Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Ciências da Computação

ESCOLHA E IMPLANTAÇÃO DE UMA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO PARA O LABORATÓRIO DE APLICAÇÃO EM TECNOLOGIA DA INFORMAÇÃO

ELTON ANDRADE DOS SANTOS

Florianópolis – SC
Ano 2007 / 2

ELTON ANDRADE DOS SANTOS

ESCOLHA E IMPLANTAÇÃO DE UMA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO PARA O LABORATÓRIO DE APLICAÇÃO EM TECNOLOGIA DA INFORMAÇÃO

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadora: Profa. Maria Marta Leite

Banca Examinadora
Profa. Maria Marta Leite
Prof. Leandro José Komosinksi
Sr. Marcio Clemes

Dedico este trabalho ao meu amigo Max,
que Deus o tenha.

Agradeço aos amigos e parentes que me apoiaram durante o desenvolvimento deste trabalho e durante toda a jornada da faculdade (e por agüentar todo esse papo de metodologia, mesmo sem entender nada do assunto), aos membros da banca e a minha orientadora pela paciência e pelo conhecimento que deles recebi, e também à banda Veruca Salt, por compor toda a trilha sonora deste trabalho. E principalmente, agradeço a minha namorada Ana por todo o amor, apoio e suporte (e por me ajudar a escrever os *abstracts* :-)

Resumo

Durante vários anos o Laboratório de Aplicação de Tecnologia da Informação (LATIN), da Universidade Federal de Santa Catarina (UFSC), operou sem nenhuma metodologia de desenvolvimento de *software*, construindo aplicações e sistemas sem nenhuma organização ou método. Na medida em que a complexidade das aplicações e tecnologias utilizadas foi crescendo, o que se notou foi uma enorme lentidão no processo de desenvolvimento do laboratório. Para resolver este problema foi proposta a escolha, adaptação e implantação de uma metodologia de desenvolvimento de *software* nos projetos do laboratório. Diversas metodologias de desenvolvimento foram estudadas, sendo que a escolhida foi a metodologia *EasyProcess*, mais conhecida como YP. Esta metodologia, desenvolvida na Universidade Federal de Campina Grande (UFCG), possui um processo robusto e completo, e propõe o uso de técnicas de gerência e programação para auxiliar o processo de desenvolvimento de *softwares*. Com a aplicação do YP no LATIN pôde-se notar melhorias em todo o processo, desde a gerência de projetos, agora mais clara e objetiva, até a codificação, que agora é mais organizada e previsível. A implantação do YP no LATIN foi um primeiro passo na direção contrária ao caos que se instalava.

Palavras-chave: processo, metodologia, desenvolvimento, LATIN, YP

Abstract

During many years the Laboratory for Application of Technology Information (LATIN), of the Federal University of Santa Catarina (UFSC), has worked without any software development methodology, building up applications and systems without any organization or method. While the complexity of the applications and technologies used grew, it was noticed a great slowness on the laboratory's development process. To deal with this problem it was proposed the choice, adaptation and implantation of a software development methodology in the projects of the laboratory. Many development methodologies were studied, and the chosen one was the easYProcess methodology, known as YP. This methodology, developed in the Federal University of Campina Grande (UFCG), has a solid and complete process, and proposes the usage of management and programming techniques to improve the software development process. With the application of YP in LATIN it was observed improvement in all the process, from the project management, now clearer and more objective, to the coding, which is now more organized and predictable. The implantation of YP in LATIN was the first step in the opposite direction of the chaos that has been installed.

Keywords: process, methodology, development, LATIN, YP

Sumário

Índice de Figuras

Índice de Gráficos e Tabelas

Lista de Abreviaturas e Siglas

CVS – *Concurrent Version System*

DSDM – *Dynamic System Development Method*

EJB – *Enterprise Java Beans*

FAPEU – Fundação de Amparo á Pesquisa e Extensão Universitária

FTP – *File Transfer Protocol*

JUDE - *Java and UML Developers' Environment*

IBM – *International Business Machines*

IDE – *Integrated Development Environment*

LATIN – Laboratório de Aplicação em Tecnologia da Informação

LATTES - Sistema de currículos de pesquisadores no Brasil, do CNPq.

NPD – Núcleo de Processamento de Dados

PAD – Plano de Atividades Departamental

PAI – Plano de Atividades Institucional

PET – Programa Especial de Treinamento

PIA – Plano Individual de Atividades

RUP – *Rational Unified Process*

SINGU – Sistema Integrado de Gestão Universitária

STELA – Instituto Stela

TAOS – *Task and Action Oriented System*

TDD – *Test Driven Development*

UFCG – Universidade Federal de Campina Grande

UFSC – Universidade Federal de Santa Catarina

UNIR – Universidade Federal de Rondônia

UML – *Unified Modeling Language*

XP – *Extreme Programming*

YP - *easYProcess*

1 INTRODUÇÃO

Neste capítulo explicam-se os motivos que implicaram na realização deste trabalho, fazendo uma breve apresentação do problema e da solução levantada, bem como o contexto de sua implantação, e a metodologia utilizada na sua execução.

1.1 APRESENTAÇÃO

O Laboratório de Aplicação de Tecnologia da Informação da Universidade Federal de Santa Catarina (LATIN – UFSC) produz uma moderada gama de aplicações *WEB*, normalmente para uso interno da própria instituição. Nele desenvolvem-se, já há algum tempo, entre outros projetos, o projeto para preenchimento dos Planos Individuais de Atividade dos Docentes (PIA), do departamento (PAD), e da própria instituição (PAI), de forma automatizada e dinâmica. Este projeto, chamado de Sistema Integrado de Gestão Universitária (SINGU-UFSC), é desenvolvido, em grande parte, apenas por estudantes, através de estágios.

Exatamente por ter uma equipe de programadores inexperientes é que existem, no processo de desenvolvimento, diversos problemas relacionados à especificação, documentação, padronização dos trabalhos, entre outros. O resultado é que o desenvolvimento dos projetos do laboratório é demasiado lento, o que pode vir a inviabilizar, inclusive, a finalização e implantação dos mesmos.

Motivado, então, pelo supervisor geral do laboratório, deu-se início a este trabalho no intuito de tentar, através de uma possível otimização no processo de desenvolvimento, aumentar a produtividade da equipe e a organização dos trabalhos, resultando em *softwares* criados com rapidez, qualidade, padronização e com prazos e metas realistas.

1.2 FORMULAÇÃO DO PROBLEMA

Após análise dos problemas levantados e constatados no processo de desenvolvimento do LATIN, optou-se por adotar uma metodologia de desenvolvimento para resolver o problema escolhido como mais crítico: a ausência de uma metodologia de desenvolvimento.

Este trabalho procura, através de uma pesquisa sobre o assunto, apresentar um estudo sobre as principais metodologias utilizadas no mercado, e subseqüente escolha e implantação de uma metodologia de desenvolvimento adequada aos projetos do laboratório, agilizando o processo de desenvolvimento no LATIN.

1.3 JUSTIFICATIVAS

Os motivos que justificam a realização deste trabalho são os que seguem:

- Aprender, através de pesquisa bibliográfica, mais sobre desenvolvimento de *software*, processos e metodologias, para que os conhecimentos adquiridos sejam aplicados no processo de desenvolvimento do laboratório.
- O processo de desenvolvimento do LATIN é caótico e desorganizado. Espera-se que com a implantação de uma metodologia de desenvolvimento se consiga uma maior organização e uma maior agilidade na tarefa de produzir código. Tudo isto através de melhorias na gerência dos projetos e na etapa de implementação das aplicações.

1.4 OBJETIVOS

A seguir serão apresentados os objetivos, gerais e específicos, deste trabalho.

1.4.1 OBJETIVO GERAL

Organizar e melhorar o processo de desenvolvimento nos projetos realizados pelo Laboratório de Aplicação de Tecnologias de Informação (LATIN) através da implantação de uma metodologia de desenvolvimento de *software* adequada.

1.4.2 OBJETIVOS ESPECÍFICOS

I – Realizar uma análise dos problemas existentes no processo de desenvolvimento atual, justificando e motivando a implantação de uma metodologia de desenvolvimento.

II – Realizar uma revisão teórica sobre processo de desenvolvimento e metodologias de desenvolvimento, realizando uma pequena pesquisa sobre as mais utilizadas atualmente no mercado, como o *Extreme Programming* (XP), *Rational Unified Process* (RUP), *Scrum*, Família *Crystal* e *easYProcess* (YP). Subseqüentemente deve-se realizar uma comparação entre as metodologias analisadas para que se possa escolher a mais adequada para a o problema.

III – Planejar uma implantação da metodologia escolhida e em seguida implantar esta metodologia em projetos do LATIN. Durante o uso da metodologia os principais problemas ocorridos, bem como as melhorias obtidas, devem ser observados.

IV – Obter, após algumas semanas de uso da metodologia em projetos do laboratório, métricas ou dados para avaliar o impacto da implantação da metodologia no ambiente do LATIN. E, de posse destas métricas, realizar uma análise de possíveis melhoras no processo de desenvolvimento, decorrente da aplicação da solução implantada.

1.5 DELIMITAÇÃO DO ESCOPO

Este trabalho limita-se a pesquisa, adaptação e implantação de uma metodologia de desenvolvimento em um ambiente já conhecido: o LATIN.

A pesquisa é focada em metodologia e processo, e a implantação é realizada apenas para a metodologia escolhida como mais propícia para os projetos.

Nota-se também que em momento algum é proposto o desenvolvimento de uma nova metodologia, e sim a adaptação e melhoria de uma já existente.

As avaliações realizadas se baseiam em métricas simples, como, por exemplo: o número de tarefas realizadas. Também são baseadas em dados como: conforto no uso da metodologia, nível de organização nos projetos e produtividade. A medição destes dados é fortemente embasada em opiniões dos envolvidos no processo.

1.6 METODOLOGIA DE DESENVOLVIMENTO DESTE TRABALHO

Este trabalho faz uma revisão sobre processo de desenvolvimento e metodologias de desenvolvimento, e um estudo de diversas metodologias.

A pesquisa é realizada em literatura especializada no assunto, tendo como principais: Cockburn (2002), Fowler (2003) e Abrahamsson et al. (2002).

Em seguida é feito um estudo para a implantação da metodologia escolhida no laboratório, levantando possíveis problemas e dificuldades que porventura venham a ser enfrentados, e como resolvê-los e/ou minimizá-los.

Logo após realiza-se a implantação da metodologia em alguns dos projetos do laboratório, e procura-se avaliar as melhorias e impactos causados pela prática.

Finaliza-se então este trabalho levantando conclusões sobre as avaliações, análises, métricas ou dados obtidos, para que se possa afirmar, ou não, que a implantação de uma metodologia de desenvolvimento auxiliou e/ou possibilitou melhorias no processo de produção de *software* no laboratório.

1.7 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado da seguinte forma: esta introdução tem o objetivo de posicionar o trabalho e definir seu escopo.

O capítulo 2 tem por objetivo descrever o laboratório onde as atividades deste são aplicadas, sua estrutura física, equipe, e projetos envolvidos. Este capítulo descreve também o processo de desenvolvimento utilizado no LATIN antes da implantação da nova metodologia, procurando, através do entendimento do modo como as atividades eram realizadas anteriormente, levantar os principais problemas e dificuldades encontradas.

Os capítulos 3 e 4 tratam da pesquisa bibliográfica sobre os assuntos: processos de desenvolvimento e metodologias de desenvolvimento. É através do conhecimento adquirido nesta etapa do trabalho que se consegue levantar possíveis métricas e métodos para escolher, adaptar e avaliar uma metodologia. Também são descritas as principais metodologias utilizadas pelo mercado e pela academia atualmente.

