Entre os 44% que não utilizaram a prática, 38% afirmou que ela seria benéfica. A Figura 6-14 mostra estes resultados.

Entre os benefícios citados, destacam-se o ganho da qualidade do produto final e a verificação contínua da qualidade do software. Como problemas, foram citados o trabalho na elaboração destes testes, e o tempo gasto nesta atividade.

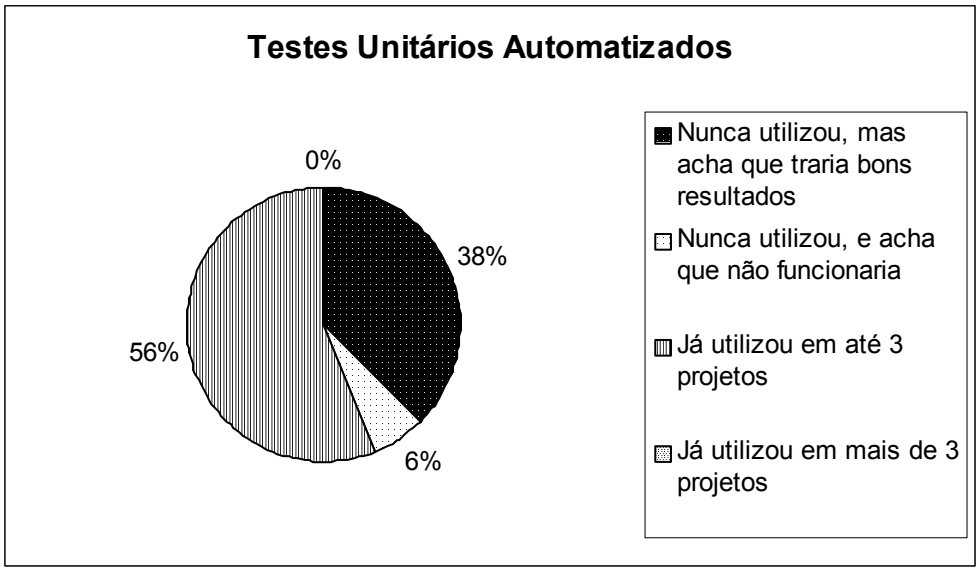


Figura 6-14 – Elaboração de testes unitários automatizados para todo o sistema

Acreditamos que este esforço é compensado pela facilidade de se encontrar erros, em virtude dos testes de regressão, e pela confiança no sistema, por causa da quantidade e facilidade de execução dos casos de testes. Além disto, o esforço inicial gasto para automatizar os testes certamente é menor do que o necessário para se executá-los com freqüência manualmente. Foi sugerida pelos entrevistados a existência de uma equipe só para esta atividade. Entretanto, em pequenas equipes, isto se torna inviável.

Elaboração dos testes antes do início da implementação

A Figura 6-15 mostra os resultados a respeito da adoção da estratégia *test-first development*, isto é, a elaboração dos testes antes da implementação dos componentes. Apenas 25% dos entrevistados já utilizou esta prática, dos quais apenas 13% classificou seu conhecimento em XP como muito pouco ou pouco. Ninguém afirmou ter utilizado esta prática em mais de 3 projetos, enquanto 44% dos entrevistados afirmou não ter utilizado, mas acha que traria bons resultados.

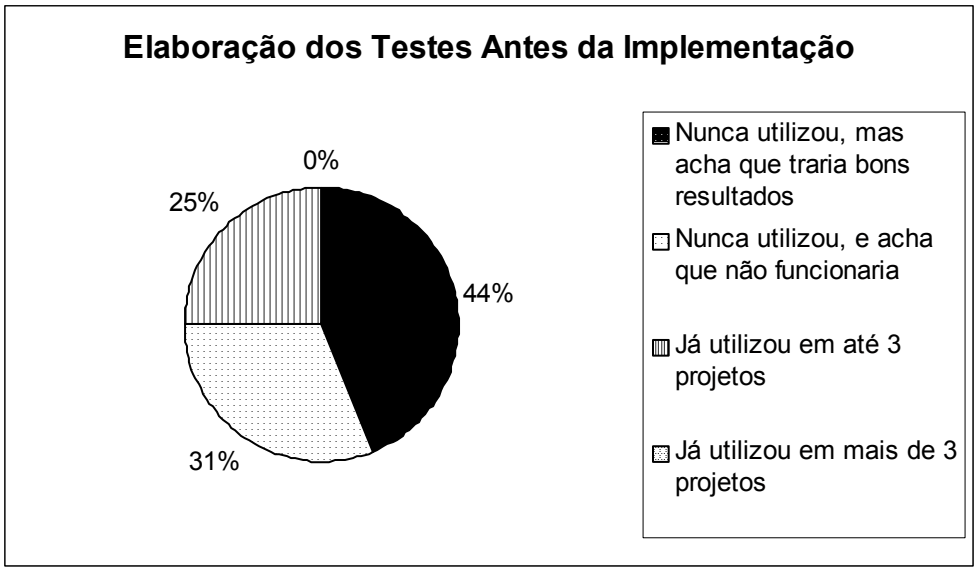


Figura 6-15 - Elaboração dos testes antes da implementação

Dentre os 31% que não utilizaram esta prática e acharam que ela não funcionaria, o principal motivo apontado foi não saber como elaborar testes para algo que ainda não foi implementando. Entretanto, estas pessoas não levaram em conta que o que deve ser usado como base para a elaboração dos testes são os requisitos. Além disto, como já explicamos nas seções 4.2.6 e 5.2.1, ao se pensar nos casos de teste antes da implementação dos componentes, é mais fácil identificar falhas no projeto do sistema. Acreditamos que a maior disseminação e melhor entendimento desta prática entre os desenvolvedores tende a ressaltar seus benefícios.

***Refactoring* do Código**

A re-estrutaração do código, como forma de eliminar redundâncias e torná-lo mais simples e legível é outra prática que foi bem aceita pelos entrevistados. 50% dos entrevistados afirmou já ter utilizado em algum projeto, sendo 44% deles em até 3 projetos. Deste total, 38% classificou seu conhecimento de XP como muito pouco ou pouco. Os 50% restantes, que não chegaram a utilizá-la, afirmaram acreditar que ela traria benefícios.

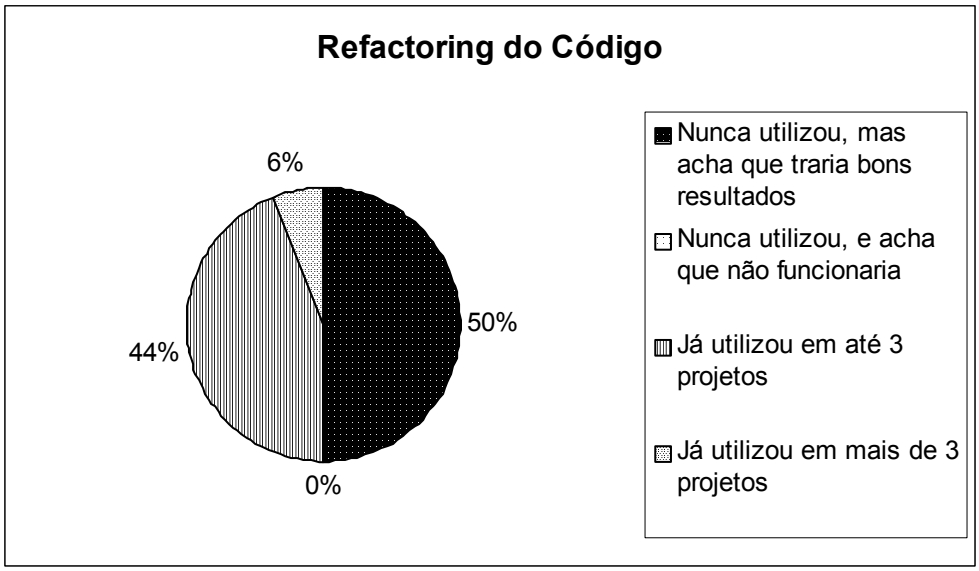


Figura 6-16 - Uso de *refactoring* do código

Como vantagens do uso do *refactoring*, foram citadas a melhor legibilidade e simplicidade do código, o que facilita a manutenção do mesmo. Entretanto, entre os que não adotaram esta prática, existe uma preocupação com o esforço gasto nesta atividade. De fato, corre-se o risco de se ficar constantemente buscando maneiras de se melhorar o código, deixando outras atividades em segundo plano. Cabe aos desenvolvedores discernimento para saber até que ponto esta atividade realmente está agregando valor ao projeto.

Propriedade Coletiva do Código

A Figura 6-17 mostra os resultados do uso da propriedade coletiva, ou seja, o fato de qualquer desenvolvedor estar apto a mexer em qualquer parte do código, sem existir a especialização dos desenvolvedores em partes ou módulos específicos.

A maioria, ou seja, 56% dos entrevistados, afirmou já ter utilizado esta prática em pelo menos um projeto. Deste total, 44% classificou seu conhecimento em XP como muito pouco ou pouco, mostrando que esta prática, apesar de ser bastante associada a XP, é adotada pelos desenvolvedores independente do conhecimento desta metodologia

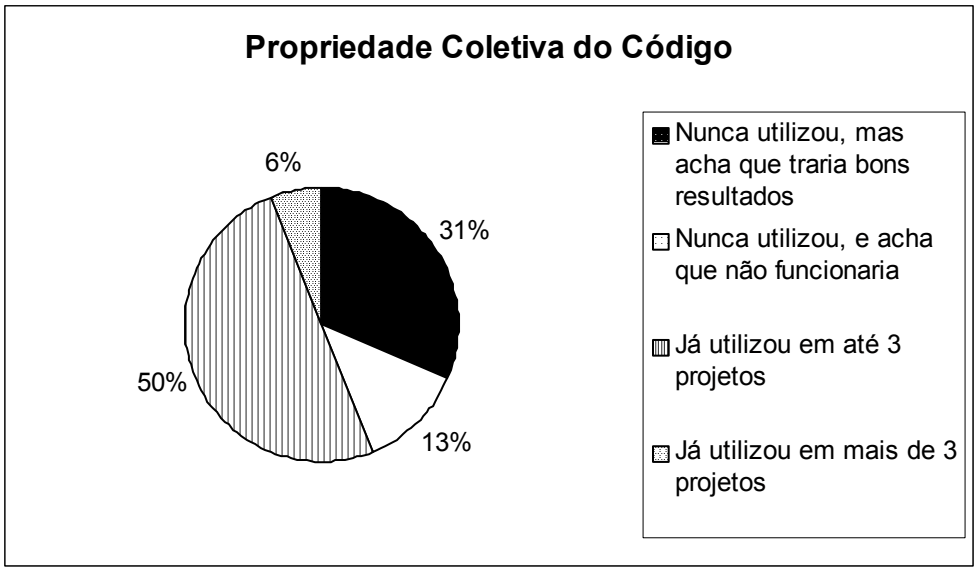


Figura 6-17 - Adoção de propriedade coletiva do código

Entre os que não utilizaram a prática, 31% afirmou acreditar que ela traz benefícios, enquanto 13% julgou que ela não funcionaria. Como benefício da propriedade coletiva do código, foi destacada a agilidade na correção de erros, já que mais de uma pessoa é capaz efetuar esta tarefa. Os principais problemas apontados são o tempo gasto para o entendimento do código, a integridade das alterações com outras partes do código e a preocupação com os erros decorrentes de várias pessoas mexendo em um mesmo trecho de código.