O capítulo 5 propõe, através de um estudo, um método para a implantação de uma metodologia no LATIN, realizando e justificando a escolha de uma metodologia de desenvolvimento, procurando prever as principais dificuldades no processo de implantação e propondo, também, adaptações na metodologia, deixando-a mais dinâmica e condizente com a habilidade da equipe que a utilizará e com os projetos em que será aplicada. Este capítulo termina propondo, também através de estudos e pesquisas, meios de avaliar o sucesso ou o insucesso desta implantação. Levantam-se métricas simples, baseadas no processo e nas pessoas envolvidas, como por exemplo: a satisfação da equipe quanto à determinada etapa do processo ou o número de tarefas completadas durante determinada iteração.

O capítulo 6 trata dos eventos decorrentes da aplicação concreta da metodologia escolhida em projetos novos e/ou já ativos. Através de constantes revisões do processo e discussão com os envolvidos no mesmo, procura-se analisar e adaptar a metodologia, almejando sempre a simplicidade e a suficiência das atividades, para manter uma boa produtividade e qualidade no desenvolvimento. Além de tratar destes assuntos, o capítulo 8 procura-se também fazer um relato preciso das dificuldades e problemas encontrados, e como foi feito para solucioná-los ou minimizá-los.

Em seguida, no capítulo 7, levantam-se conclusões das atividades realizadas, com ênfase nas experiências adquiridas. São também relacionados possíveis trabalhos para o futuro.

Finalmente, no capítulo 8, são apresentadas as referências bibliográficas para este trabalho e, logo em seguida, os apêndices.

2 O LABORATÓRIO DE APLICAÇÃO EM TECNOLOGIA DA INFORMAÇÃO

Este capítulo tem como objetivo realizar uma breve apresentação do Laboratório de Aplicação em Tecnologia da Informação (LATIN), localizado no Núcleo de Processamento de Dados (NPD) da Universidade Federal de Santa Catarina (UFSC). Procura-se comentar seus objetivos em relação à instituição, sua estrutura e espaço físico, e seus recursos. Finalmente, este capítulo é encerrado com uma breve descrição dos projetos em que o LATIN está envolvido, e que são pertinentes a este trabalho.

2.1 OBJETIVOS DO LATIN

O Laboratório de Aplicação de Tecnologia de Informação (LATIN) tem como objetivos o estudo e a aplicação de tecnologias que auxiliem no desenvolvimento de softwares, utilizando diversas ferramentas e linguagens de programação.

O LATIN faz parte do Núcleo de Processamento de Dados da UFSC, núcleo este que tem por finalidade prestar serviços na área de sua especialidade ao Ensino, Pesquisa, e a atividades de Extensão e Administração da Universidade Federal de Santa Catarina.

2.2 ESTRUTURA E ESPAÇO FÍSICO

Atualmente o LATIN está alocado em uma sala no andar térreo do prédio do NPD da UFSC, ficando, assim, próximo das equipes que mantém e desenvolvem grande parte dos diversos subsistemas da UFSC, bem como de toda a estrutura de rede da instituição.

O laboratório possui um ambiente agradável, climatizado, e equipado com diversas máquinas para o desenvolvimento de sistemas computadorizados. Possui também um servidor, com sistema operacional Linux, que oferece serviços *WEB*, *E-*

Mail, CVS, FTP, entre outros. Também conta com acesso aos servidores de banco de dados de desenvolvimento e de produção da instituição.

2.3 EQUIPE ATUAL

O laboratório possui uma equipe de cinco desenvolvedores, todos estagiários do curso de Ciências da Computação e de Sistemas de Informação da UFSC. A maioria destes estagiários está em fases avançadas dos cursos, e possui média ou pouca experiência, sendo que o colaborador mais antigo do laboratório é o autor deste trabalho, que atualmente faz o papel de gerente de alguns projetos do mesmo.

2.4 PROJETOS

O LATIN possui em seu portfólio diversos projetos finalizados, como, por exemplo, o sistema de matrícula para alunos da graduação. Possui também projetos em andamento, como o fórum da graduação e o projeto de acompanhamento das atividades dos docentes.

Além de projetos desenvolvidos para a UFSC, o LATIN também desenvolveu o projeto do Sistema Integrado de Gestão Universitária da Universidade Federal de Rondônia (UNIR), ou SINGU/UNIR. O mesmo foi desenvolvido para a UNIR através da Fundação de Amparo á Pesquisa e Extensão Universitária (FAPEU). Durante o desenvolvimento do SINGU/UNIR, também foi desenvolvido o *framework* que hoje serve de base para o desenvolvimento de aplicações *WEB* no laboratório.

Os projetos são desenvolvidos, em sua grande maioria, utilizando tecnologia Java, e normalmente na forma de aplicações para a *WEB*.

2.5 O PROCESSO DE DESENVOLVIMENTO NO LATIN

O processo de desenvolvimento utilizado anteriormente a este trabalho no laboratório possuía diversas peculiaridades. Esta seçao entra em mais detalhes

sobre o mesmo, descrevendo os métodos e ferramentas utilizados. Finalmente, procura levantar as dificuldades e problemas inerentes a este processo, propondo uma solução para o problema.

2.5.1 METODOLOGIA UTILIZADA

O laboratório não utilizava, no processo descrito nesta seção, uma metodologia específica para o desenvolvimento de aplicações. Na falta de uma metodologia, e de qualquer outro método de organização, se categoriza este processo de desenvolvimento como “Codificar e Arrumar”, ou, do inglês, *“Code and Fix”* (este tipo de processo será abordado com mais detalhes no capítulo 3).

2.5.2 FERRAMENTAS UTILIZADAS

Utiliza-se, para o processo aqui descrito, largamente, apenas as três ferramentas seguintes:

- *Eclipse* IDE[1] (Figura 2.1), um ambiente integrado de desenvolvimento, que integra diversas pequenas ferramentas, algumas imprescindíveis para o desenvolvimento dos projetos em andamento no laboratório;
- CVS, integrado ao *Eclipse*, um sistema de controle de versões, que possibilita o desenvolvimento em equipe do sistema, fazendo a “mescla” do código gerado pelos diversos envolvidos no projeto;
- ASEisql: uma ferramenta utilizada para o acesso ao servidor de banco de dados *Sybase* da UFSC, realizando consultas, inserções, atualizações, remoções e estruturações nas bases de dados utilizadas.

[1] *Integrated Development Environment*, ou: Ambiente de desenvolvimento integrado.

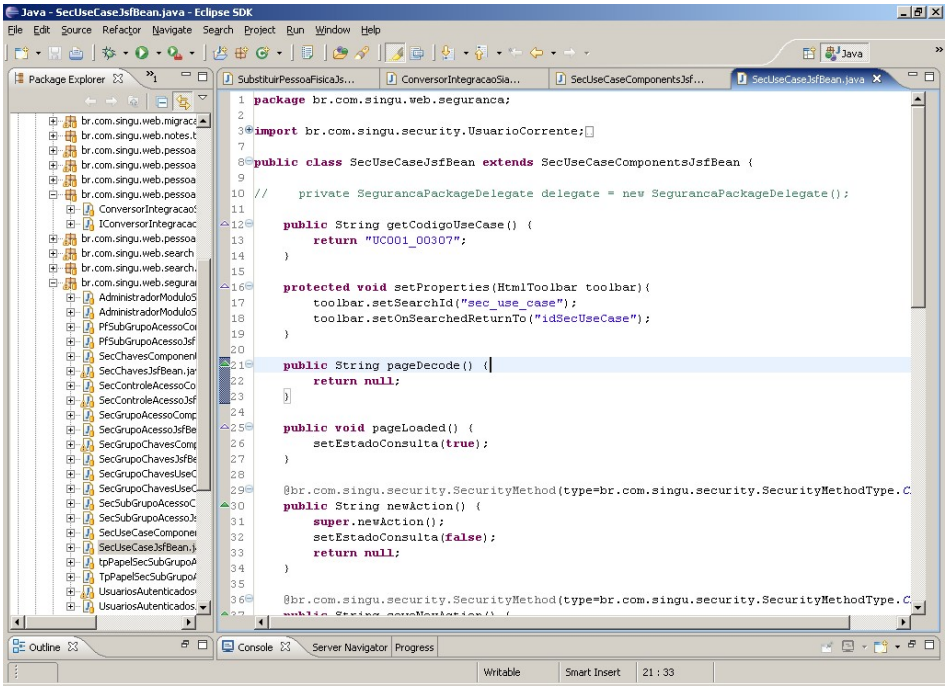


Figura 2.1: *Eclipse* IDE.

Também são utilizadas outras ferramentas no laboratório, mas seu uso é mais esporádico, e normalmente vindo de uma necessidade imediata. São algumas delas:

- *Sybase PowerDesigner* 10: Ferramenta para edição e/ou criação de diagramas UML;
- ERWin: Ferramenta para modelagem e especificação de bancos de dados;
- Gerador de Códigos *EJB*: Pequeno *software* desenvolvido pela equipe do laboratório, que consiste de um gerador de código, para a camada de persistência, de projetos baseados no *framework* SINGU, desenvolvido e mantido pelo LATIN.

2.5.3 PROBLEMAS NESTE PROCESSO DE DESENVOLVIMENTO

Como foi abordado no início desta seção, a falta de metodologia e organização para o desenvolvimento gerava diversos problemas e dificuldades. Esta

subseção se propõe a apresentar e analisar esses problemas e dificuldades encontrados.

Durante o período de utilização do processo abordado nesta seção o laboratório se encontrava com problemas sérios de produtividade no desenvolvimento de *software*. Isso pôde ser constatado com facilidade pela equipe e pela supervisão do laboratório, já que o desempenho estava muito aquém do esperado, e até mesmo as tarefas mais simples demoravam dias para serem terminadas, e todos os prazos estipulados eram freqüentemente ultrapassados.

Após uma análise deste processo de desenvolvimento, feita por toda a equipe envolvida, bem como pelos responsáveis pela supervisão das atividades, e contrastada com os estudos de Martin Fowler (2003) e Alistair Cockburn (2002, 1999), chegou-se a algumas constatações sobre os problemas que possivelmente são as causas desta baixa produtividade. São elas:

- Alta rotatividade da equipe, gerando um constante processo de treinamento dos novos envolvidos no projeto. Uma agravante também é a incompatibilidade de horários dos estagiários, todos graduandos e com grades de horários nas mais variadas formas. Por não se ter uma gerência de tarefas adequada muita vezes estes estagiários não sabem o que fazer, ficando “parados” e, assim, não produzindo;
- Ausência de documentação, tanto dos projetos quanto do próprio processo de desenvolvimento, o que muitas vezes impossibilita tomar-se conhecimento de diversos aspectos dos sistemas desenvolvidos e/ou em desenvolvimento. Também inviabiliza o auto-aprendizado de novos membros, pois toda dúvida sobre o desenvolvimento, ou sobre os projetos, deve ser sanada com os colaboradores mais experientes;
- Ferramentas obsoletas e/ou descontinuadas, como o CVS, por exemplo, que podem ser substituídas por outras mais recentes, com melhor potencial e maior número de mecanismos facilitadores, e que possivelmente auxiliariam o desenvolvedor em diversos pontos do processo de criação do *software*. A falta de ferramentas também é uma agravante deste ponto, por exemplo: em algumas aplicações

WEB desenvolvidas, que utilizam o *framework* SINGU, as interfaces são escritas em código, sem auxílio de editores especializados, o que gera um esforço muito grande para que se criem, até mesmo, as mais simples interfaces;

- Atualmente não há uma metodologia para o treinamento de novos recursos humanos, o que acaba por implicar em um processo de aprendizado lento e ineficiente. Desta forma impede-se que se desenvolva o máximo potencial de cada desenvolvedor, e geram-se, muitas vezes, diversas dúvidas por parte dos novatos. Essas dúvidas, geradas pelo treinamento insuficiente, devem ser sanadas pelos colaboradores mais experientes, resultando em um decrescimento da produtividade de toda a equipe;
- A falta de uma metodologia de desenvolvimento, ponto este considerado pela equipe do LATIN como o mais crítico, gera uma grande variedade de problemas: ausência de padronização no processo de desenvolvimento, inviabilidade do planejamento de prazos e metas, maus-hábitos no desenvolvimento (código escrito de qualquer jeito, sem nenhum critério ou padronização), ausência de uma etapa de especificação, o que acarreta em não conhecer plenamente o problema, falta de uma etapa de documentação do sistema desenvolvido, ausência de planejamento e execução de testes adequados, etc.;
- Ausência de uma gerência das tarefas que devem ser realizadas pela equipe, bem como de um histórico das tarefas já executadas, o que impede o planejamento de prazos baseado em desenvolvimentos similares já realizados. Este problema também implica na total falta de um cronograma de atividades do laboratório.
- A equipe possui pouquíssima experiência com programação (especialmente se voltada para a *WEB*) e em técnicas de Engenharia de *Software*. Isto faz com que a equipe tenha muitas dificuldades em resolver problemas mais complexos de programação, exigindo dos

programadores mais experientes um esforço adicional para ensiná-los. Isto também pode acarretar em um código fraco e com muitos erros.

2.5.4 CONCLUSÕES

Aliados, todos os problemas levantados geram uma grande lentidão em todo este processo, baixando por demais a produtividade do desenvolvimento e muitas vezes, quase inviabilizando alguns projetos.

Os problemas de alta-rotatividade da equipe, pouca experiência em programação, e ausência de treinamento de novos recursos-humanos não têm solução no momento, visto que grande parte da equipe é formada por estagiários e a instituição não pode (ou não quer) alocar funcionários para o laboratório. Já os demais problemas são de ordem técnica, e conseqüentemente de possível solução. A ausência de ferramental de apoio atualizado pode ser resolvida com implantação de novas ferramentas, mais atuais, desde que o impacto de sua implantação seja pequeno ou inexistente. Já os problemas restantes: gerência de tarefas, documentação e metodologia, podem todos ser resolvidos com a implantação de uma metodologia de desenvolvimento de *software* apropriada.

Concluiu-se então, analisando estes problemas, que mudanças eram necessárias no processo utilizado. Em reunião com a equipe envolvida nas atividades do laboratório foi proposta a implantação de uma metodologia de desenvolvimento, visto que, além de se tratar de um problema de solução viável, a falta de organização no processo foi considerada o “gargalo” de todo o problema de produtividade. Também foi considerado o fato de existirem diversas metodologias disponíveis na literatura para o uso e adaptação ao laboratório, facilitando todo o processo de pesquisa, estudo e implantação.

3 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Neste capítulo se faz uma breve introdução sobre Desenvolvimento de *Software* e Processo de Desenvolvimento.

3.1 "UM JOGO COOPERATIVO DE COMUNICAÇÃO E INVENÇÃO"

Quando se pensa em desenvolvimento de *software*, logo se pensa em programação, codificação de programas. Programar faz parte do desenvolvimento de *software*, mas desenvolver *software* não é apenas programar. Há muito mais coisas envolvidas no chamado processo de desenvolvimento de *software*.

O autor Alistair Cockburn (2002) faz uma analogia entre desenvolvimento de *software* e jogos muito interessante: ele define desenvolvimento de *software* como um “Jogo cooperativo de comunicação e invenção”. Cooperativo por que os envolvidos na equipe de desenvolvimento ajudam uns aos outros a atingirem suas metas. Invenção por que existe um conjunto de idéias que vai sendo formado durante o processo que é aplicado para atingir as metas. E, finalmente, comunicativo por que essas idéias têm que ser repassadas, para a equipe e/ou para o computador, para que o desenvolvimento flua com rapidez. As metas deste jogo de invenção e comunicação são:

- Desenvolver um *software*;
- Criar uma posição vantajosa para o próximo jogo. Ou seja, produzir mais *software*, ou fazer manutenção no *software* desenvolvido anteriormente.

Tão importante quanto a primeira meta, desenvolver *software*, a segunda meta propõe que, durante este desenvolvimento, também se deixe “preparado o terreno” para possíveis modificações no *software* desenvolvido, e que não é uma boa prática desenvolver um *software* que num futuro, próximo ou não, se torne inviável de modificar, seja por falta de documentação, por ser muito complexo, etc.

Então, de acordo com a analogia de Cockburn, um bom “time” é aquele que consegue criar e se comunicar com eficiência, e não apenas um time com

muitos talentos individuais que não se comunicam. Também é uma equipe que prepara *software* de qualidade, possibilitando modificações futuras, através de código bem formulado, ou através de uma boa documentação, entre outras técnicas possíveis para atingir esta meta.

Então, de acordo com esta analogia, desenvolver *software* é, além de programação, formulação de idéias e comunicação.

3.2 O QUE É UM PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*

Durante muitos anos o processo de desenvolvimento de *software* era algo caótico e desorganizado. A chamada “crise do *software*”, termo referenciado por Dijkstra (1972), foi um período caracterizado pela ausência de engenharia de *software*, aumento da demanda por *software*, e o uso predominante, por parte de desenvolvedores mundo afora, de um método chamado “programar e consertar”, ou do inglês “*code and fix*”. Um processo simples, em que “o *software* é escrito sem um plano definido e o projeto do sistema é repleto de várias decisões de curto prazo”, de acordo com Fowler (2003).

Com o passar do tempo os projetos foram crescendo, a complexidade aumentando, e o uso de um processo caótico como o “c*ode and fix*” foi se tornando impraticável, pois em projetos de médio ou grande porte, os erros acumulados, e a falta de previsibilidade, proporcionados pelo processo, acabavam, muitas vezes, por inviabilizar sua realização.

A falta de técnicas específicas para o desenvolvimento de *software*, principalmente de processos de desenvolvimento bem definidos, aliada ao crescimento da indústria do *software*, impulsionou o surgimento e evolução da Engenharia de *Software*.

A Engenharia de *Software*, importante área de conhecimento da Informática, é “uma abordagem disciplinada, matemática e quantificável para o desenvolvimento, operação e manutenção de *software*”, IEEE (1990). Um processo de desenvolvimento é algo que possibilita isto.

Um processo de desenvolvimento de *software* é um “conjunto de ferramentas, métodos e práticas usados para construir um produto de *software*” de acordo com Humphrey (1990). As grandes fases de qualquer processo de desenvolvimento são: (Larman, 2000):

- Planejamento e elaboração: etapa em que se planeja o que vai ser feito, quem faz o que, quando e como;
- Construção do sistema: etapa caracterizada por codificação e testes;
- Instalação, ou implantação: onde se põe o sistema desenvolvido em produção, treina-se usuários, entre outras atividades.

De acordo com estas definições, se conclui que um processo de desenvolvimento deve ser uma ferramenta para planejar e executar de forma disciplinada, seguindo uma seqüência pré-definida de atividades, e que, eventualmente, culminará no cumprimento das metas do jogo cooperativo de criação e comunicação que é o desenvolvimento de um *software*.

3.3 O QUE ESTÁ ENVOLVIDO

Na seção anterior definiu-se o que é um processo de desenvolvimento, mas é impossível analisar sua eficácia ou não sem saber que “componentes” ou “recursos” compõem um processo de desenvolvimento.

Se analisado torna-se óbvio que, entre tantos, o principal recurso que compõe o processo de um desenvolvimento é o elemento humano. E é exatamente este “recurso” que muitas vezes define a eficácia ou não de um processo.

O elemento humano representa uma variável altamente imprevisível em um processo de desenvolvimento de *software*. Cockburn (1999) defende em seu artigo chamado “*Characterizing People as Non-linear, First-order components in Software Development*”, que as pessoas envolvidas no desenvolvimento não são apenas mais uma variável no processo, mas são a variável mais importante do mesmo:

> No título eu me refiro às pessoas como "componentes". É assim que as pessoas são tratadas na literatura de processo/metodologia. O engano nesta abordagem é que "pessoas" são altamente variáveis e não-lineares, com fatores de sucesso e fracasso únicos. Esses fatores são de primeira ordem, não negligenciáveis. Falhas em levar em conta esses fatores por parte dos criadores de processos e metodologias contribuem para todo tipo de trajetórias não-planejadas que comumente vemos.

Analisando-se esta citação, pode se ver que Cockburn coloca o elemento humano como o maior coringa de qualquer processo e/ou metodologia. Ele parte do princípio de que pessoas são imprevisíveis, e que esta imprevisibilidade acaba influenciando, e muito, no sucesso de todo o processo de desenvolvimento. Novamente pensando no desenvolvimento de *software* como um jogo cooperativo de invenção e comunicação, pode-se concluir que, se a equipe não quer jogar o jogo, não haverá jogo para ser jogado. Pode-se concluir também que, mesmo com muita disciplina, sem comunicação e cooperação a equipe muito provavelmente falhará. Em um bom processo tudo isto deve ser evitado para que se possibilite a comunicação e a cooperação necessária para que a equipe vença o jogo.

3.4 UM BOM PROCESSO

Um processo deve empregar a disciplina e a organização, deve planejar com antecedência atividades do desenvolvimento, mas sempre tendo em mente que imprevistos acontecem e influenciam neste processo. Nesta seção é discutido como deve ser um processo para conseguir isto, que características do projeto e da equipe de desenvolvimento influenciam na escolha ou criação de um processo de desenvolvimento, e se o desenvolvimento deve se moldar ao processo ou o processo deve se adaptar ao desenvolvimento.

Certamente o “programar e codificar” não é um bom processo. Talvez seja para projetos muito pequenos, mas definitivamente não é para projetos com alguma complexidade. Exatamente por não haver alguma organização, ou um bom planejamento anterior à atividade de codificar.

De acordo com o autor Martin Fowler (2003), foi tentando resolver essa deficiência que se procurou espelhar-se em disciplinas de engenharia para propor

uma organização ao caos do desenvolvimento. Surgem os primeiros processos bem-definidos de desenvolvimento.

Esses processos vindos da engenharia propunham o planejamento prévio de todas as atividades do processo. Mas isso não resolve o problema da imprevisibilidade, pelo contrário, provavelmente não será cumprido o planejado, pois imprevistos sempre acontecem.

“Pessoas cometem erros. Isso não é surpresa para nós” (Cockburn, 2002). Para tentar detectar, e possivelmente evitar, estes erros é que surgiram os processos iterativos e incrementais. Abaixo segue uma breve definição de cada tipo de processo, de acordo com Martin Fowler (2003) e Alistair Cockburn (2002).

- Processo iterativo: propõe a organização do projeto em estágios, ou iterações, em que diversas partes do sistema são desenvolvidas e melhoradas a cada iteração, o que permite o re-trabalho, ou refatoração, de pedaços do sistema.
- Processo incremental: propõe a divisão do sistema em subsistemas, que devem ser implementados em separado, possivelmente em seqüência, e posteriormente integrados. Esta divisão em estágios não ajuda muito a prever erros, mas prepara o desenvolvimento para eles, antecipando sua detecção e diminuindo seu impacto no tempo de desenvolvimento.
- Processo predeterminante: neste processo não há espaços para erros, pois tudo é planejado antes do desenvolvimento. Se esses erros acontecem, provavelmente se acumulam até as fases finais do projeto. E há outro fator de imprevisibilidade causado pelo fator humano: o cliente nem sempre sabe o que quer, e quando sabe não sabe transmitir o que quer. Isso implica na mudança freqüente dos requisitos do sistema, o que faz com que o planejamento anterior de todo o sistema se torne algo muitas vezes inviável.

Na abordagem incremental os erros acontecem, e devem ser corrigidos e detectados antes da próxima etapa, ou estágio. Já na abordagem iterativa, o

processo de re-visitar e melhorar partes do sistema de iterações anteriores faz com que os erros sejam corrigidos durante a evolução do *software*.