Os dois primeiros problemas são resolvidos com a adoção do padrão de codificação, do *refactoring* do código e da programação em pares. Uma vez que todos os desenvolvedores obedecem ao padrão definido e a programação em pares está sendo usada corretamente, com a troca constante das duplas de desenvolvedores, o conhecimento sobre o código será disseminado entre todos os membros. Além disto, como o código deve ser mantido simples e existe uma bateria de testes automatizados que pode ser executada a qualquer momento, os erros nas alterações podem ser prontamente identificados.

O terceiro problema, que trata da atualização simultânea de um artefato, pode ser resolvido através do uso de ferramentas de controle de versão e é tratado na disciplina Gerência de Configuração e Mudanças, estando fora do escopo deste trabalho.

6.2.4 Artefatos do RUP impactados

Nesta seção, são analisados os artefatos originais do RUP que foram excluídos ou modificados na definição do RUP-pe. Para cada artefato, o entrevistado opinou sobre sua utilidade, julgando se ele seria adequado para todos ou somente alguns tipos de projeto, ou

se não teria utilidade. Estas perguntas estavam presentes em ambos os perfis de questionários.

Plano de Resolução de Problemas

Este artefato, que foi eliminado no RUP-pe, não foi utilizado pela maioria dos entrevistados, como mostra a Figura 6-18. 59% deles afirmaram nunca o ter utilizado, dos quais 50% julga que ele seria útil. Dentre os que afirmaram já ter utilizado, 36% considerou o Plano de Resolução de Problemas um artefato útil na maioria dos projetos.

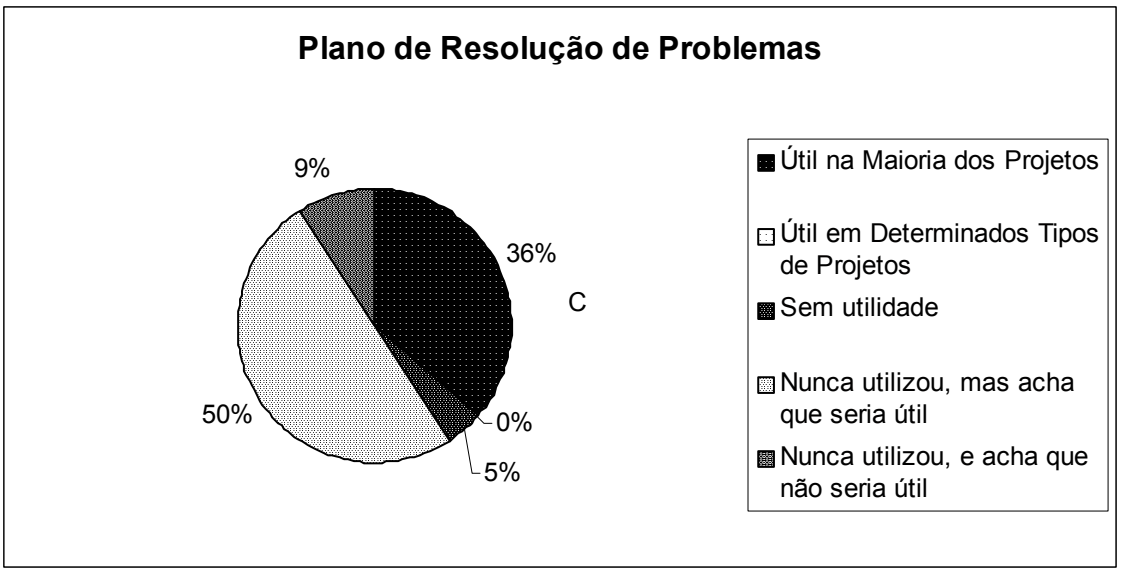


Figura 6-18 - Utilidade do Plano de Resolução de Problemas

O Plano de Resolução de Problemas foi descartado no RUP-pe, pois o processo para tratar os problemas e exceções já está embutido na própria metodologia, utilizando os encontros diários com toda a equipe. Caso haja algum procedimento especial para o tratamento de problemas, isto pode ficar documentado do Plano de Desenvolvimento de Software, como sugerido originalmente pelo próprio RUP.

Plano de Gerenciamento de Riscos

Já utilizado por metade dos entrevistados, o Plano de Gerenciamento de Riscos foi considerado bastante útil. Entre os 50% que não utilizaram-no, somente 9% afirmou achar que ele não seria útil. Entre os 50% que afirmaram ter utilizado, 41% consideraram este artefato útil na maioria dos projetos.

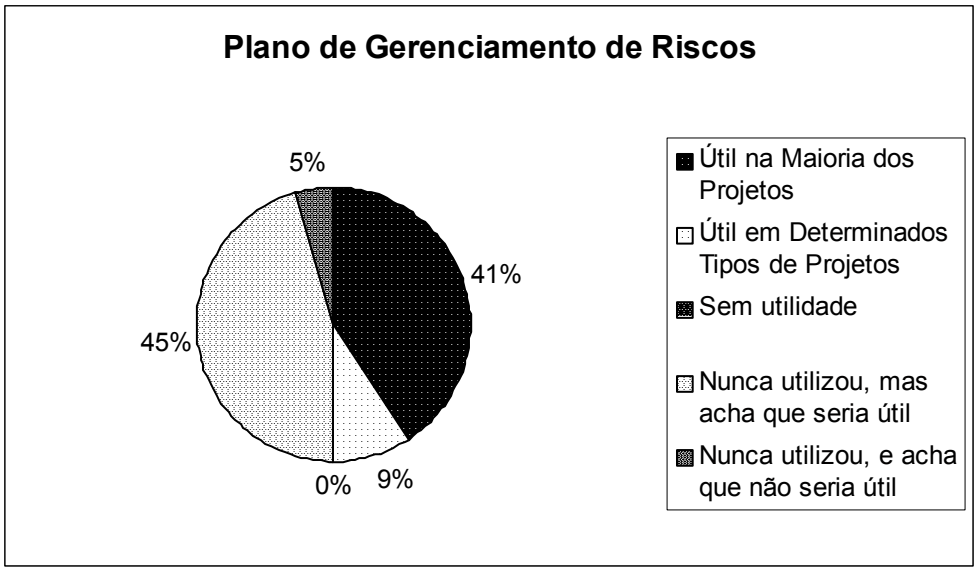


Figura 6-19 - Utilidade do Plano de Gerenciamento de Riscos

O gerenciamento dos riscos é, sem dúvida, uma parte essencial de um bom gerenciamento de projeto. No RUP-pe, os riscos são tratados constantemente, nos encontros diários, na priorização dos casos de uso, durante o planejamento da iteração e nas avaliações de iteração. Os resultados aqui obtidos podem ser resultado da pouca experiência dos entrevistados no processo de desenvolvimento, que leva a uma concepção equivocada da função do artefato.

Para projetos com pequenas equipes, acreditamos que a Lista de Riscos é suficiente para o gerenciamento dos mesmos. Quaisquer informações adicionais sobre o gerenciamento de riscos pode ser documentada no Plano de Desenvolvimento de Software, não sendo necessária a produção de mais um artefato. O próprio RUP, ao falar da adaptação do Plano de Gerenciamento de Riscos, recomenda que "a Lista de Riscos pode ser suficiente para projetos de pequeno porte". O guia de uso do RUP para pequenos projetos, incluído no RUP 2002, também não recomenda a produção deste artefato.

Ordem de Trabalho

A Ordem de Trabalho é praticamente considerada obrigatória pelos entrevistados, como podemos observar na Figura 6-20. A imensa maioria, isto é, 91% dos entrevistados, afirmou que este artefato é útil na maioria dos projetos. Os 9% restantes afirmaram não ter utilizado-o, mas julgavam o artefato igualmente útil.

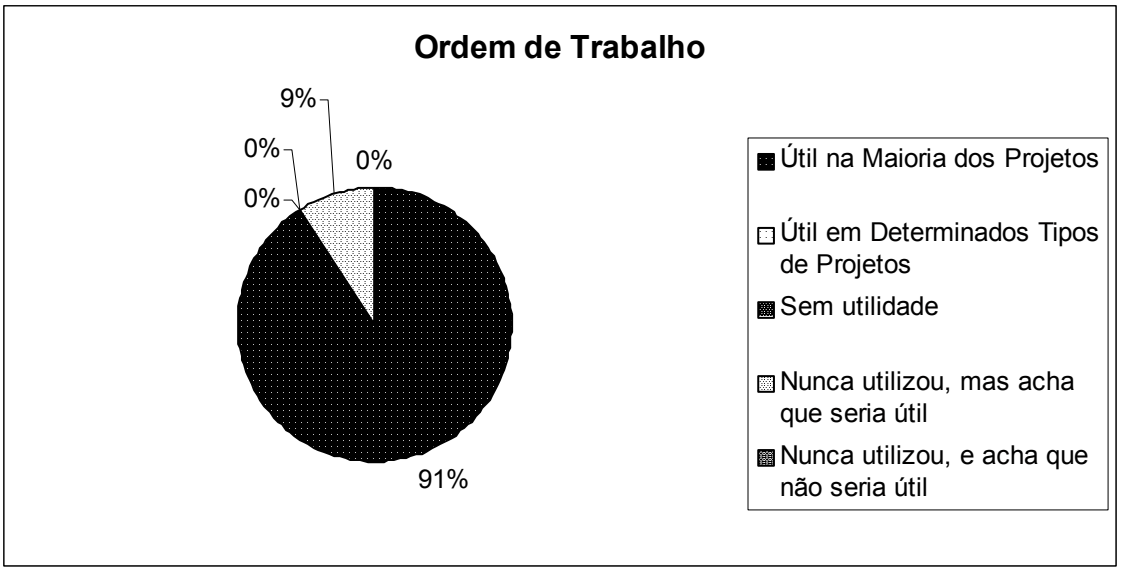


Figura 6-20 - Utilidade da Ordem de Trabalho

No RUP-pe, este artefato é produzido durante o planejamento a iteração, já que toda a equipe deve participar desta atividade. Neste momento, cada desenvolvedor escolhe as tarefas que deseja realizar e estima o esforço necessário para concluí-la. Este planejamento foi detalhado na seção 5.3. O conteúdo do artefato foi simplificado em relação ao RUP original, de forma que não seja custoso produzi-lo ou atualizá-lo. Acreditamos que, uma vez que todos estão presentes no planejamento da iteração, não é necessário que o conteúdo deste artefato seja complexo, detalhado ou bastante elaborado.