Outro aspecto interessante das abordagens iterativas e incrementais é o constante planejamento do desenvolvimento, o que torna a modelagem de um sistema algo mais recente e não um plano intocável, feito há muito tempo. Isso possibilita a re-modelagem de partes do sistema, causadas por uma mudança nos requisitos, diminuindo o impacto destas mudanças no desenvolvimento do sistema. Essa constante necessidade de planejar o próximo passo também aumenta a comunicação entre os membros da equipe, bem como entre a equipe e o cliente, incentivando a geração e a troca de idéias. Isso não resolve todo o problema de comunicação, mas é um passo a frente neste sentido.

Mas nem sempre só o projeto muda com o tempo. Em um projeto longo pode ocorrer, principalmente no âmbito acadêmico, uma freqüente mudança na equipe de desenvolvimento. Esse “efeito rotativo” da equipe de trabalho é muito notado em lanchonetes de *fast-food* mundo afora. Normalmente um emprego nestas áreas é algo temporário, passageiro, ou o primeiro emprego, e normalmente não é tratado como o final de uma carreira. Logo, a equipe muda o tempo todo. Mas, mesmo com a mudança freqüente da equipe, o lanche continua sendo servido do mesmo jeito, e com a mesma rapidez. Isso ocorre porque o processo usado nessas casas de *fast-food* continua, mesmo quando um membro da equipe sai. Logo outro entra em seu lugar, aprende o processo (que normalmente é simples) e executa as mesmas tarefas, do mesmo jeito, sempre. O novo funcionário sabe que deve fritar o bife 15 segundos de cada lado, colocar então o pão, o molho, que já vem preparado, o bife, o alface, e o queijo, nesta mesma ordem, e por fim, embalar e passar para o balcão.

Essa é uma característica muito importante dos processos predeterminantes. Em um processo predeterminante as pessoas envolvidas são tratadas como “recursos”, como peças substituíveis. Isso é muito observado também em ramos da engenharia, como a construção civil: o projeto e o planejamento são feitos pela equipe mais qualificada, sobrando a maior parte do processo, a construção em si, para a mão-de-obra menos qualificada, e substituível.

Mas processos relativamente simples como esses, de levantar uma parede de tijolos ou fritar batatas, são diferentes de um processo relativamente mais complexo como o de desenvolver *software*. Não é necessário muito talento e experiência para fritar *hamburgers* e pintar paredes, mas para projetar e programar *software* talento e experiência contam.

No referido processo de desenvolvimento do tipo predeterminante, teoricamente não. Tudo é planejado e modelado antes, logo é apenas uma questão de seguir o planejado, sempre do mesmo jeito. Mas como diz o ditado popular: “A teoria na prática é outra...”. Principalmente com os problemas de imprevisibilidade em um projeto, discutidos anteriormente. Já em um processo do tipo incremental ou iterativo, o talento e a experiência contam, e muito. Pois a habilidade de se desenvolver com mais rapidez e qualidade diminui os erros e acelera as etapas. Logo, esses processos são centrados em pessoas (Cockburn, 2002), e dependem dos conhecimentos, talentos e experiências conjuntas dos envolvidos, tornando-os peças fundamentais, e muitas vezes não substituíveis, desses processos.

Com base no que foi abordado, conclui-se então que um processo ideal seria um processo rápido, que prevê falhas, diminui o impacto dos erros, e que não dependa exclusivamente do talento e experiência dos envolvidos no mesmo, possibilitando inclusive a substituição de envolvidos sem perdas em produtividade.

3.5 CONCLUSÕES

O autor Alistair Cockburn usou a seguinte definição, que, na opinião do autor deste trabalho é muito apropriada, para “fazer engenharia”. Ele define: “vamos considerar engenharia como “pensar e fazer trocas”. Estas são frases apropriadas. Gostaríamos que nossos desenvolvedores de *software* pensassem, e que entendessem as trocas que optam por fazer” (Cockburn, 2000).

Devemos então “fazer engenharia” também na hora de escolher um processo adequado à equipe e ao projeto. Devemos entender os prós e contras de cada processo, e escolher aquele que mais se adapta ao projeto, ao desenvolvimento e à equipe.

Se conseguirmos encontrar um bom ponto de balanço nesta escolha, possivelmente teremos em mãos um bom processo para trabalhar.

4 METODOLOGIAS DE DESENVOLVIMENTO

Alistair Cockburn (2002) define o termo metodologia como “uma série de métodos e técnicas”, onde método se refere a “procedimentos sistemáticos”. O mesmo autor usa também outra definição, no mesmo livro: “uma metodologia é o conjunto de convenções em que o seu grupo concorda”. Já Martin Fowler (2003) diz que “metodologias impõem um processo disciplinado no desenvolvimento de *software*, com o objetivo de torná-lo mais previsível e mais eficiente”. E, finalmente, a *Wikipedia* (2007d) diz que “uma metodologia é um conjunto estruturado de práticas (...) que deve ser repetível durante o processo de produção do *software*”.

Logo, seguindo estas definições, pode-se dizer que uma metodologia de desenvolvimento é um conjunto de práticas e convenções, adotadas pela equipe para disciplinar e ditar os rumos de um processo.

Neste capítulo aprofunda-se um pouco mais este assunto, diferenciando e categorizando os principais tipos de metodologia. Também são apresentadas algumas das metodologias mais usadas no mercado. E finaliza-se efetuando uma breve conclusão do que foi discutido.

4.1 CLASSES DE METODOLOGIAS

A classificação mais encontrada na literatura especializada, como Cockburn (2002) e Fowler (2003), é a seguinte:

- Metodologias Tradicionais, ou “pesadas”;
- Metodologias Ágeis, ou “leves”;

A seguir, discorre-se brevemente sobre cada uma destas classes.

4.1.1 METODOLOGIAS TRADICIONAIS

Martin Fowler (2003) se refere às metodologias tradicionais como “metodologias de engenharia”, pois diz que elas se baseiam em práticas da engenharia, como foi discutido no capítulo anterior. Estas metodologias, como o

modelo *Waterfall* (ou “cascata”) e o Espiral, são essencialmente predeterminantes, e são chamadas de “tradicionais” por seguirem as idéias formuladas para metodologias nos primórdios da engenharia de *software*, ou seja, funcionam do jeito que se pensava que deveria se gerenciar desenvolvimento: como se faz na engenharia.

Esse tipo de metodologia valoriza o processo, a formalidade, a documentação e o planejamento inicial de um projeto. Os envolvidos na atividade de codificar são tratados como recursos, facilmente substituíveis, pois recebem, teoricamente, tudo “mastigado”, simplificando muito esta atividade (Fowler, 2003).

Um aspecto interessante desta categoria de metodologia é o fato de rejeitar mudanças. Este comportamento é facilmente justificado: como o planejamento é feito antes da codificação, e de qualquer protótipo ou interação do cliente com o *software* desenvolvido (período onde geralmente o cliente descobre o que realmente quer do *software*), é natural que modificações nesse planejamento atrapalhem muito o cronograma de um projeto regido por metodologias deste tipo.

A burocracia gerada por metodologias deste tipo faz com que elas sejam classificadas como metodologias pesadas (Cockburn, 2002), levando muitas vezes à falta de produtividade, pois há muita coisa para se fazer para seguir a metodologia corretamente (Fowler, 2003).

O uso deste tipo de metodologia justifica-se em projetos de grande porte, onde a equipe de desenvolvimento não se encontra no mesmo espaço, muitas vezes nem na mesma cidade, estado ou país. Neste caso valoriza-se muito a documentação como forma de comunicação entre equipes (Cockburn, 2002). Em projetos mais críticos uma metodologia mais pesada possibilita gerenciar dezenas, centenas de colaboradores, e o planejamento inicial viabiliza a previsão de maiores erros antes mesmo da fase de implementação iniciar.

São exemplos de metodologias desta classe: *Rational Unified Process* (ou RUP), *Waterfall* e suas variantes, Espiral, entre outras.

4.1.2 METODOLOGIAS ÁGEIS

O surgimento deste tipo de metodologia foi uma resposta a esta “burocratização” do desenvolvimento de software. Mas a idéia não é voltar à Idade Média (que, no caso de software, seria o período conhecido como “Crise do *software*”), mas sim propor metodologias mais simples, com menos artefatos e passos a seguir. Ou seja, mais “leves”.

Esse tipo de metodologia tem prioridades contrárias às suas antepassadas: incentivam “entendimento invés de documentação, disciplina invés de processo e perícia invés de formalidade” (Cockburn, 2002). São metodologias normalmente “baseadas em pessoas”, onde o conhecimento não fica todo armazenado na documentação, e sim na comunicação entre os membros envolvidos.

A proposta é, por ser mais leve, proporcionar à equipe mais tempo para desenvolver, com menos interrupções e papelada para preencher.

Normalmente criadas para que sejam incrementais e iterativas, e com o planejamento ocorrendo entre estas etapas, mudanças nos requisitos têm um impacto muito menor que nas suas contrapartes predeterminantes. O impacto de erros também é reduzido desta mesma forma. O desenvolvimento agora se torna algo dinâmico, maleável, rápido. Em teoria, pelo menos.

O principal problema deste tipo de metodologia é justamente depender demais da equipe. Algumas metodologias desta categoria exigem uma equipe mais madura de desenvolvedores para suceder. E, agora, um membro da equipe passa a ser um pedaço do acervo da especificação do projeto, pois com ele fica o conhecimento sobre o mesmo. Conhecimento esse que nem sempre é documentado claramente no código-fonte.

Esse tipo de metodologia, justamente por ser “centrada em pessoas” (Cockburn, 2002) torna os membros de um projeto muito menos substituíveis.

Também é exigido um nível muito maior de disciplina e perícia, pois o desenvolvedor agora tem maior liberdade para usar as técnicas e ferramentas que desejar em certas etapas do processo. Cabe ao desenvolvedor também a

especificação e planejamento durante uma etapa, iteração, incremento, *sprint*, ou o que for.

Metodologias deste tipo têm um ponto muito interessante, e crucial para o sucesso de um projeto: participação constante do cliente. Como o *software* é construído pedaço por pedaço, o cliente tem agora a possibilidade de ir moldando o sistema a seu gosto, como um escultor de cerâmica, participando ativamente de todo o processo.

Como se pôde notar, outro ponto crucial é a comunicação. Em metodologias deste tipo ela flui com muito mais facilidade, pois é feita normalmente face a face. Entretanto, este é o motivo principal para que o uso destas metodologias se restrinja a equipes pequenas. O *Extreme Programming* (ou XP), por exemplo, é projetado para o uso em equipes de até 12 pessoas, preferencialmente lotadas no mesmo espaço físico. Utilizar uma metodologia baseada em comunicação em equipes onde a comunicação toma muito tempo (diferentes andares, prédios, bairros, cidades, etc.) pode tornar o projeto proibitivo, inviável.

Metodologias como o XP são então usadas em projetos onde uma equipe relativamente pequena e produtiva é mais eficiente que uma equipe de médio porte com uma metodologia pesada.

Fazem parte desta categoria o XP, *Scrum*, entre tantas outras.

4.2 O ÁGIL CONTRA O PREDETERMINANTE

O já referenciado autor Alistair Cockburn (2002) cunhou o termo “peso” de uma metodologia. Ele define este “peso” como sendo “o produto do tamanho com a cerimônia” de uma metodologia. Vejamos abaixo uma melhor explicação de alguns dos termos utilizados pelo autor, pertinentes a este trabalho.

- Tamanho: “o número de elementos de controle em uma metodologia”. Esses elementos de controle, aos quais o autor se refere, podem ser: atividades, medidas de qualidade, artefatos, padrões, etc. Sendo assim, uma metodologia “grande” teria muitos artefatos, atividades,

medidas e/ou padrões, enquanto uma metodologia dita “pequena” possui poucos dos ditos elementos de controle;

- Cerimônia: “quantidade de precisão e tolerância em uma metodologia”. Corresponde a quantidade de “burocracia” em um desenvolvimento. Em sistemas de maior risco, propõe-se o uso de uma maior cerimônia para um maior planejamento dos problemas, revisões mais detalhadas dos artefatos, etc. Pouca cerimônia, indicado pelo autor para projetos com menor risco, possibilita uma margem maior para possíveis erros, pois a precisão dos artefatos gerados será bem menor, e o planejamento anterior também. Porém, com menos burocracia, o processo anda mais rápido, pois o tempo perdido no planejamento anterior diminui drasticamente;
- Peso: como já colocado anteriormente, “o produto do tamanho pela cerimônia”;
- Suficiência: uma metodologia suficiente é aquela que procura utilizar o mínimo possível de elementos de controle para produzir *software* de qualidade, procurando cumprir as duas metas do “jogo cooperativo”: desenvolver *software*, e deixar uma posição vantajosa para a próxima partida.

Ou seja, se uma metodologia possui um grande tamanho e uma grande cerimônia envolvida em cada elemento de controle, o processo será extremamente mais “pesado” em comparação a uma metodologia com pouca cerimônia e com poucos elementos de controle, ou mais "leve”. Conclui-se também que uma metodologia “suficiente” possuirá apenas os elementos de controle necessários, e nada mais.

Cockburn diz também que: “uma metodologia ágil[2] é leve e suficiente”. O autor Martin Fowler concorda com Cockburn em (Fowler, 2003), e diz também que

[2] De acordo com a *Wikipedia* (Wikipedia, 2007c), as metodologias ágeis eram chamadas de *lightweight*, ou leves. Em 2001 um grupo de membros da comunidade de metodologias ditas “leves” se reuniu e cunhou o termo metodologia “ágil” para as suas metodologias. Muitos destes membros vieram a formar, pouco tempo depois, a Aliança Ágil, ou *Agile Alliance*, uma entidade sem fins lucrativos que promove a filosofia ágil de desenvolvimento.

uma metodologia predeterminante é normalmente mais pesada e com mais "resíduos[3]".

A escolha de qual utilizar depende de várias coisas, sendo as mais importantes: o tamanho do projeto, os riscos envolvidos e características da equipe.

Um projeto maior, com uma grande equipe e grandes riscos envolvidos exige uma maior burocracia, e comunicação através de artefatos, possivelmente devido a separações geográficas na equipe. Já projetos menores, com pouca complexidade, equipe reduzida, lotada em um único ambiente, exigirão uma burocracia muito menor. Nestes casos a comunicação flui com mais facilidade quando a equipe conversa face a face, dispensando assim o uso de comunicação escrita e/ou por meio de artefatos.

Conclui-se, então, que a escolha entre qual tipo de metodologia utilizar deve ser feita após uma análise do projeto, da equipe e dos riscos envolvidos, para que se trabalhe com suficiência para a resolução do problema.

4.3 A ESCOLHA DE UMA METODOLOGIA

Baseado nas características do laboratório levantadas no capítulo 2 e nos problemas encontrados no processo de desenvolvimento (capítulo 3) pode-se concluir algumas das características desejadas em uma metodologia para implantar. São elas:

- Facilidade de aprendizado: sugere-se uma metodologia simples, leve, com pouca cerimônia, e poucos artefatos, para que tenha uma curva de aprendizado muito pequena, tendo em vista que muito provavelmente será a primeira experiência dos envolvidos com qualquer metodologia de desenvolvimento;
- Considerando a pouca experiência da equipe, se conclui que a mesma tenha pouca ou nenhuma experiência com geração de artefatos,

[3] O autor Alistair Cockburn define resíduos em (Cockburn, 2002) como sendo artefatos e atividades não aproveitadas posteriormente ou com pouca ou nenhuma utilidade dentro de um processo.

acentuando a necessidade de uma metodologia de pouco tamanho, ou seja, com poucos artefatos;

- A metodologia deve ter etapas bem definidas, que devem ser seguidas sempre: isto facilita o aprendizado e a padronização dos trabalhos, característica desejada no laboratório. Esta característica, se bem aplicada, também facilita o planejamento e a estimativa de prazo das tarefas;
- Facilidade na comunicação: por ser uma equipe pequena, lotada em um mesmo ambiente, se deseja que a troca de idéias e soluções flua com facilidade e rapidez entre a equipe, facilitando o aprendizado e incentivando a padronização do desenvolvimento através de uma propriedade comunitária do código-fonte;
- Padronização do código e documentação: deseja-se que o desenvolvimento não dependa demais da equipe que o desenvolve atualmente, para isto é proposto que a metodologia escolhida priorize um código bem escrito, de acordo com padrões definidos pela equipe, e que blocos importantes do sistema sejam documentados para futura referência, de novos recursos humanos ou de pessoas já envolvidas no desenvolvimento;
- Acompanhamento contínuo do desenvolvimento: é desejado um acompanhamento permanente dos projetos do laboratório. Desta forma podem ser detectados os possíveis problemas no desenvolvimento de um projeto com mais rapidez, agilizando a resolução destes problemas e evitando o efeito acumulativo, conhecido como “bola de neve”, em que os problemas vão se acumulando, atravancando todo o desenvolvimento.

Estes são os pesos definidos para serem utilizados na escolha da metodologia. Utilizando estes parâmetros é possível, através de uma análise das metodologias existentes, e dos conhecimentos adquiridos, escolher a mais adequada para os projetos do LATIN. A seção seguinte apresenta algumas das metodologias mais utilizadas atualmente e o capítulo 5 fornece uma comparação

entre todas as metodologias analisadas e discute a escolha final de uma metodologia para o laboratório.

4.4 ANÁLISE DE METODOLOGIAS

Nesta seção são abordadas algumas das metodologias mais conhecidas e mais utilizadas de cada uma das duas classes apresentadas neste capítulo. São analisadas suas principais características, vantagens e desvantagens.

O objetivo é detectar pontos positivos nestas metodologias para traçar comparações, e possivelmente aplicar as práticas que levam a estes pontos positivos à metodologia implantada no laboratório. Para isto deve-se atentar aos princípios de suficiência e agilidade desejados.

4.4.1 *RATIONAL UNIFIED PROCESS* (RUP)

O *Rational Unified Process* (Processo Unificado da Rational, ou RUP), é um processo proprietário, desenvolvido pela *Rational Software Corporation*[4], para complementar a *Unified Modeling Language* (Linguagem de Modelagem Unificada, ou UML). Apesar de ter sido desenvolvido visando Orientação a Objetos, este processo não obriga o uso deste paradigma em específico (Abrahamson et al. 2002).

4.4.1.1 O Processo

O RUP é divido em quatro fases (ou etapas) principais, chamadas de Concepção, Elaboração, Construção e Transição. Cada uma destas fases é então dividida em iterações (como mostra a figura 4.1) que podem durar de algumas semanas a alguns meses. Como na maioria dos processos iterativos, ao final de cada iteração se deve ter produzido um protótipo funcional ou artefatos do *software* em desenvolvimento. Neste processo todas as fases geram artefatos, que serão utilizados nas fases seguintes, que possibilitam o acompanhamento dos trabalhos, e que documentam o projeto.

---

[4] A *Rational Software Corporation* foi comprada pela IBM numa transação de 2,1 bilhões de dólares em 2002 (The Register, 2002), sendo assim. Agora é chamada IBM *Rational*.

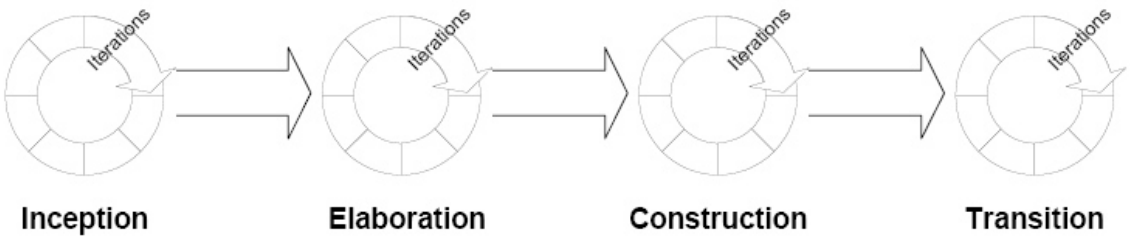


Figura 4.1: Fases do RUP.
Fonte: IBM Rational (2007)

A seguir é explicado brevemente, de acordo com Kruchten (1998), cada uma destas fases:

- Concepção: nesta fase é definido o escopo do problema, de acordo com as necessidades de cada interessado no projeto (os *stakeholders*). Nesta fase também são propostas possíveis arquiteturas e identificados os principais casos de uso do sistema.
- Elaboração: considerada pelo autor deste trabalho a fase mais importante do processo. Esta propõe a modelagem e construção da arquitetura do sistema, bem como a definição dos requisitos e planos necessários, e a maioria dos atores e casos de uso. É também nesta fase que é decidido o suporte a ferramentas de automação (artifício este que é recomendado pelo RUP), e que são descritos os ambientes de desenvolvimento, o processo, e a infra-estrutura a ser utilizada. Enfim, ao final desta fase é possível se realizar uma análise de todo o projeto, para que se conclua sobre sua viabilidade, os riscos envolvidos e custos.
- Construção: esta etapa consiste em programar e testar o *software*, de acordo com o projetado, mantendo-se a qualidade de *software* e os prazos estabelecidos na etapa de elaboração.
- Transição: esta etapa tem ênfase na implantação do sistema, sendo assim, só é atingida quando o *software* está “maduro” o suficiente para isto. É nesta fase que são realizados novos *releases* do *software*, que são desenvolvidos manuais e documentação para o usuário, que é

oferecido treinamento, que ocorrem manutenções para correção de *bugs*[5], etc.

O RUP possui também nove *workflows* (Kruchten, 1998) ou disciplinas (Kroll, 2003). Esses *workflows* ocorrem em todas as etapas, em paralelo, e servem para categorizar as tarefas que devem ser executadas em cada fase. São estes *workflows:* Modelagem de Negócios, Requisitos, Análise e Projeto, Implementação, Testes, *Deployment*, Configuração e Controle de Mudanças (ou Versões), Gerência do projeto e Ambiente. A figura 4.2 demonstra a distribuição de cada *workflow*/disciplina nas etapas e em um conjunto proposto de iterações.

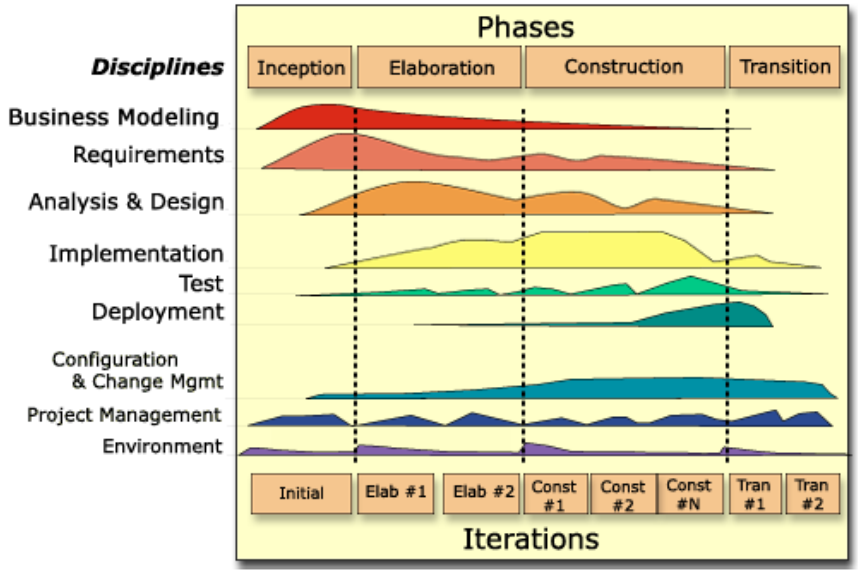


Figura 4.2: RUP: *Workflow* x Fases.
Fonte: IBM *Rational* (2007)

4.4.1.2 **Papéis e Responsabilidades**

Os papéis e responsabilidade são definidos de acordo com a atividade a ser executada. O RUP define 30 papéis, chamados de *workers* (trabalhadores) (Abrahamson, 2002).