Avaliação da Iteração

A Avaliação da Iteração é um artefato considerado útil por quase todos os desenvolvedores. Entre os 77% que o utilizaram, nenhum entrevistado considerou-o sem utilidade. Já entre osque não utilizaram este artefato, 18% afirmou que ele seria útil, enquanto apenas 5% julgou que ele não seria útil. Estes resultados são mostrados na Figura 6-21.

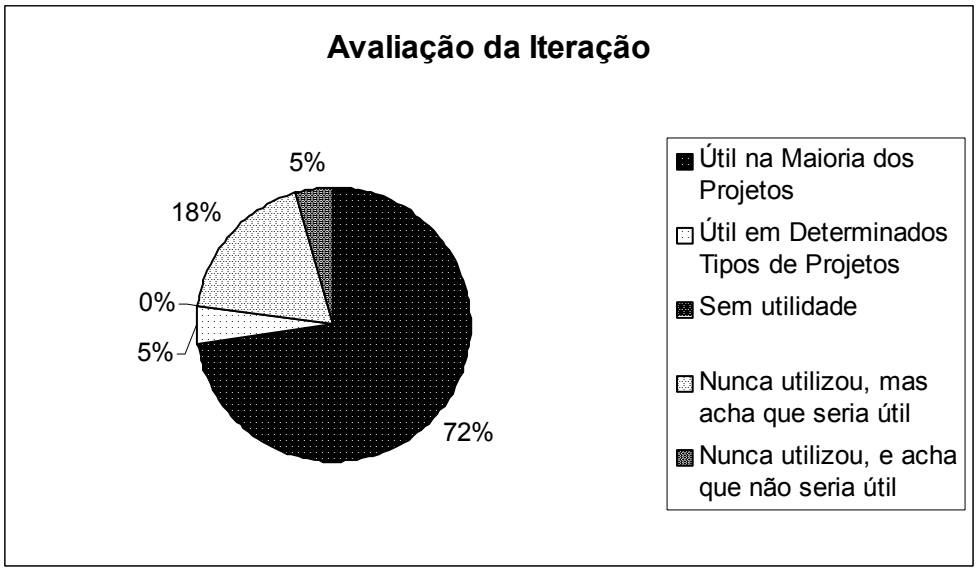


Figura 6-21 - Utilidade da Avaliação da Iteração

De fato, um dos fundamentos do processo iterativo é avaliar constantemente os resultados obtidos como forma de evitar os mesmo erros nas iterações seguintes. A Avaliação da Iteração desempenha um papel fundamental nesta análise, pois documenta justamente a situação do projeto e os problemas e erros encontrados.

No RUP-pe, como forma de se diminuir a quantidade de artefatos, este artefato foi unido à Avaliação de Status. Desta forma, o relatório apresentado aos níveis gerenciais superiores é o resultado da consolidação das avaliações de iterações realizadas. Pelo fato das iterações terem uma duração curta, produzir uma nova Avaliação da Iteração a cada 2 ou 4 semanas traz uma sobrecarga desnecessária ao processo. Por isso, o resultado da atividade Avaliar Iteração, no RUP-pe, fica documentado na Avaliação de Status, não sendo necessário produzir o artefato Avaliação da Iteração.

6.3 Considerações Finais

Neste capítulo apresentamos o resultado de um levantamento sobre as principais mudanças propostas pelo RUP-pe, feito com desenvolvedores de diversos perfis e experiências. Este levantamento não pode ser considerado uma análise precisa da eficácia da metodologia, mas nos dá uma idéia da viabilidade das modificações propostas. O grupo de pessoas entrevistadas é bastante diverso, o que possibilita uma análise do uso das práticas a partir de diferentes pontos de vista.

Grande parte das práticas incluídas na disciplina Gerenciamento de Projeto já vem sendo utilizada pelos desenvolvedores, com benefícios. Os que não as utilizaram, em geral, julgam que estas práticas seriam benéficas. Entre as práticas incorporadas à disciplina Implementação, todas têm sido bem utilizadas. Umas, em maior escala como, por exemplo,

a adoção de um padrão de codificação, a integração contínua e o desenvolvimento de testes unitários automatizados. Outras, como a propriedade coletiva do código, ainda apresentam uma certa resistência, muito provavelmente devido à sua má interpretação ou falta de conhecimento por parte dos desenvolvedores. Entre os artefatos excluídos, todos foram julgados úteis. Entretanto, os entrevistados não possuíam conhecimento sobre a abordagem do RUP-pe, que levou à exclusão destes artefatos. Infelizmente, a validade desta abordagem só pode ser determinada através do uso do RUP-pe em um projeto real.

Os resultados obtidos apontam que as práticas incorporadas ao RUP-pe são viáveis e realmente trazem benefícios ao desenvolvimento em equipes com até 15 desenvolvedores. A grande maioria das práticas já tem sido aplicada por diversas equipes, mesmo sem um grande conhecimento prévio de XP, o que mostra que têm sido utilizadas por sua eficácia e não somente pelo fato de estarem em evidência na comunidade de desenvolvimento de software. Isto leva a crer que, ao se implantar o RUP-pe em um projeto real, os resultados serão bastante positivos.

Capítulo 7: Conclusões

7.1 Principais Contribuições

Neste trabalho, apresentamos uma metodologia de desenvolvimento voltada para pequenos projetos, com equipes de até 15 desenvolvedores. Projetos deste tipo têm mostrado maiores chances de obter sucesso.

O RUP-pe foi desenvolvido com base no RUP e no TSP, além de incorporar práticas de metodologias ágeis como XP e Scrum. O RUP, já bastante consolidado na comunidade de desenvolvimento de software, provê a adoção de uma série de boas práticas e uma excelente maneira de se estruturar e descrever as atividades do processo de desenvolvimento. Entretanto, para o uso em pequenas equipes, deve sofrer adaptações, por ser geral demais.

Através do alinhamento com o TSP, obtemos, como conseqüência, um alinhamento com o SW-CMM, modelo de qualidade de processo mundialmente reconhecido. Este alinhamento traz como vantagens a qualidade do produto final e um diferencial competitivo, caso a empresa decida obter um reconhecimento formal em relação ao SW-CMM. Como o processo já está alinhado, desde sua definição, com o SW-CMM, o impacto no processo de mudanças decorrentes da implantação do modelo será menor, reduzindo o custo e o tempo desta atividade.

Por fim, a incorporação de práticas das Metodologias Ágeis para a simplificação da execução de uma série de atividades contribui para deixar a metodologia enxuta, facilitando seu aprendizado e sua implantação. Estas práticas, como já citado, têm contribuído de maneira positiva para o desenvolvimento de software.

Unindo-se estes três elementos, obtemos uma metodologia que possui somente a complexidade suficiente para não comprometer a qualidade do processo, e conseqüentemente do produto final, sem torná-la burocrática e de difícil uso.

7.2 Dificuldades Encontradas

Nas seções a seguir, são apresentados as principais dificuldades encontradas na realização deste trabalho e seus respectivos impactos no resultado final.

7.2.1 Complexidade do Processo de Software

A definição de uma metodologia é uma tarefa extremamente complexa. Para que a metodologia seja realmente efetiva e contribua para o sucesso do projeto, ela deve ser adequada às necessidades deste. A escolha das atividades a serem executadas e a forma de apresentação dos artefatos resultantes destas atividades deve levar em conta as várias características do projeto em questão. Entre estas características, podemos citar o tamanho, a experiência e a distribuição geográfica da equipe, e a complexidade e criticidade do projeto.

Inicialmente, o objetivo deste trabalho era definir uma metodologia de desenvolvimento completa, ou seja, todas as atividades das disciplinas originais do RUP seriam analisadas em relação ao cenário das equipes pequenas. Entretanto, isto se mostrou inviável devido à necessidade de um profundo conhecimento de cada disciplina, para discernir o que seria impactado ou não pelo fato da equipe do projeto ser pequena.

Em virtude disso, foram definidas somente duas disciplinas: Implementação e Gerenciamento de projetos, para servirem de ponto de partida para a metodologia. A justificativa desta escolha foi mostrada no Capítulo 5. Todavia, para as duas disciplinas escolhidas, as respectivas atividades foram completamente detalhadas, sendo analisados inclusive os passos de cada atividade original do RUP. Desta forma, o RUP-pe oferece todo o suporte para que possa ser adotado mais facilmente pelas equipes que desejarem utilizá-lo.

7.2.2 Bibliografia Escassa Sobre o TSP

Pelo fato do TSP ser recente, sua bibliografia ainda é relativamente escassa. Além dos artigos técnicos do próprio SEI, existe somente um livro-texto [48] e alguns poucos artigos publicados em revistas especializadas, como a IEEE Software e o CrossTalk Journal. A relação entre o TSP e o CMM só foi apresentada pelo SEI em junho de 2002 [56], enquanto o CrossTalk Journal, por exemplo, dedicou uma edição especial ao TSP

somente em setembro de 2002 [73]. Isto dificultou a decisão sobre a adoção do TSP na definição da metodologia, pois, apesar de ter propósitos adequados às pequenas equipes, tais como a formação de uma equipe realmente comprometida, não havia indícios de sua eficácia.

Além disto, a única fonte que apresenta em detalhes a estrutura e as atividades de cada fase do TSP, o livro *Introduction to the Team Software Process* [48], é uma versão simplificada do processo, voltada para o ensino do mesmo em universidades. Segundo o próprio livro:

> "TSPi (o Team Software Process introdutório) introduz conceitos de equipe e guia você nos passos necessários para a formação destas equipes e no modo como trabalhar nelas. Note, contudo, que este texto é voltado para um curso introdutório e não cobre todo o material que você precisará para utilizar o TSP em projetos de larga escala industrial."
>
> (*Introduction to the Team Software Process,* p. xi, tradução nossa)

Foram tentados, em diferentes momentos, vários contatos por correio eletrônico com o SEI, em busca de maiores informações sobre o TSP oficial, sem nenhum tipo de resposta. Aparentemente, a única forma de se conhecer o TSP "industrial" é através de cursos do próprio SEI, o que seria obviamente inviável para a elaboração deste trabalho, já que seria necessário um alto investimento para a realização deste curso, além do deslocamento para o SEI.