Como explicado na seção anterior, as tarefas são categorizadas de acordo com os *workflows*. O mesmo acontece com os papéis, visto que são

[5] *Bug* é um erro no funcionamento comum de um software, também chamado de falha na lógica programacional de um programa de computador, e pode causar discrepâncias no objetivo, ou impossibilidade de realização, de uma ação na utilização de um programa de computador. (*Wikipédia*, 2007)

definidos de acordo com as tarefas. Cada *workflow* possui então, o seu próprio conjunto de *workers*.

A maioria destes *workers* possui papéis bem comuns, e bem específicos, como, por exemplo, Arquiteto, Projetista, Gerente de Configuração, entre outros (Abrahamson, 2002).

4.4.1.3 **Princípios do RUP**

O RUP se baseia nos princípios (ou práticas) descritos a seguir. Estas práticas devem ser a base da filosofia de um praticante do processo. (Kruchten, 1998)

- Desenvolvimento Iterativo de *Software*;
- Gerência de Requisitos;
- Uso de arquiteturas baseadas em Componentes;
- Modelagem Visual de *Software* (utilizando UML, basicamente);
- Garantir a Qualidade do *Software* (através de testes);
- Controle de versões do *software*.

4.4.1.4 **Conclusões**

Como visto nas seções anteriores, o RUP é um processo grande e burocrático, se aplicado como descrito nos manuais da IBM *Rational* e nos diversos livros que tratam do assunto. O RUP propõe um grande número de artefatos (mais de 100, embora nem todos sejam realmente necessários) e um grande número de papéis.

O RUP, entretanto, pode ser modificado e ajustado às necessidades do projeto. O único problema é que há pouca documentação quanto a isto, ficando totalmente a cargo do usuário do processo decidir o que deve ser usado ou não, aumentando muito o esforço de implantação.

O RUP, por ser uma metodologia pesada, tem um processo muito bem definido, e extensa quantidade de artefatos que documentam o projeto. Sendo assim, o RUP se adapta muito bem a projetos maiores ou mais críticos, em comparação com o tipo de projetos que as metodologias ágeis cobrem.

4.4.2 *EXTREME PROGRAMMING* (XP)

Hoje em dia é impossível falar de Metodologias Ágeis sem pensar em XP. E, realmente, a história de um está intimamente relacionada com a de outro, já que a XP foi uma das primeiras metodologias chamadas ágeis existentes[6], e com certeza é a mais popular, mais utilizada e mais consagrada da categoria.

O XP partiu de um conceito de "simplesmente uma desculpa para realizar o trabalho" (Haungs, 2001), e utilizava conceitos e práticas que são consideradas óbvias, ou senso-comum, como filosofia. O nome *Extreme Programming* (Programação Extrema) vem do fato de que estas práticas e princípios são levados ao extremo (Beck, 1999a).

4.4.2.1 **O Processo**

O ciclo de vida do processo do XP se baseia em seis fases principais: Exploração, Planejamento, Iterações para o *Release*, Produção, Manutenção e Morte. A figura 4.3 demonstra a organização do processo.

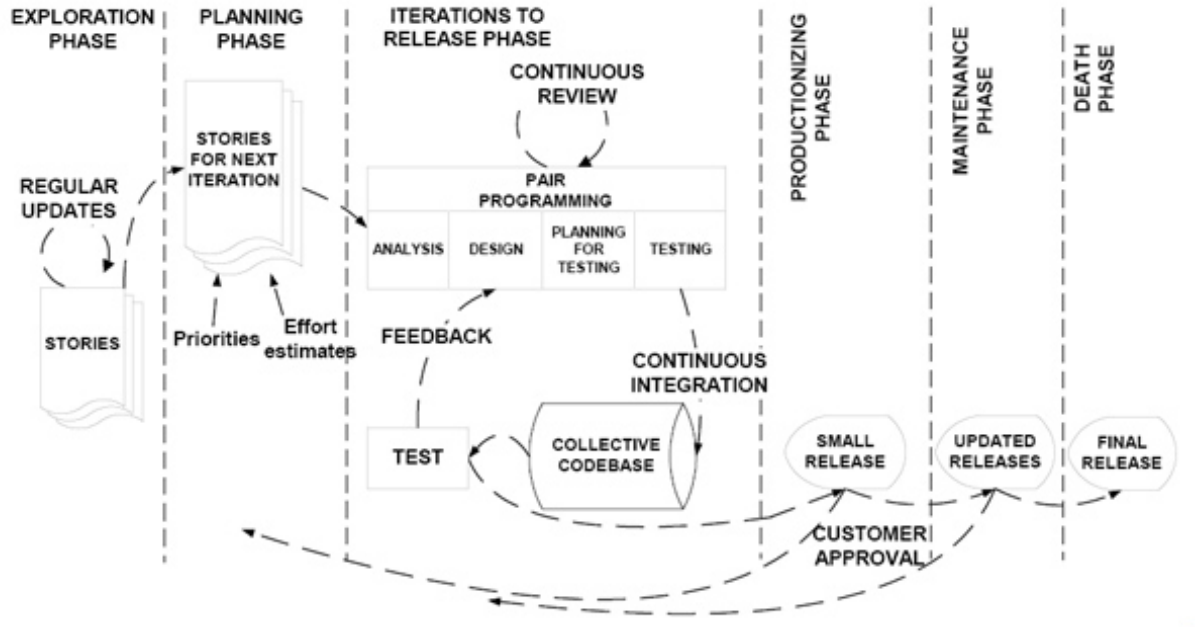


Figura 4.3: As fases do XP
Fonte: Abrahamson et al., 2002.

[6] De acordo com (Abrahamson et al., 2002), a primeira metodologia chamada ágil foi a *Dynamic Systems Development Method*, ou DSDM. Esta foi desenvolvida na década de 90 por um consórcio chamado *DSDM Consortium* (Wikipédia,2007e).

A seguir uma breve descrição de cada etapa, baseada nas descrições do criador da metodologia, Kent Beck, em (Beck, 1999b):

- Exploração: similar às duas primeiras fases do RUP, esta etapa consiste em “capturar” dos clientes as principais funcionalidades do sistema (escritas em *Story Cards*, onde cada estória descreve uma funcionalidade do sistema), enquanto a arquitetura do sistema é proposta e prototipada pela equipe de desenvolvedores. A duração desta fase depende mais da familiaridade dos desenvolvedores com a tecnologia a ser utilizada do que da complexidade do sistema.
- Planejamento: esta etapa demora alguns poucos dias. É nesta etapa que é definida a prioridade da implementação de cada estória, bem como estimados os esforços para suas realizações. É criada também uma agenda com o planejamento do primeiro *release* do sistema, que não deve exceder, em duração, o período de dois meses.
- Iterações para o *Release*: esta etapa é dividida em diversas iterações para a implementação do *release*, planejado na etapa anterior. Porém, a primeira iteração deve montar toda a arquitetura do sistema. Isto é feito desenvolvendo-se as histórias que forçam a implementação da arquitetura. Ao final de cada iteração, os testes funcionais (definidos pelos clientes) são rodados e o sistema deve estar funcional e pronto para produção.
- Produção: nesta etapa são executados mais testes sobre o sistema, e novas mudanças (*bugs*, novas funcionalidades, etc.) podem ser encontradas. Nesta fase decide-se se estas mudanças serão implementadas no *release* atual ou documentadas para implementação em um *release* futuro. Nesta etapa as iterações podem ser reduzidas.
- Manutenção: após o primeiro *release* entrar em produção, pode-se tornar necessário o suporte ao cliente. Nesta etapa este aspecto deve ser contemplado. Lembrando que a equipe ainda continuará trabalhando nas novas iterações. Isto pode acarretar na mudança da

equipe envolvida, com a contratação de mais pessoal para trabalhar com suporte, ou acarretar no acúmulo de funções em alguns, ou todos os membros da equipe de desenvolvimento. (Abrahamson et al., 2002)

- Morte: esta etapa pode ser alcançada quando o sistema está estável o suficiente, ou seja, não há mais histórias para implementar e os clientes estão satisfeitos com o *software* desenvolvido. Ou quando não há mais recursos para manter o desenvolvimento. De um modo ou de outro, esta etapa sinaliza o final do projeto.

4.4.2.2 **Papéis e Responsabilidades**

O XP possui, se comparado ao RUP, um número pequeno de papéis dentro do processo. Mas, se tratando de um processo ágil, para projetos relativamente pequenos, isso é algo já esperado. Porém, esses papéis não deixam de ser bem específicos. Temos o Cliente, o Programador, o Gerente (ou *Big Boss*) e o Testador. Mas, além destes, o XP possui alguns papéis que o diferenciam um pouco dos demais processos. Estes papéis são brevemente explicados a seguir (Abrahamson et al, 2002):

- *Tracker* (ou rastreador): Tem a função de captar um *feedback* dentro do processo. Ele que verifica se os prazos e esforços planejados estão sendo cumpridos e realizados dentro do previamente combinado. Com os dados obtidos ele pode também propor alterações e mudanças no processo.
- *Coach* (ou Técnico): Ele é o responsável pelo correto seguimento do processo pela equipe. Ele é o guia do XP para o time, e deve ser o membro com os maiores conhecimentos da metodologia.
- Consultor: é um membro externo que possui o conhecimento específico para o projeto. Normalmente um especialista em sua área.

4.4.2.3 **Práticas e Princípios**

Apesar de possuir um processo bem definido e um bom número de papéis com responsabilidades específicas, o ponto forte do XP são as práticas e princípios adotados no processo. São estas práticas e princípios que diferenciaram o

XP das abordagens clássicas, e que o tornaram tão popular, apesar do fato de que estas práticas foram tiradas de outras abordagens já existentes à época de sua criação (Beck, 1999a).

A seguir são abordados alguns dos principais princípios e práticas utilizados no XP:

- *Pair Programming* e espaço de trabalho aberto: programação em pares, duas pessoas em um computador, e, além disso, toda a equipe deve trabalhar no mesmo ambiente, de preferência em proximidade, para uma melhor comunicação;
- Desenvolvimento baseado em testes: os testes unitários são escritos antes do desenvolvimento. Também existem testes funcionais, descritos pelos clientes;
- Cliente "*on site*": o cliente deve estar sempre disponível para a equipe;
- Integração contínua de código: um novo trecho de código é inserido na base toda vez que ficar pronto. E o sistema é integrado e construído (compilado) várias vezes por dia;
- Propriedade coletiva do código: toda a equipe pode mexer em qualquer parte do código;
- Refatoração do código: simplificação do código, remoção de duplicações, e o aperfeiçoamento do que já está escrito, reescrevendo trechos do código sem alterar sua funcionalidade;
- *Design* simples: nada de padrões de projeto ou opções de *design* mais complexas do que o necessário para o momento. Código desnecessário deve ser removido;
- Padrão de código e regras do time: o código deve seguir um padrão definido pela equipe, facilitando a comunicação através do código. Se houver alguma mudança nesses padrões, bem como nas regras da equipe, esta deve ser avaliada de acordo com o impacto que provocará em todo o processo;

- Metáfora: um dos mais importantes aspectos do XP, a Metáfora ajuda na comunicação entre os desenvolvedores e os clientes, propondo um conjunto de metáforas que simulam uma história, que é escrita em conjunto por todos os envolvidos no projeto.