Dessa forma, adotamos a versão do livro introdutório, que é bastante detalhada e apresenta todos os *scripts* que guiam as atividades desta versão do processo. Por ter como objetivo o ensino, acreditamos que esta versão contenha os pontos principais do processo, e que a versão industrial não seja muito diferente da aqui apresentada.

7.2.3 Aplicação em um Projeto Real

Outra grande dificuldade deste trabalho foi conseguir aplicar o RUP-pe a um projeto real, para validar nossa proposta. Isto foi difícil devido à pequena margem de manobra que as empresas possuem atualmente em seus projetos. Não há como garantir que a implantação de uma nova metodologia não irá afetar os resultados do projeto, pois, mesmo que as atividades descritas estejam corretas, existe a curva de aprendizagem por parte da equipe, que irá mudar sua cultura de trabalho. Hoje em dia, maioria das empresas não está disposta a correr este risco, caso a iniciativa da definição e implantação da metodologia não seja uma iniciativa própria.

Outro obstáculo, já comentado no Capítulo 6, é que seria necessário acompanhar um projeto desde o início até seu encerramento, para que fossem verificadas todas as

atividades de Gerenciamento de Projeto e se o projeto analisado iria ou não ser concluído com sucesso. Isto demandaria alguns meses para a realização do estudo de caso, o que acabou inviabilizando esta abordagem.

7.3 Trabalhos Relacionados

Entre as metodologias de desenvolvimento voltadas para pequenas equipes, as que mais têm ganhado destaque são as Metodologias Ágeis, já discutidas no Capítulo 3. Estas metodologias vêm, cada vez mais, sendo utilizadas em projetos de pequeno porte. Todavia, como já foi discutido anteriormente, estas metodologias apresentam algumas desvantagens:

- Por serem apresentadas de forma simples, muitas delas são descritas superficialmente, o que pode ser um problema para sua adoção em uma equipe que seja inexperiente. Scrum, por exemplo, considera que a fase de desenvolvimento é um processo "caixa-preta", onde não interessa que técnica a equipe utilizará para produzir o sistema. Já XP não deixa claro como os riscos devem ser tratados pela equipe;
- Não existe uma preocupação com o alinhamento em relação a um modelo de qualidade. Em relação ao SW-CMM, especificamente, há trabalhos comparando somente XP [26] e estes trabalhos têm sido, inclusive, questionados pela comunidade [74].

Com a crescente popularização das metodologias ágeis, algumas metodologias "pesadas" começaram a divulgar versões "leves", para serem utilizadas em projetos de pequeno porte. Pollice, por exemplo, afirma em [27] e [28] que o RUP pode incorporar as práticas de XP em um cenário em que a equipe é pequena, está no mesmo ambiente físico e pode contar com a presença do cliente no local. Esta descrição, entretanto, não entra em um nível de detalhes que possibilite o seu uso, pois não são apresentadas que atividades não são necessárias ou que podem ser executadas de maneira mais simples.

A nova versão do RUP [16] apresenta dois guias: um de utilização de práticas "ágeis" e outro de como utilizar o RUP para pequenos projetos, uma incorporação do trabalho previamente descrito por Pollice em [75]. No primeiro guia, são apresentadas as 12 práticas de XP e analisadas aquelas que podem ser incorporadas ao RUP e quais exigem cautela quando o tamanho do projeto começa a crescer. Além disto, é apresentado um mapeamento entre os artefatos e papéis de XP e do RUP. Todavia, mais uma vez não é

explicitado que atividades ou artefatos podem ser eliminados, ficando a cargo do desenvolvedor julgar quais as adaptações necessárias.

No segundo guia, é apresentado que artefatos são fundamentais no RUP e quais podem ser excluídos em um projeto de pequeno porte. Entretanto, este guia continua referenciando as atividades originais do RUP, sem deixar claro se estas atividades podem ser realizadas de maneira mais simples ou mesmo juntar duas ou mais atividades em uma só, como fizemos em nossa proposta. Como já apresentado na seção 2.4, acreditamos que este guia não é suficiente para utilizar a metodologia de forma efetiva.

Retting apresenta em [76] o Processo de Planejamento e Desenvolvimento PADRE, um processo de software para pequenas equipes, que incorpora o ciclo PDCA (*Plan-Do-Check-Act*) do movimento de Qualidade Total. O processo é dividido em três níveis:

- Nível de projeto: atividades relacionadas à concepção inicial do projeto. É definida a equipe responsável pelo planejamento e condução do projeto, são definidos os objetivos do projeto, e os recursos e planos para as fases posteriores;
- Nível de estágio: estágios são períodos de tempo, de quatro a seis meses, onde serão implementados um ou mais unidades de trabalho, denominadas módulos. As atividades deste nível tratam do planejamento do trabalho em módulos, definição de recursos e utilização do processo do nível seguinte para desenvolver cada módulo;
- Nível de módulo: neste nível, as atividades do processo dizem que devem ser elaborados projetos detalhados de implementação e de testes, e que os módulos devem ser codificados de acordo com estes projetos.

O ciclo de vida do processo é iterativo, e são utilizadas métricas para acompanhar o progresso do projeto. Além disto, são descritos papéis específicos, que devem ser atribuídos no início do projeto, por exemplo, o Facilitador, responsável por promover as reuniões da equipe, e o Implementador, que irá escrever o código que implementa o projeto descrito. Entretanto, o PADRE não entra em maiores detalhes sobre a execução das atividades, o que pode dificultar sua adoção.

Larman propõe em [17] uma versão "ágil" do RUP, enfatizando que o RUP não deve ser seguido à risca para a maioria dos projetos. São incorporados vários princípios

apresentados aqui, como iterações de curta duração e tamanho fixo, *test-first programming*, integração contínua, entre outros.

Como o principal foco do seu trabalho é a disciplina Análise e Projeto, Larman não entra em muitos detalhes sobre as atividades de outras disciplinas. São tratados em capítulos especiais apenas alguns tópicos de descrições de casos de uso e de gerenciamento de projeto. Além disto, não há nenhuma preocupação de que seu processo esteja alinhado com um modelo de qualidade. Todavia, a abordagem de Larman para as atividades de análise e projeto serve como um bom complemento ao RUP-pe que, atualmente, não tem esta disciplina definida.

7.4 Trabalhos Futuros

A definição das disciplinas aqui apresentadas se mostrou uma tarefa bastante complexa, pois foi necessário analisar criteriosamente cada atividade e cada artefato envolvidos. Isto impossibilitou a realização de outras atividades igualmente importantes, que vêm enriquecer nossa proposta. A definição das outras disciplinas do RUP-pe certamente é uma destas atividades. É necessário analisar o impacto do tamanho da equipe nas atividades que não foram tratadas, de modo a se obter uma metodologia completa. Existem hoje propostas "ágeis" para Análise e Projeto [77], e para Gerência de Configuração [78], por exemplo. Estas propostas podem ser usadas como base para a definição das atividades.

Outro ponto importante é a aplicação da metodologia a um projeto real. Sua aplicação prática servirá para validar as práticas incorporadas e modificações propostas. Além disto, permite a identificação de pontos falhos a serem considerados para o refinamento do RUP-pe.

Um outro ponto a ser explorado é a definição de uma metodologia seguindo o caminho inverso ao escolhido neste trabalho, ou seja, partindo-se de uma metodologia ágil, por exemplo, XP, e adicionando-se atividades e artefatos necessários para se obter o alinhamento com o TSP. Com isso, podemos investigar se é possível obter uma metodologia que seja mais simples do que o RUP-pe.

Por fim, para melhorar o suporte aos interessados em utilizar a metodologia, devem ser incorporados ao *website* guias sobre tópicos específicos, por exemplo, programação em pares e *test-first development*, como forma de ajuda aos desenvolvedores que não conhecem estas práticas, mas têm interesse em aplicá-las

Referências

[1] BOEHM, B. BASILI, V. R. ***Gaining Intelectual Control of Software Development***. IEEE Computer, vol. 33, maio de 2000.

[2] JOHNSON, J. H. ***Micro Projects Cause Constant Change***. Disponível em www.xp2001.org/xp2001/conference/ papers/Chapter30-Johnson.pdf, 2001.

[3] THE STANDISH GROUP INTERNATIONAL. ***CHAOS: A Recipe For Sucess***. The Standish Group International Inc, 1999.

[4] LINDVALL, M., RUS, IOANA. ***Process Diversity in Software Development***. IEEE Software, Julho/Agosto de 2000.

[5] COCKBURN, A. ***Selecting a Projects's Methodology***. IEEE Software, Junho/Julho de 2000.

[6] THE OPEN WEBSITE. http://www.open.org.au, último acesso em 22/08/2003.

[7] KRUTCHEN, P. ***The Rational Unified Process***. Addison-Wesley, 1998.

[8] BECK, K. ***Extreme Programming Explained: Embrace Change***. Addison-Wesley, 1999.

[9] SCHWABER, K. ***The Scrum Development Process***. Workshop on Business Object Design and Implementation, ACM Conference Object-oriented Programming Systems, Languages and Applications (OOPSLA'95). 1995.

[10] HUMPHREY, W. ***The Team Software Process (TSP)***. Relatório Técnico. Pittisburgh: Universidade de Carnegie Mellon, 2000.

[11] PAULK, M et al. ***The Capability Maturity Model – Guidelines for Improving the Software Process***. Addison-Wesley, 1995.

[12] WEBSITE DO RUP-PE. http://www.cin.ufpe.br/~jpco/rup-pe, último acesso em 22/08/2003.

[13] JACOBSON, I et al. ***Object-Oriented Software Engineering – A Use Case-Driven Approach***. Addison-Wesley, 1992.

[14] BOOCH, G. ***Object-Oriented Analysis and Design with Applications***. Addison-Wesley, 1994.

[15] RUMBAUGH, J et al. ***Object-Oriented Modeling and Design***. Prentice-Hall, 1991.

[16] RATIONAL SOFTWARE CORPORATION. ***Rational Unified Process 2002***. 2002.

[17] LARMAN, C. ***Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and the Unified Process***. Prentice Hall, 2001.

[18] BOOCH, G; RUMBAUGH, J; JACOBSON I. ***The Unified Modeling Language User Guide***. Addison-Wesley, 1998.

[19] FOWLER, M. ***The New Methodology***. Disponível em http://martinfowler.com/articles/newMethodology.html. Último acesso em 22/08/2003.

[20] EVANS, G. ***A Simplified Approach to RUP***. The Rational Edge, janeiro de 2001.

[21] AUGUSTINE, L. ***Using the Rational Unified Process (RUP) Sucessfully for Small Development Projects***. The Rational Edge, setembro de 2001.

[22] COELHO, C. ***MAPS: Um Modelo de Adaptação de Processos de Software***. Centro de Informática, Universidade Federal de Pernambuco, dissertação de mestrado. 2003.

[23] AGILE ALLIANCE WEBSITE. http://www.agilealliance.org. Último acesso em 22/08/2003.

[24] ABRAHAMSSON, P; et al. ***Agile Software Development Methos – Review and Analysis***. VTT Publications, disponível em www.agilealliance.org/articles. 2002.

[25] FOWLER, M. ***Is Design Dead?***. Disponível em http://www.martinfowler.com/articles/designDead.html, último acesso em 22/08/2003.

[26] PAULK. M., ***Extreme Programming from a CMM Perspective***. IEEE Software vol.18, número 6. 2001.

[27] POLLICE, G. ***RUP and XP, Part I: Finding Common Ground***. The Rational Edge. Disponível em http://www.therationaledge.com/content/mar_01/f_xp_gp.html. 2001.

[28] POLLICE, G. ***RUP and XP, Part II: Valuing Differences***. The Rational Edge. Disponível em http://www.therationaledge.com/content/apr_01/f_xp2_gp.html. 2001.

[29] SIDDIQI, J. ***eXtreme Programming pros and cons: What questions remain?***. IEEE Computer Society Dynabook, novembro de 2000. Disponível em http://computer.org/seweb/Dynabook/index.htm, último acesso em 22/08/2003.

[30] HUMPHREY, W. ***Comments on Extreme Programming***. IEEE Computer Society Dynabook. Novembro de 2000. Disponível em http://computer.org/seweb/Dynabook/HumphreyCom.htm, último acesso em 22/08/2003..

[31] http://groups.yahoo.com/group/extremeprogramming/, último acesso em 22/08/2003..

[32] http://c2.com/cgi/wiki?ExtremeProgramming, último acesso em 22/08/2003.

[33] JEFFRIES, R. et al. ***Extreme Programming Installed***. Addison-Wesley, 2000.

[34] HIGHSMITH, J. ***Agile Software Development Ecosystems***. Addison-Wesley, 2002.

[35] BECK, K; FOWLER, M. ***Planning Extreme Programming***. Addison-Wesley, 2000.

[36] COAD, P. et al. ***Java Modeling in Color with UML: Enterprise Components and Process***. Prentice Hall, 1999

[37] PALMER, S. ***Coad Letter - Feature Driven Development and Extreme Programming***. Disponível em http://www.togethercommunity.com/coad-letter/Coad-Letter-0070.html, último acesso em 22/08/2003.

[38] Website do DSDM Consortium. http://www.dsdm.org, último acesso em 22/08/2003.

[39] MARTIN, J. ***RAD: Rapid Application Development***. MacMillan Coll Div, 1991.

[40] http://www.surgeworks.com/dsdm/dsdmfaq.htm, último acesso em 22/08/2003.

[41] http://www.surgeworks.com/dsdm/DSDMPrinciples.pdf, último acesso em 22/08/2003.

[42] TUFF, D. et al. ***Inter-operability of DSDM with the Rational Unified Process***. *Whitepaper* do DSDM Consortium, 1999.

[43] CALDOW, D. ***DSDM and Extreme Programming***. *Whitepaper* do DSDM Consortium, 2002.

[44] http://www.crystalmethodologies.org, último acesso em 22/08/2003.

[45] HUMPREY, W. ***A Discipline for Software Engineering***. Addison-Wesley, 1995.

[46] ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ***Sistema de Gestão da Qualidade***. NBR ISO9001:2000, 2000.

[47] ISO – INTERNATIONAL ORGANIZATION FOR STARDARDIZATION. ***Software Process Assessment.*** ISO/IEC TR-15504. 1998.

[48] HUMPHREY, W. ***Introduction to the Team Software Process***. 1. ed. Massachusetts: Addison-Wesley, 1999.

[49] ANDREWS, D. ***The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices***. Relatório Técnico. Pittisburgh: Universidade de Carnegie Mellon, 2000.

[50] GOTH, G. ***The Team Software Process: A Quiet Quality Revolution***. IEEE Software, novembro/dezembro de 2000.

[51] ROYCE, W. ***Software Project Management: A Unified Framework***. Addison-Wesley, 1998.

[52] WEBB, D. ***Managing Risk With TSP***. CrossTalk: The Journal of Defense Software Engineering, junho de 2000.

[53] McHALE, J. ***TSP: Process Costs and Benefits***. CrossTalk: The Journal of Defense Software Engineering, setembro de 2002.

[54] OCA, C. M. De; SERRANO, M. A. ***Managing a Company Using TSP Techniques***. CrossTalk: The Journal of Defense Software Engineering, setembro de 2002.

[55] DAVIS, N. ***Using the TSP to Implement the CMM***. CrossTalk: The Journal of Defense Software Engineering, setembro de 2002

[56] DAVIS, N.; McHALE J. ***Relating the Team Software Process (TSP) to the Capability Maturity Model for Software (SW-CMM)***. Relatório Técnico. Pittisburgh: Universidade de Carnegie Mellon, 2002.

[57] HEFLEY, B; SCHWALB, J.; PRACCHIA L. ***AV-8B's Experience Using the TSP to Accelerate SW-CMM Adoption***. CrossTalk: The Journal of Defense Software Engineering, setembro de 2002.

[58] MUSSON, R. ***How TSP Impacts the Top Line.*** CrossTalk: The Journal of Defense Software Engineering, setembro de 2002.

[59] VU, J.D. ***Process Improvement Journey (From Level 1 to Level 5)***. European Software Engineering Process Group, 2001.

[60] SHINE TECHNOLOGIES. ***Agile Methodologies Survey Results***. Shine Technologies Pty Ltd. 2003.

[61] RUMPE, B., SCHRÖDER, A. ***Quantitative Survey on Extreme Programming Projects***. Disponível em http://www.agilealliance.org/articles. último acesso em 22/08/2003.

[62] FOWLER, M. FOEMMEL, M.; ***Continuous Integration***. Disponível em http://www.martinfowler.com/articles/continuousIntegration.html, último acesso em 22/08/2003.

[63] GNU MAKE. http://www.gnu.org/software/make/make.html, último acesso em 22/08/2003.

[64] APACHE ANT. http://ant.apache.org, último acesso em 22/08/2003.

[65] CRUISE CONTROL. http://cruisecontrol.sourceforge.net, último acesso em 22/08/2003.

[66] WILLIAMS, L., COCKBURN, A. ***The Costs And Benefits of Pair Programming***. In: Extreme Programming Examined. Addison Wesley, 2001.

[67] TOMAYKO, JAMES. ***A Comparison of Pair Programming to Inspections for Software Defect Reduction.*** Computer Science Education Journal, Vol.12, No.3, 2002.

[68] FOWLER, M. ***Refactoring: Improving the Design of Existing Code***. Addison-Wesley. 1999.

[69] EXTREME PROGRAMMING TOOLS. http://www.xprogramming.com/software, último acesso em 22/08/2003.

[70] CAMPÊLO, G. ***A Utilização de Métricas na Gerência de Projetos de Software: Uma Abordagem Focada no CMM Nível 2***. Centro de Informática, Universidade Federal de Pernambuco, dissertação de mestrado. 2002.

[71] BOEHM, B. ***Get Ready for Agile Methods, with Care***. IEEE Computer, janeiro de 2002.

[72] PARKISON, N. ***Parkinson's Law: The Pursuit of Progress***. John Murray. 1958.

[73] CROSSTALK: THE JOURNAL OF DEFENSE SOFTWARE ENGINEERING. Setembro de 2002. Disponível em de http://www.stsc.hill.af.mil/crosstalk/2002/09/websites.html, último acesso em 22/08/2003.

[74] REIFER, D. ***XP and the CMM***. IEEE Software, maio/junho de 2003.

[75] POLLICE, G. ***Using the Rational Unified Process for Small Projects: Expading Upon eXtreme Programming***. *Whitepaper* da Rational Software Corporation. 2001.

[76] RETTING, M.; SIMONS, G. ***A Project Planning and Development Process for Small Teams***. Communications of the ACM, vol. 36, outubro de 1993.

[77] AGILE MODELING WEBSITE. http://www.agilemodeling.com. Último acesso em 22/08/2003.

[78] APPLETON, B. ***Agile Configuration Management Environments***. CM Crossroads News, abril de 2003.

[79] WILLIAMS, L. et al. ***Strengthening the Case for Pair-Programming.*** IEEE Software, julho/agosto de 2000.

[80] WILLIAMS, L. A. ***The Collaborative Software Process***. PhD Thesis, Department of Computer Science, The University of Utah, agosto de 2000.

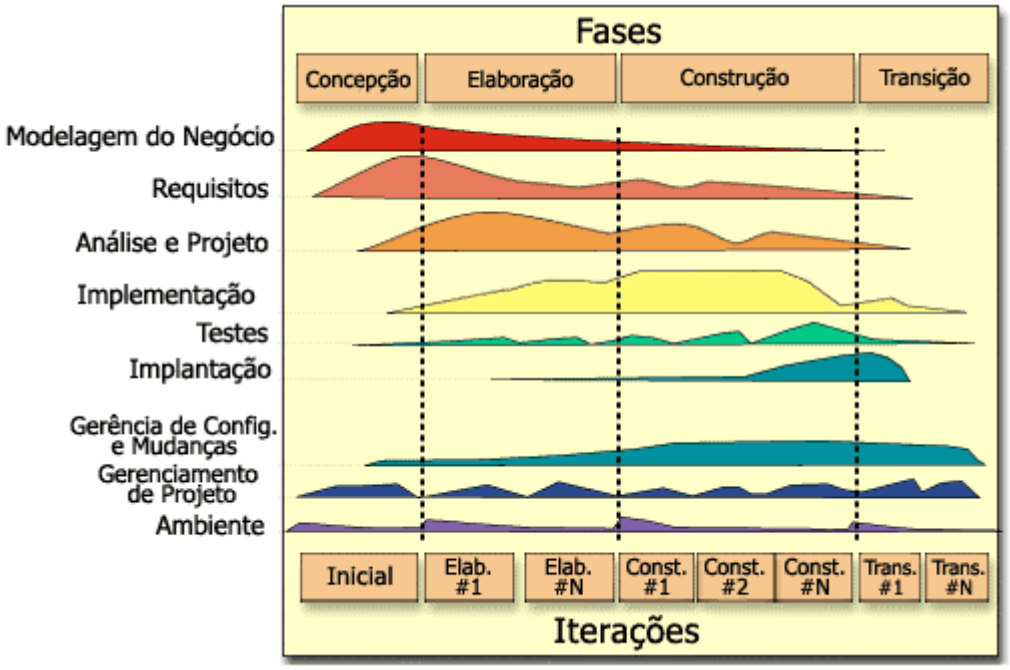
[81] NOSEK, J. T. ***The Case for Collaborative Programming***. Communications of the ACM, vol. 41, março de 1998.