4.4.2.4 **Conclusões**

Bem diferente do RUP, o XP é um processo pequeno, bem mais leve, e bem menos formal. Mas, como a comunicação é feita basicamente face a face, o tamanho da equipe (e conseqüentemente do projeto), diminui bastante. O autor Alistair Cockburn (2002) definiu o XP como um processo para projetos de médio porte com até 12 pessoas, no máximo.

O maior problema do XP, se analisado de acordo com os problemas do LATIN, é a necessidade de uma equipe disciplinada e experiente. Como visto, a duração da primeira etapa do XP é altamente influenciada pelo conhecimento da equipe, nas tecnologias utilizadas e no problema a ser resolvido. O fato de haver pouco planejamento de como o código vai ser feito durante as etapas de implementação e manutenção agrava ainda mais esse problema. Com tudo isto, pode-se ver que o XP se baseia muito no talento e perícia dos envolvidos na equipe. Mas, como a equipe do LATIN é, em sua grande maioria, inexperiente, isto se tornaria um problema.

A maior vantagem do XP realmente é a agilidade do processo. O XP propõe um “ataque” direto, sem rodeios, ao problema em questão, e o *design* simples e o código padronizado reduzem em muito o esforço de uma futura manutenção no sistema desenvolvido, mesmo sem documentação.

4.4.3 SCRUM

O termo “*Scrum” é* um diminutivo para *Scrummage,* e vem do *rugby*. Este termo simboliza um meio de recolocar a bola em jogo, com trabalho de equipe (Abrahamson et al, 2002). A metodologia recebeu esse termo como nome por causa da prática de reuniões diárias com toda a equipe de, no máximo, 15 minutos, que, por ser filosoficamente similar ao artifício usado no *rugby* para recolocar a bola em jogo, foi também chamada de *Scrum.*

A idéia por trás do *Scrum* veio de uma adaptação para o desenvolvimento de *software* das técnicas para desenvolvimento e produção de produtos citadas por Takeuchi e Nonaka, em um artigo de 1986 chamado "*The New New Product Development Game*". Em 1993 Jeff Sutherland, John Scumniotales e Jeff McKenna desenvolveram o *Scrum* incorporando técnicas de gerência propostas por Takeuchi e Nonaka. Mas foi Ken Schwaber quem formalizou o *Scrum* em 1995, ajudando a popularizá-lo no mundo todo como metodologia ágil de desenvolvimento.

O *Scrum* agrada por que, se bem implantado, é um processo flexível, e que se apóia em técnicas de gerência e controle que ajudam a identificar potenciais problemas com antecedência.

4.4.3.1 **Processo**

O processo do *Scrum* é bastante similar ao do XP, porém mais condensado, possuindo apenas três fases, como pode ser visto na Figura 4.4.

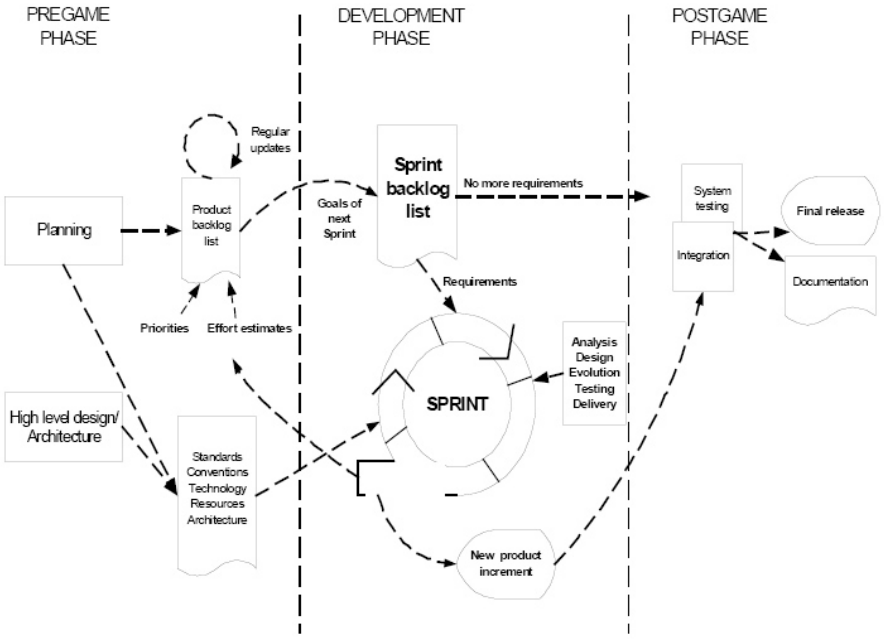


Figura 4.4: As fases do *Scrum*
Autor: (Abrahamson et al., 2002)

Estas fases são descritas, de acordo com Schwaber em (Schwaber, 1995), com mais detalhes abaixo:

- *Pre-game* (ou Pré-Jogo): similar as fases de planejamento e elaboração do XP, esta fase é, inclusive, divida em duas sub-etapas: Planejamento e Projeto de Arquitetura. A sub-etapa de planejamento é a responsável pela definição do *Backlog* do produto (explicado mais a frente) pelos clientes, um dos pilares de todo o processo *Scrum*. A cada iteração este *Backlog* deve ser revisado e atualizado. Já na sub-etapa de projeto da arquitetura, como o próprio nome diz, é projetada a arquitetura do sistema, baseado nas informações já inseridas no *Backlog*. Nesta sub-etapa também se faz os primeiros planos dos *releases*.
- Desenvolvimento (ou Jogo/*Game*): dividida em ciclos iterativos que variam de uma semana a um mês, chamadas de *Sprints*, esta etapa é a parte ágil do *Scrum* (Abrahamson et al., 2002). Nesta etapa o imprevisível é esperado, e através de técnicas de gerência e controle, todo o processo se adapta para contornar um problema ou assimilar uma nova técnica ou abordagem utilizada. Um *Sprint* (Figura 4.5) é dividido em sub-etapas tradicionais do desenvolvimento de *software*: requisitos, análise, projeto, evolução (implementação) e entrega. E durante um *Sprint*, a arquitetura de todo o sistema pode ser alterada.
- *Post-game* (ou Pós-Jogo): similar a fase “Morte” do XP, esta etapa é alcançada quando não há mais requisitos para implementar, ou problemas para resolver. Nesta etapa o *software* é preparado para o *release* final, passando por etapas de integração, documentação e testes.

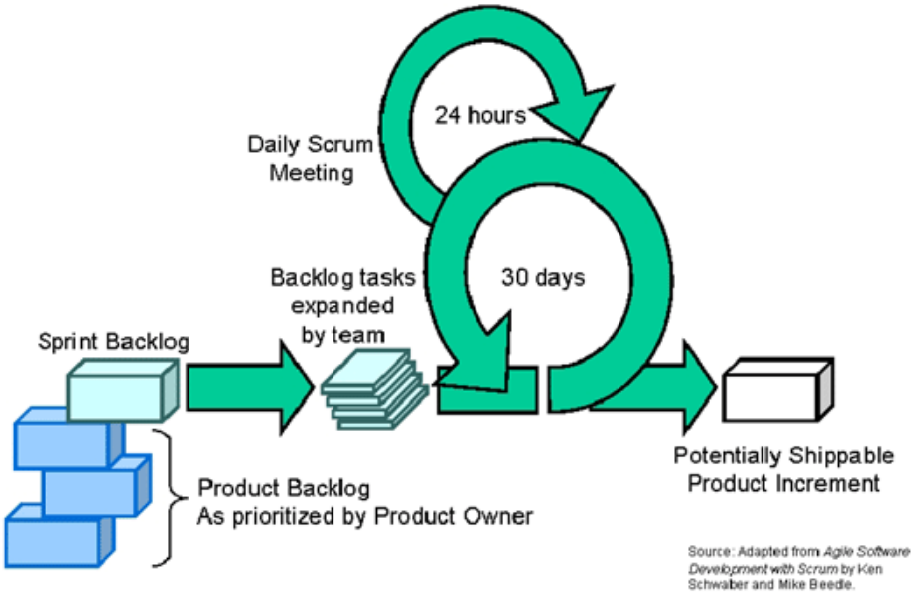


Figura 4.5: Um *Sprint* no *Scrum*
Fonte: Craig Murphy em (Murphy, 2007).

4.4.3.2 **Papéis e Responsabilidades**

O *Scrum* possui apenas seis papéis, cada um com responsabilidades bem definidas. O número reduzido de papéis faz do *Scrum* uma boa opção para equipes pequenas e médias. A seguir, são descritos cada um destes papéis, de acordo com os autores Schwaber e Beedle (Schwaber, Beedle, 2002):

- *Scrum Master* (Mestre *Scrum*): O *Scrum Master* é similar ao *Coach* do XP, sendo responsável pelo correto seguimento do processo e das práticas do *Scrum*. Ele é também responsável por adaptar a metodologia, removendo possíveis impedimentos que o processo ou alguma das práticas podem causar;
- *Product Owner* (Dono do Produto): É o responsável direto pelo *Backlog* do produto (explicado mais a frente). Ele é escolhido pelo cliente, pela gerência e pelo *Scrum Master*. Ele também participa nas estimativas de tempo e esforço dos itens do *Backlog*, e os transforma em tarefas a serem implementadas;
- Time *Scrum*: O time *Scrum* tem autoridade para decidir ações necessárias para o cumprimento das tarefas de um *Sprint*, bem como é responsável por planejá-lo através da criação do *Backlog* do *Sprint*.

Eles também revisam o *Backlog* do produto, indicando possíveis impedimentos que devem ser removidos do projeto;

- Cliente: O Cliente é quem patrocina o desenvolvimento, e participa do processo através do *Backlog* do produto, já que influencia diretamente na criação das tarefas que lá ficam;
- Gerente: Responsável pela tomada de decisão final no processo e do controle gerencial de todo o projeto. Participa na definição de metas e requisitos, na seleção do *Product Owner* e na redução do *Backlog*, juntamente com o *Scrum Master*.

4.4.3.3 **Práticas e Princípios**

O *Scrum* não indica nenhuma prática de programação específica, como o XP, por exemplo. Porém, possui um grande número de práticas, ou regras, para o controle gerencial do projeto e do processo em si. As principais práticas do *Scrum* são:

- *Backlog* do Produto: Trata-se de uma lista que define tudo que deve ser feito no sistema. Itens do *Backlog* podem ser: funcionalidades, ações, correção de *bugs*, *upgrades*, entre outros. Esta lista é organizada por prioridade, e deve ser atualizada constantemente, para que reflita sempre mudanças recentes nos requisitos, riscos associados, entre outras mudanças. Como dito anteriormente, o responsável pela manutenção deste artefato é o *Product Owner*;
- Estimar Esforço: Esta estimativa de esforço é traçada em cima de tarefas do *Backlog* do Produto por membros do(s) time(s) *Scrum* envolvidos e pelo *Product Owner*. Esta estimativa é sempre traçada de forma iterativa, de modo que seu valor seja atualizado sempre com base nas informações mais recentes sobre a tarefa estimada;
- *Sprint*: Trata-se de um procedimento de adaptação rápida a mudanças súbitas nas regras do jogo (sejam elas de requisitos, funcionalidades, tecnologia utilizada, recursos, etc.). A idéia é que o time do *Scrum* tenha uma organização própria para entregar código implementado, fazendo toda a parte de planejamento das tarefas do *Sprint*, criando

um *Backlog* só para o *Sprint*, e avaliando possíveis riscos e impedimentos através da reunião diária, chamada de *Scrum*. Um *Sprint* deve demorar um mês, aproximadamente. Além do *Backlog* do *Sprint*, e da reunião diária, o *Sprint* deve ter também uma Reunião de revisão, no seu último dia, onde todos os envolvidos no processo acessam o produto desenvolvido naquele período e traçam decisões sobre os rumos do projeto deste ponto em diante.