Apêndice A – O Site do RUP-pe

Neste apêndice são apresentados exemplos de páginas do *website* do RUP-pe, de atividades que foram completamente reformuladas a atividades que sofreram apenas pequenas alterações.

Página Inicial do Site

Rational Unified Process para Pequenas Equipes: Visão Geral



Clique em uma área da figura para mais informações.

O RUP-PE, Rational Unified Process para Pequenas Equipes é um *framework* para processos de engenharia de software voltado para times de até 15 desenvolvedores, fruto de uma dissertação de mestrado do Centro de Informática da Universidade Federal de Pernambuco. Seu objetivo principal é ser uma metodologia ágil, para equipes pequenas, mas que esteja alinhada com modelos de qualidade reconhecidos, como o SW-CMM.Ele é o resultado da junção de conceitos e práticas do RUP®, de eXtreme Programming e do Team Software Process (TSP).

Este *site* é um protótipo inicial de um guia do RUP-PE. Aqui são apresentadas as duas disciplinas abordadas na dissertação: **Implementação** e **Gerenciamento de Projeto**. São descritos o fluxo de atividades de cada disciplina, os passos destas atividades, bem como os papéis envolvidos. O ciclo de vida adotado é o mesmo do RUP®, com duas dimensões, mostrado na figura acima:

- o eixo horizontal representa o tempo e mostra aspectos do ciclo de vida do processo, à medida que ele evolui;
- o eixo vertical representa as disciplinas, que agrupam atividades naturalmente relacionadas.

A primeira dimensão represnta o aspecto dinâmico do processo à medida que ele é executado, e é expressa em termos de fases, iterações e marcos. A segunda dimensão representa o aspecto estático do processo: como ele é descrito em termos de componentes do processo, disciplinas, atividades, fluxos, artefatos e papéis (veja Conceitos Chave).
O gráfico mostra como a ênfase em cada disciplina varia no decorrer do tempo. Por exemplo, nas iterações iniciais, gasta-se mais tempo em requisitos, e nas iterações posteriores, mais tempo em implementação.
Neste *site*, a seguinte convenção foi adotada durante a elaboração das páginas:

- Páginas em português indicam que a atividade/artefato original do RUP® foi completamente reformulada;
- Páginas em inglês com trechos destacados indicam alterações pontuais na atividade/artefato original do RUP® ;
- Páginas em inglês com o aviso "Atenção: esta atividade não necessitou de alteração!", ou sem nenhuma indicação, são atividades do RUP® que não necessitaram de adequação para pequenas equipes, sendo mantidas as originais.

Este trabalho está em constante evolução. Dúvidas, críticas e sugestões são muito bem-vindos. Para isto, basta clicar aqui e mandar um e-mail para o autor.

Atividade: Desenvolver Plano de Iteração

Atividade: Desenvolver Plano de Iteração

Propósito
Desenvolver um plano detalhado para uma iteração, contendo:

- Os casos de uso e requisitos não-funcionais que serão trabalhados
- As atividades necessárias para desenvolver estes casos de uso e requisitos não-funcionais
- O cronograma das atividades e seus respectivos responsáveis

Passos

- Determinar o Escopo da Iteração
- Definir Tarefas da Iteração
- Estimar Tarefas e Dividir Responsabilidades

Artefatos de Entrada:	**Artefatos de Saída:**
▪ Plano de Desenvolvimento de Software ▪ Caso de Desenvolvimento (*Development Case*) ▪ Lista de Riscos ▪ Lista de Pendências ▪ Avaliação de Status ▪ Visão ▪ Documento de Arquitetura de Software	▪ Plano de Iteração

Freqüência: A cada iteração

Papel: Gerente de Projeto

Subfluxo:

- Gerenciamento de Projeto
  - Planejar Próxima Iteração

A iteração é um conjunto de tarefas que devem ser executadas em um **período de tempo fixo (entre duas e quatro semanas)**, focado em produzir um executável. Para todas as iterações, exceto a última da fase de transição, este executável é um produto que contém apenas um subconjunto das funcionalidades requeridas. O foco na produção de um executável força a integração contínua do produto e permite que o projeto ataque cedo os riscos técnicos, diminuindo assim outros riscos associados a eles.

Trabalhar iterativamente implica uma certa quantidade de retrabalho dos artefatos existentes e, juntamente com isso, uma mudança de atitude em relação ao retrabalho. Uma certa dose de retrabalho é necessária para entregar um produto de qualidade:

construindo produtos intermediários e avaliando a adequação da arquitetura do produto **cedo e freqüentemente**, aumenta-se a qualidade do produto final e mudanças são menos custosas e mais fáceis de implementar.

Determinar o Escopo da Iteração

Propósito
▪ Selecionar um conjunto de casos de uso ou de cenários para serem trabalhados durante a iteração. ▪ Selecionar um conjunto de requisitos não-funcionais para serem tratados durante a iteração.
Guias:
▪ O Plano de Iteração ▪ Determinar o Escopo de Acordo com a Fase

A equipe, em conjunto com representantes do cliente ou especialistas do negócio, deve escolher os casos de uso e requisitos não-funcionais que serão trabalhados durante a iteração. Esta escolha deve ser guiada por cinco fatores:

- **Riscos do projeto:** um dos objetivos do desenvolvimento iterativo é diminuir os riscos do projeto. Isto é feito atacando-se os riscos o mais cedo possível, pois, quanto mais cedo os riscos forem eliminados, maiores as chances do sucesso. Por isso, é importante escolher casos de uso cujo desenvolvimento force a equipe a enfrentar os maiores riscos existentes.

- **Valor para o negócio:** além dos riscos, é extremamente importante levar em consideração que funcionalidades do sistema oferecem maior valor para o usuário. Este fator muitas vezes conflita com o fator Riscos do projeto, pois o cliente priorizará os casos de uso que mais agregam valor ao negócio, e a equipe do projeto, os que apresentam maior risco técnico. É necessário chegar a um meio-termo entre estes fatores. A equipe tem obrigação de manter o cliente ciente dos eventuais riscos que está assumindo, caso decida adiá-los em detrimento de funcionalidades menos arriscadas, mas deve aceitar a escolha feita por ele. Satisfazer as necessidades do cliente deve ser o objetivo principal.

- **Velocidade da equipe:** a quantidade de casos de uso a serem desenvolvidos em uma iteração depende da velocidade da equipe, ou seja, quantos casos de uso e de que complexidade a equipe é capaz de desenvolver em uma iteração. É importante saber qual a real capacidade de desenvolvimento, para que não sejam assumidos compromissos que não serão cumpridos. As medições possuem um papel fundamental neste ponto. É extremamente importante acompanhar quanto a equipe consegue produzir, em média, por iteração, e não se comprometer com mais trabalho do que se consegue realizar.

- **Cobertura dos casos de uso**: outro fator que deve ser levado em conta, principalmente nas iterações iniciais, é a cobertura do sistema, ou seja,

desenvolver casos de uso que lidem com as mais variadas partes do sistema possível, ainda que em cenários simples. Isto é importante para se ter uma idéia do todo e descobrir eventuais riscos que não tenham sido levantandos ou novas necessidades que não tenham sido descobertas na disciplina de Requisitos..

- **Os objetivos da fase atual**: dependendo da fase em que o projeto se encontra, alguns objetivos devem ser cumpridos. Estes objetivos também devem ser levados em conta na escolha dos casos de uso. Na fase de Elaboração, por exemplo, deve-se procurar estabelecer uma arquitetura robusta para o sistema. Para maiores detalhes, veja o Guia: Definindo o Escopo de Acordo com a Fase

Definir Atividades da Iteração

Com base nos objetivos da iteração, deve-se determinar que atividades devem ser executadas para cumpri-los. Tipicamente, cada iteração irá passar, ainda que parcialmente, por todas as atividades do processo de desenvolvimento, por exemplo:

- Casos de uso e cenários que representam a funcionalidade requerida são escolhidos;
- O comportamento do caso de uso (ou cenário) é pesquisado e documentado;
- O comportamento é analisado e alocado entre os subsistemas e classes que provêem o comportamento necessário;
- As classes e subsistemas são projetados, implementados e testados unitariamente;
- O sistema é integrado e testado como um todo;
- Para *releases* externo, o produto é preparado e é feita sua transição para o ambiente do usuário.

O grau em que estas atividades são executadas varia conforme a iteração e a fase do projeto. Cada uma das disciplinas (Requisitos, Análise e Projeto, Testes, etc.) define atividades genéricas, que devem ser adaptadas para a organização durante a configuração do processo.

Identificar Atividades Envolvidas e Artefatos Afetados

Após ter escolhido os cenários ou casos de uso (mais os defeitos a serem consertados), o Gerente de Projeto precisa determinar que atividades serão necessárias e que artefatos serão afetados:

- Que classes precisam ser revistas?
- Que subsistemas serão afetados ou criados?
- Que interfaces provavelmente serão modificadas?
- Que documentos devem ser atualizados?

Para fazer isto, toda a equipe deve estar reunida e realizar um *brainstorming* das atividades necessárias para desenvolver cada caso de uso ou cenário escolhido.A participação de todos no planejamento é importante para garantir que nenhuma atividade será esquecida, já que quem realmente as realiza está presente, e para aumentar a motivação da equipe e o comprometimento com o plano, pois todos participam ativamente de sua construção e têm direito a expressar sua opinião.

As atividades devem ser determinadas com base no processo utilizado pela equipe. Caso seja detectado que uma atividade necessária não está no processo, deve-se registrar uma Requisição de Mudança para a melhoria do processo, que será avaliada na Atividade: Avaliar Iteração.
Algumas atividades, como esta, são realizadas uma vez a cada iteração. Outras, como Modelar Interface com o Usuário, podem ser realizadas para cada caso de uso, ou cada subsistema. Se houver uma atividade muito abrangente, ou seja, que irá envolver um grande número de passos ou de artefatos, ela pode ser dividida em atividades menores. Isto facilita tanto as estimativas quanto a divisão de responsabilidades.

Além das atividades de desenvolvimento dos casos de uso, podem haver outras que não estão relacionadas diretamente a um caso de uso específico, como "configurar o servidor de aplicação" ou "atualizar o banco de dados". Elas também devem ser levadas em conta no planejamento.

Estimar Tarefas e Dividir Responsabilidades

Uma vez definido o conjunto de atividades da iteração, é preciso estimar a duração de cada uma e dividi-las entre a equipe do projeto. Estes dois passos são feitos em paralelo. Cada membro da equipe escolhe uma tarefa que deseja realizar e diz quanto tempo levará para completá-la. Isto é repetido até que todos os membros da equipe estejam com o máximo de trabalho que possam realizar ou até que não haja mais tarefas.

No primeiro caso, onde nota-se não haverá tempo para realizar todas as tarefas, deve-se reduzir o escopo da iteração em vez de estender sua duração, pois as iterações devem ter a duração fixa (*timeboxing*), ditando o ritmo do desenvolvimento. Depedendo da fase em que o projeto se encontra, torne os cenários mais simples ou elimine algumas funcionalidades. No segundo caso, menos provável de acontecer, deve-se incluir mais casos de uso na iteração atual, retornando-se ao primeiro passo desta atividade.

Ao final da divisão das atividades, deve-se verificar se não há ninguém da equipe sobrecarregado. Caso haja, deve-se remanejar as tarefas para desenvolvedores menos ocupados. Atividades que não foram escolhidas por ninguém também devem ser distribuídas entre as pessoas que estão mais livres. De preferência, elas devem escolher, dentre as que restaram, as que preferem realizar.

Atividade: Desenvolver Plano de Medições

Activity: Develop Measurement Plan

Purpose

- To define management goals, in terms of quality, progress, and improvement
- To determine what needs to be measured periodically to support these goals

Steps

- Define the Primary Management Goals
- Validate the Goals
- Define the Subgoals
- Identify the Metrics Required to Satisfy the Subgoals
- Identify Project Indicators
- Identify the Primitive Metrics Needed to Compute the Metrics
- Write the Measurement Plan
- Evaluate the Measurement Plan
- Put in Place the Collection Mechanisms

Input Artifacts:	**Resulting Artifacts:**
▪ Risk List ▪ Business Case ▪ Vision	▪ Project Measurements ▪ Measurement Plan

Frequency: Once per project (updated with each iteration, if necessary)

Role: Project Manager

Workflow Details:

- Project Management
  - Develop Software Development Plan

The Measurement Plan describes the goals which the project must track towards for successful completion and the measures and metrics to be used to determine whether the project is on track.

The activity Develop Measurement Plan is done once per project, in the inception phase, as part of the general planning activity. The measurement plan may be revised, like any other section of the software development plan, during the course of the project.

Define the Primary Management Goals

Purpose

- To determine and record the important functional, non-functional, budgetary and schedule requirements and constraints, which need to be tracked.

The Project Manager should decide which of the project's requirements and constraints are important enough to require an objective monitoring program. Additionally, organizational requirements may be imposed that are related to business needs (cost reduction, time-to-market and productivity improvements), not directly to project needs.

Typically, a project manager will want to track the growth in capability and reliability of the software under construction, as well as expenditures (effort, schedule, other resources), and there may be performance and other quality requirements, as well as memory and processor constraints. See Guidelines: Metrics for more details. The sources of information for selection of goals include the Vision, Risk List and Business Case, as well as organizational requirements and constraints not specified in the Rational Unified Process.

Validate the Goals

Purpose

- To review the relevance, clarity, feasibility and sufficiency of the selected goals

The Project Manager should review the selected goals with relevant stakeholders to ensure that the focus of the goals selected is correct, that there is adequate coverage of all areas of interest and risk, that it is possible to reduce the goals to collectible metrics and that adequate resources can be committed to the measurement program.

Define the Subgoals

Purpose

- Analyze complex goals to determine subgoals to which metrics can be applied

It may be difficult or impossible to formulate direct measures for some high-level or complex goals. Instead it is necessary to decompose such a goal into simpler subgoals, which together will contribute to the achievement of the high-level goal. For example, project costs will not usually be tracked simply through a single overall cost figure, but through some Work Breakdown Structure, with budget allocated to lower levels and cost information collected at this lower level of granularity. The depth of decomposition should be limited to a maximum of two levels of breakdown below the primary or high-level goal. This is to limit the amount of data collection and reduction needed, and because it may become very difficult in deep hierarchies to be sure that tracking the subgoals is really contributing to understanding progress against the high-level goal.

Identify the Metrics Required to Satisfy the Subgoals

Purpose

- To determine the metrics which will enable the subgoals to be tracked

The task here is to associate the subgoals with some entity or artifact with measurable properties or attributes. Metrics that are objective and easily quantified are to be preferred.

Identify Project Indicators

Purpose

- To define the information that will be used to monitor and control the project progress, quality and risks, their sources and thresholds for corrective actions.

Project "indicators" are pieces of project information that give a picture of the health of the project's progress against the software development plan. Typically a project manager will be concerned with indicators that apply to the project's scope of work, budget, quality, and risks. The project manager should associate the project indicators with the goals they help to achieve. As a project progresses, the project manager will monitor these indicators and instigate corrective actions when they exceed pre-defined trigger conditions.
These project indicators may include:

- Total spending vs. budget
- Revised scope (work done + estimates to complete) vs. planned scope
- Defect density vs. quality objectives
- Risk indicators (situations that tell you a risk is being realized)

The Project Manager should also define the sources of information for the project indicators. These indicators, in most cases, will be consolidated project measures calculated from more granular primitive metrics that are reported by the project team on a regular basis. How these primitive metrics are to be captured, and the process for using them to calculate the project indicators is defined in the project's Measurement Plan. Other indicators (especially risk indicators) may be simply the status of whether a particular situation has occurred or not. For these cases, the source of the information on the indicator status is all that needs to be identified.

In order to maintain effective control of the project, the project manager defines threshold (or trigger) values/conditions for each of the defined project indicators. These threshold conditions are recorded in the appropriate sections of *Section 4.4 Project Monitoring and Control of the Software Development Plan*. When these thresholds are exceeded, the project manager must take corrective action in order to bring the project back on track. Depending on the severity of the condition, the project manager may be able to resolve the situation within his authority. If the situation goes beyond the project manager's authority he will need to issue a Change Request and activate the project's change control process.

Identify the Primitive Metrics Needed to Compute the Metrics

Purpose

- To determine the basic measurements that will be used to derive the metrics

In this step, the elementary data items, from which the metrics will be derived, are identified. These are the items that will need to be collected.

Write the Measurement Plan

Purpose

- To produce the Measurement Plan artifact

The Measurement Plan captures the goals, subgoals and the associated metrics and primitive metrics. It will also identify the resources (e.g. Project Measurements) and responsibilities for the metrics program.

Evaluate the Measurement Plan

Purpose

- To check the Measurement Plan for consistency, clarity, appropriateness, feasibility and completeness

The Project Manager should have the Measurement Plan reviewed by:

- those directly engaged in the metrics program (in the default organization, the Assessment Team, see Guidelines: Project Plan, Project Organization
- the Project Review Authority (PRA)
- a metrics expert external to the project, unless there are individuals in the Assessment Team who are considered experts.

Put in Place the Collection Mechanisms

Purpose

- To establish the means to collect, record, reduce and report the planned measurements

The instructions, procedures, tools and repositories for metrics collection, computation, display and reporting have to be acquired or produced, installed and set-to-work according to the Measurement Plan. This will include the Project Measurements artifact. See Guidelines: Metrics.

Atividade: Monitorar Status do Projeto

Atividade: Monitorar Status do Projeto

Propósito

- Capturar o status atual do projeto
- Avaliar o status em relação aos planos

Passos

- Capturar status do trabalho
- Derivar indicadores de progresso
- Derivar indicadores de qualidade
- Avaliar indicadores vs. planos

Artefatos de Entrada:	**Artefatos de Saída:**
▪ Plano de Desenvolvimento de Software ▪ Plano de Iteração ▪ Plano de Medições ▪ Registro de Revisão ▪ Medições do Projeto ▪ Lista de Riscos ▪ Lista de Pendências	▪ Medições do Projeto ▪ Lista de Riscos ▪ Lista de Pendências

Freqüência: Diariamente

Papel: Gerente de Projeto

Workflow Details:

- Gerenciamento de Projeto
  - Monitorar e Controlar Projeto

Capturar Status do Trabalho

O Gerente de Projeto captura o status das atividades através de encontros diários com a equipe. Estes encontros devem ter uma duração curta, entre quinze e trinta minutos, e focar na breve reportagem das atividades e dos novos problemas encontrados. Para garantir que o encontro não irá se estender, em geral, todos os participantes ficam de pé. Por isso, são conhecidos como *stand-up meetings*.
Cada membro da equipe deve informar, resumidamente:

- Tarefas completadas e o respectivo esforço gasto;
- Tarefas em que está envolvido e o tempo estimado para completá-las;
- Potenciais problemas na execução destas tarefas;
- Artefatos produzidos;

- Questões ou problemas que necessitem da atenção do Gerente . O gerente de projeto pode registrar estas questões na Lista de Pendências, para uma análise e acompanhamento posteriores.

É importante ressaltar que as soluções para os problemas **não devem ser discutidas** nestes encontros. As pessoas que puderem contribuir para a solução de um problema devem ser reunir posteriormente. A realização de encontros diários melhora a comunicação entre a equipe e garante que todos possuem uma visão geral sobre o que está acontecendo no projeto.
O gerente de projeto deve coletar métricas primitivas sobre o progresso do projeto e a qualidade do produto, a partir destas informações.

Derivar Indicadores de Progresso

Para avaliar apropriadamente o progresso do projeto em relação aos planos, o gerente de projeto consolida as métricas primitivas reportadas pela equipe do projeto para fornecer uma visão geral do progresso alcançado. O Plano de Medições do projeto descreve como estas métricas derivadas (os "indicadores de progresso") são calculadas.

Derivar Indicadores de Qualidade

Além de monitorar o progresso do trabalho, o gerente de projeto também deve monitorar a qualidade dos artefatos do projeto. Métricas de qualidade (também definidas no Plano de Medições do projeto) são consolidadas para fornecer uma visão geral do status do projeto em relação aos objetivos de qualidade.

Avaliar Indicadores vs. Planos

Tendo derivado os indicadores de qualidade e de progresso do projeto, o gerente de projeto os compara com o estado esperado do projeto, conforme definido do Plano de Desenvolvimento de Software e no Plano de Iteração. Neste ponto, o gerente de projeto deve avaliar o seguinte:

- Todas as tarefas planejadas foram cumpridas?
- Os artefatos foram publicados como planejado?
- O esforço estimado para completar as tarefas está de acordo como plano?
- As métricas de qualidade (ex: número de defeitos encontrados) estão dentro da tolerância planejada?

O gerente de projeto também revê a Lista de Riscos para decidir se alguma estratégia de mitigação é necessária no momento.

Ao comparar o progresso com o Plano de Iteração, o gerente de projeto deve sempre ter em mente que as iterações são de **duração fixa** e começar a considerar o caso de alguma funcionalidade ser omitida da iteração, caso pareça que o planejamento original não pode ser cumprido, em vez de modificar a data final da iteração

Qualquer questão reportada deve ser capturada na Lista de Pendências do projeto (que será reportada na Avaliação de Status).Questões que sejam da alçada do gerente de projeto devem ser resolvidas diretamente, como parte da Atividade: Tratar Problemas e Exceções. Algumas vezes pode ser necessário promover o status de uma pendência, por

exemplo, criando uma Requisição de Mudança para ela, ou atualizando a Lista de Riscos, se for uma questão importante.

Novas questões que necessitem ser escaladas para a Autoridade de Revisão do Projeto são incluídas na Avaliação de Status e encaminhadas à ARP, para resolução.Em geral, isto é feito durante a Atividade: Revisão de Projeto com a ARP.

Atividade: Integrar Componentes ao Sistema

Atividade: Integrar Componentes ao Sistema

Propósito
Integrar um novo componente ao sistema, logo que ele é finalizado, produzindo uma versão parcial do produto final.

Passos

- Integrar Novo Componente ao Sistema
- Executar Testes de Regressão
- Promover Código

Artefatos de Entrada:	**Artefatos de Saída:**
▪ Plano de Integração ▪ Componente	▪ *Build*

Papel: Integrador

Subfluxos:

- Implementação
  - Integrar Componentes ao Sistema

Integrar Novo Componente ao Sistema

Logo após um componente ser finalizado e testado unitariamente, a dupla responsável por sua implementação deve integrá-lo ao código estável já existente. Para isso, deve, primeiramente, obter a versão estável mais recente do código, a partir da ferramenta de controle de versões do projeto. Em seguida, deve-se fazer a integração propriamente dita, incluindo-se os novos trechos de código produzidos pela dupla no código estável já existente. Eventuais conflitos decorrentes desta operação devem ser resolvidos pela dupla responsável pela integração, mesmo que sejam em trechos do código não produzidos por eles. Caso seja necessário, pode-se recorrer a ajuda de outros membros da equipe que conheçam melhor o trecho de código em questão.

Após a junção do novo código com o código estável, deve-se gerar um *build* do sistema. Assim como na integração do código, os problemas que aparecerem nesta operação devem ser resolvidos, a princípio, pela dupla que produziu o componente que está sendo integrado. Para a geração do *build*, é fortemente aconselhado o uso de ferramentas, para automatizar este processo. Algumas opções são o GNU-make, Ant e CruiseControl, este último mais voltado para a integração contínua.

Deve ser gerado um novo *build* a cada componente integrado. Caso este processo seja muito demorado e venha a comprometer a produtividade da equipe, pode-se diminuir esta freqüência. Contudo, deve ser gerado no mínimo um *build* diário.

Executar Testes de Regressão

Uma vez gerado o *build* com sucesso, os desenvolvedores devem executar os testes de regressão definidos no Plano de Integração, para garantir que o novo código integrado com o sistema não afetou as funcionalidades já existentes. Esta atividade deve, preferencialmente, ser automatizada. Uma sugestão é a utilização de um dos *frameworks* da família xUnit,de acordo com a linguagem de programação escolhida para o projeto. Um exemplo é a ferramenta JUnit, para projetos em Java.

Caso algum teste falhe, os desenvolvedores que estão integrando o código devem efetuar as correções necessárias para resolver os problemas existentes. Para isso, pode ser necessário, inclusive, alterar trechos de código escritos por outras pessoas. Estas alterações são facilitadas, pois todos na equipe devem estar seguindo o padrão de codificação definido e refatorando o código constantemente, para torná-lo simples e eliminar as redundâncias.

Promover Código

Após a execução com sucesso dos testes de regressão, deve-se promover o código integrado, colocando-o em uma linha-base do sistema juntamente com o código estável já existente. Para isso, é importante seguir os padrões e procedimentos definidos para a Gerência de Configuração e Mudanças no projeto. Uma vez promovido, a nova linha-base do código será utilizada pela próxima dupla de desenvolvedores que for integrar o seu componente.

Apêndice B – Questionários

Neste apêndice são apresentados os dois modelos de questionários distribuídos aos desenvolvedores, para se efetuar a análise crítica das práticas propostas pelo RUP-pe nas disciplinas Implementação e Gerenciamento de Projeto.

Perfil: Gerente de Projeto / Líder de Equipe

1. Perfil do Entrevistado

Experiência técnica (quantos projetos participou antes do atual):

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Experiência no domínio da aplicação:

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Experiência no processo:

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Conhecimento sobre eXtreme Programming (XP):

- ☐ Muito pouco
- ☐ Pouco
- ☐ Médio
- ☐ Extenso
- ☐ Bastante extenso

2. Práticas de Gerenciamento de Projeto

Para cada item abaixo, indique se você já utilizou ou não a prática sugerida, em quantos projetos, e que benefícios e/ou problemas ela trouxe ou poderia trazer, com base na sua experiência em projetos **de até 15 pessoas**.

Iterações de curta duração (entre 2 e 4 semanas)

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Participação de toda a equipe no planejamento (estimativas e divisão de tarefas):

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Balanceamento constante da carga de trabalho entre a equipe, ou seja, acompanhamento periódico de quem está ou não sobrecarregado e nova divisão das atividades:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Acompanhamento do progresso e dos problemas do projeto através de encontros diários da equipe

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Uso de métricas para acompanhar o progresso do projeto e melhorar estimativas e re-planejamento

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Cliente ou especialista do negócio participando da definição do escopo das iterações:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

3. Artefatos

Para os seguintes artefatos do RUP, determine a respectiva utilidade dele nos projetos **de até 15 pessoas** nos quais você participou:

Plano de Resolução de Problemas (define os responsáveis pela análise e resolução de cada tipo de problema, bem como as ferramentas, técnicas e formas de documentação para isto)

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei.
Seria útil? □ Sim □ Não

Plano de Gerenciamento de Riscos (determina as atividades de gerenciamento de riscos que devem ser executadas, bem como responsáveis e recursos):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei
Seria útil? □ Sim □ Não

Ordem de trabalho (lista as atividades que um determinado membro da equipe deve executar, com prazos, dependências e resultados esperado):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
Seria útil? □ Sim □ Não

Avaliação da Iteração (captura os resultados da iteração, para compará-los com os critérios de avaliação estabelecidos e identificar pontos de melhoria para as próximas iterações):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei
Seria útil? □ Sim □ Não

Perfil: Arquiteto / Engenheiro de Software

1. Perfil do Entrevistado

Experiência técnica (quantos projetos participou antes do atual):

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Experiência no domínio da aplicação:

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Experiência no processo:

- ☐ Nenhuma
- ☐ 1 projeto
- ☐ 2 a 3 projetos
- ☐ 4 a 5 projetos
- ☐ Mais de 5 projetos

Conhecimento sobre eXtreme Programming (XP):

- ☐ Muito pouco
- ☐ Pouco
- ☐ Médio
- ☐ Extenso
- ☐ Bastante extenso

2. Práticas de Gerenciamento de Projeto

Para cada item abaixo, indique se você já utilizou ou não a prática sugerida, em quantos projetos, e que benefícios e/ou problemas ela trouxe ou poderia trazer, com base na sua experiência em projetos **de até 15 pessoas**.

Iterações de curta duração (entre 2 e 4 semanas)

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Participação de toda a equipe no planejamento (estimativas e divisão de tarefas):

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Balanceamento constante da carga de trabalho entre a equipe, ou seja, acompanhamento periódico de quem está ou não sobrecarregado e nova divisão das atividades:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Acompanhamento do progresso e dos problemas do projeto através de encontros diários da equipe

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Uso de métricas para acompanhar o progresso do projeto e melhorar estimativas e re-planejamento

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Cliente ou especialista do negócio participando da definição do escopo das iterações:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

3. Práticas de Implementação

Para cada item abaixo, indique se você já utilizou ou não a prática sugerida, em quantos projetos, e que benefícios e/ou problemas ela trouxe ou poderia trazer, com base na sua experiência em projetos **de até 15 pessoas.**

Adoção de um padrão de codificação, para o melhor entendimento do código por parte de todos:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Integração contínua, integrando um novo trecho de código, de maneira automatizada, ao sistema assim que o trecho é finalizado e não somente no final na iteração ou da fase:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Programação em pares:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Desenvolvimento de testes unitários para todo o sistema, com execução automatizada:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Elaboração dos testes antes do início da implementação:

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Refactoring **do código, reestruturando-o constantemente para mantê-lo simples e evitar redundâncias:**

- ☐ Sim, em _____ projeto(s)
- ☐ Nunca utilizei, mas acredito trazer bons resultados
- ☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

Propriedade coletiva do código, onde qualquer pessoa é capaz de alterar qualquer parte do código, se necessário (não existe a noção de "dono" do módulo A ou B):

☐ Sim, em _____ projeto(s)
☐ Nunca utilizei, mas acredito trazer bons resultados
☐ Nunca utilizei e acho que não funcionaria

Benefícios/problemas:

4. Artefatos

Para os seguintes artefatos do RUP, determine a respectiva utilidade dele nos projetos **de até 15 pessoas** nos quais você participou:

Plano de Resolução de Problemas (define os responsáveis pela análise e resolução de cada tipo de problema, bem como as ferramentas, técnicas e formas de documentação para isto)

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei.
Seria útil? □ Sim □ Não

Plano de Gerenciamento de Riscos (determina as atividades de gerenciamento de riscos que devem ser executadas, bem como responsáveis e recursos):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei
Seria útil? □ Sim □ Não

Ordem de trabalho (lista as atividades que um determinado membro da equipe deve executar, com prazos, dependências e resultados esperado):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
Seria útil? □ Sim □ Não

Avaliação da Iteração (captura os resultados da iteração, para compará-los com os critérios de avaliação estabelecidos e identificar pontos de melhoria para as próximas iterações):

☐ Útil na maioria dos projetos
☐ Útil em determinados tipos de projeto. Quais? _____
☐ Sem utilidade
☐ Nunca utilizei
Seria útil? □ Sim □ Não

Dissertação de Mestrado apresentada por **Joaquim Pedro Carvalho de Oliveira** a Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título, **"RUP-pe: Uma Metodologia Ágil, Baseada no RUP e no TSP, para Pequenas Equipes"**, orientada pelo Prof. Alexandre Marcos Lins de Vasconcelos e aprovada pela Banca Examinadora formada pelos professores:

Prof. Hermano Perrelli de Moura
Centro de Informática / UFPE

Prof. Jorge Henrique Cabral Fernandes
Departamento de Informática e Matemática Aplicada / UFRN

Prof. Alexandre Marcos Lins de Vasconcelos
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 11 de setembro de 2003

Prof. JAELSON FREIRE BRELAZ DE CASTRO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.