4.4.3.4 **Conclusões**

O *Scrum* se caracteriza por ser uma metodologia dinâmica, com uma grande flexibilidade para adaptar-se durante o desenvolvimento, diminuindo riscos e impedimentos que poderiam atravancar o processo. Deste modo, em ambientes de trabalho ou projetos com muitas mudanças, o *Scrum* é, provavelmente, a metodologia mais adequada.

O *Scrum* não possui práticas para codificação, mas nada impede que sejam utilizadas. Esta escolha deve ser feita pelos envolvidos no projeto. Existem até mesmo pesquisas e experiências sendo realizadas para integrar o XP com o *Scrum*, sendo que a gerência fica por parte do *Scrum* e as práticas de codificação e integração ficam com o XP (Abrahamson et al., 2002).

A idéia das reuniões diárias é saber o que foi feito no dia, e o que deve ser feito no próximo. Deste modo, fica fácil saber quais possíveis riscos e impedimentos surgirão e, prevendo-os com antecedência, é possível também solucioná-los com mais rapidez.

O *Scrum Master* deve ser um profundo conhecedor das práticas da metodologia e do processo *Scrum*, e deve servir como um guru para toda a equipe. Portanto, deve ter experiência na prática do *Scrum*. Isto pode ser um problema na implantação da metodologia em uma equipe composta apenas por iniciantes no uso de processos e metodologias.

4.4.4 A FAMÍLIA *CRYSTAL*

A família *Crystal* de metodologias consiste em um conjunto de metodologias, que partilham das mesmas bases e práticas, mas que se diferenciam

quanto ao seu “peso”. Assim, o aspecto mais importante é o modo de escolher qual a metodologia mais adequada para um projeto.

O autor Alistair Cockburn, famoso metodologista e desenvolvedor desta família de metodologias, acredita que a metodologia adequada é baseada no tamanho da equipe e nos riscos envolvidos no projeto, e criou então um método para a escolha da metodologia de acordo com a classificação do projeto quanto a estes dados. (Cockburn, 2002)

A classificação do projeto possui uma letra (que representa o risco envolvido) e um número (que representa o tamanho do projeto). As letras são C (perda de conforto), D (perda moderada de dinheiro), E (perda de dinheiro essencial) e L (risco de vida). Ou seja, de acordo com essa classificação um projeto D6 representa um risco de perda moderada de dinheiro e uma equipe de até seis programadores, e um projeto C6[7] representa um risco de perda de conforto dos usuários em uma falha do sistema e uma equipe de até seis programadores.

A classificação da metodologia é dada através de cores. Quanto mais escura a cor, mais “pesada” a mesma é. A escolha da metodologia apropriada de acordo com o projeto é proposta por Cockburn de acordo com a figura 4.6.

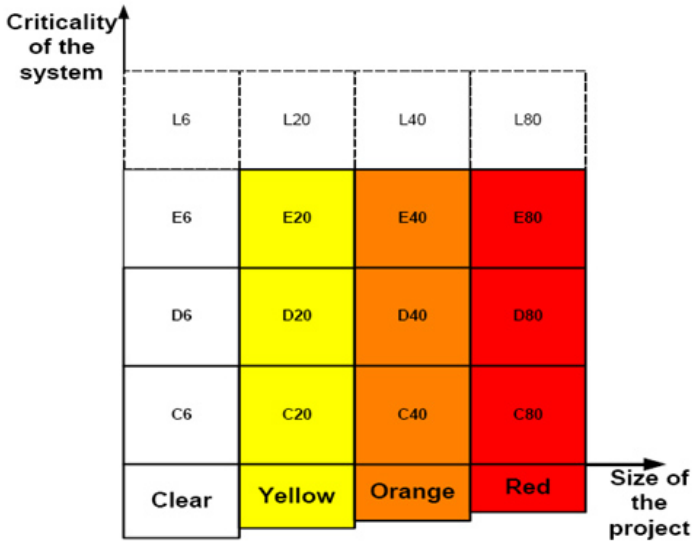


Figura 4.6: A escolha da metodologia *Crystal*.

Autor: (Abrahamson et al, 2002).

[7] O Projeto SINGU, carro-chefe do LATIN, possui uma classificação entre C6 e D6

Apesar de possuir diversas metodologias, todas possuem diversas características em comum, como, por exemplo, a abordagem iterativa, a ênfase na comunicação e na cooperação entre a equipe (Cockburn, 2002).

As metodologias da família *Crystal* desenvolvidas e utilizadas até hoje são a *Crystal Clear* e a *Crystal Orange*. Na seção seguinte, que explica em mais detalhes o processo, os papéis, com suas respectivas responsabilidades, e as práticas, serão discutidas as diferenças entre cada uma destas duas metodologias.

4.4.4.1 **O Processo**

Como visto na classificação anterior, a metodologia *Crystal Clear* é apropriada para projetos pequenos, de até seis desenvolvedores. Ela é adequada para projetos de baixo risco e os desenvolvedores devem compartilhar a mesma sala, devido às limitações na sua estrutura de comunicação (Cockburn, 2002). Já a *Crystal Orange* abrange projetos de médio porte, com até 40 colaboradores. Na *Orange*, os trabalhos são divididos em times multifuncionais diversos e há a criação de artefatos suficientes para a comunicação entre eles.

Ambas podem ser utilizadas para projetos um pouco mais críticos que o proposto por Cockburn, com algumas mudanças básicas para evitar alguns dos riscos envolvidos (Abrahamson et al., 2002).

Os processos das metodologias *Crystal* não possuem diferenças drásticas entre si, até por que tanto a *Clear* quanto a *Orange* não possuem um processo propriamente dito, apenas atividades, que devem ser executadas em incrementos de um a três meses (podendo chegar a quatro meses na *Orange*). Isto acontece por que o autor destas metodologias prioriza as pessoas envolvidas e não processos, focando-se apenas nas atividades que devem ser desempenhadas e deixando os ajustes do processo por conta dos envolvidos no mesmo (Fowler, 2003).

Em um incremento, como exposto anteriormente, diversas atividades são executadas. A figura 4.7 mostra um incremento da *Crystal Orange* com mais detalhes.

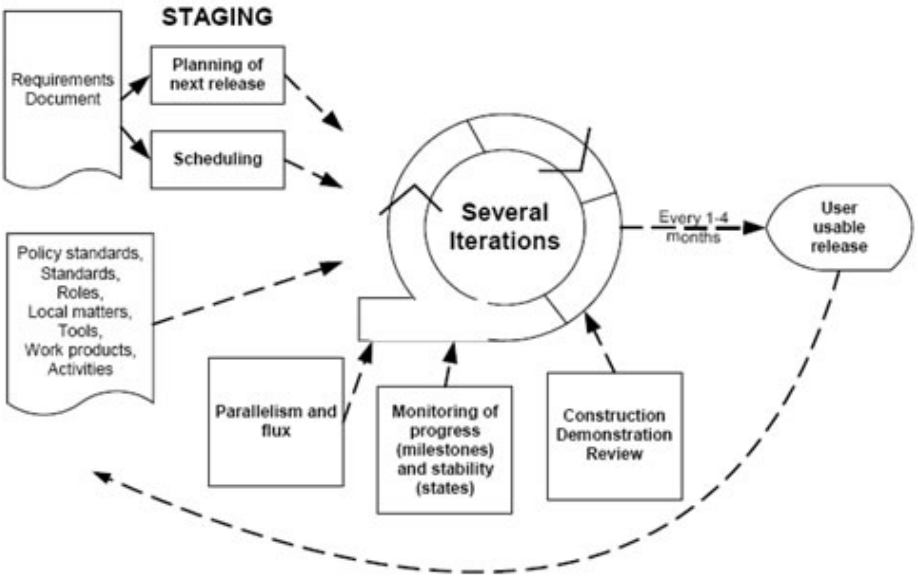


Figura 4.7: Um Incremento na *Crystal Orange*
Autor: (Abrahamson et al., 2002)

Na *Crystal Orange* a etapa em que o Documento de Requisitos é criado é obrigatória, já que é um documento importante para a gerência de vários times. Isto não é tão necessário na *Clear*, visto que é apropriada para apenas um time trabalhando em um mesmo local. Outras etapas, como Agendamento (*Scheduling*) e Revisões (*Reviews*) são mais aprofundadas na *Orange*, e mais brandas na *Clear*.

As práticas de paralelismo e fluxo, é claro, se aplicam apenas a *Crystal Orange*, já que na *Clear* não há múltiplos times, não havendo necessidade de paralelismo nas atividades.

4.4.4.2 **Papéis e Responsabilidades**

Em uma metodologia baseada em pessoas, invés de processos, os papéis, e suas respectivas responsabilidades, são aspectos muito importantes. Tanto a *Crystal Clear* quanto a *Crystal Orange* possuem diversos papéis em comum. É claro que, sendo uma metodologia mais pesada, é natural que a *Crystal Orange* possua mais papéis. Porém, as diferenças não param por aí: a principal diferença entre estas duas metodologias é o número de times que abrangem.

Enquanto a *Crystal Clear* não possui uma estrutura de comunicação adequada para portar mais de um time, isto não ocorre na *Orange*, que possibilita, e

obriga, o uso de diversos times multifuncionais (com profissionais de diversos papéis juntos).

Nas metodologias da família *Crystal*, uma pessoa pode assumir mais de um papel, acumulando funções.

Os principais papéis básicos, necessários para ambas as metodologias são:

- Patrocinador: quem financia o projeto;
- Usuários: que usam o *software* desenvolvido;
- Programador/Analista Sênior: membro mais experiente em programação e análise/projeto de sistemas;
- Programador/Analista: membro menos experiente, mas capaz de programar e projetar/analisar um sistema.

A *Crystal Orange*, entretanto, inclui um variado número de papéis a este conjunto, sendo a grande maioria de profissionais mais especializados, como Projetista de Interfaces com o Usuário, analista de banco de dados, mentor do projeto, arquitetos, analista de requisitos, analista de negócios, entre outros.

Entre tantos papéis, são ressaltados dois em (Abrahamson et al., 2002), que merecem um pouco mais de atenção. São eles:

- Escritor: Responsável pela produção de documentação externa, como manual de usuário, por exemplo;
- Analista de Negócios: É ele que negocia e se comunica com os usuários para obter o que deve ser especificado em termos de requisitos e interfaces, e também para revisar o *design* do projeto.

Um time de trabalho na *Crystal Orange* deve ser composto de no mínimo: Um analista de negócios, um projetista de interfaces com o usuário, dois a três programadores/analistas, um analista de banco de dados e, se possível, um testador (Abrahamson et al., 2002).

4.4.4.3 **Práticas e Princípios**

Diversas práticas e princípios são aplicados às metodologias da família *Crystal*, já que se tratam de metodologias ágeis. Nesta subseção são vistas as principais:

- Desenvolvimento Incremental: Como já visto, o desenvolvimento do projeto se dá através de incrementos;
- Políticas-Padrão (*Policy Standard*): são as práticas que devem ser executadas no desenvolvimento, tais como Desenvolvimento Incremental, Controle do Progresso do projeto via *Milestones*[8], envolvimento direto do cliente, testes automatizados regressivos, *workshops* para ajustes no produto e na metodologia (no início do incremento, no final, e opcionalmente no meio), entre outras. A única diferença nestas práticas em relação ao *Clear* e ao *Orange* é o tempo dos incrementos (1~3 meses no primeiro, 1~4 no segundo). Entretanto, outras práticas de outros processos, como o XP e o *Scrum*, podem também ser aplicadas no processo;
- Produção de Artefatos: as duas metodologias possuem alguns artefatos em comum, como manual, *test cases* (casos de teste), código de migração, modelos de objetos comuns e seqüência de releases. Porém, a *Crystal Orange* inclui diversos outros artefatos, visto que pode operar com vários times. Um exemplo é a Agenda do projeto e os relatórios de *Status*.
- *Local Matters* (ou Assuntos Locais): este aspecto do processo é praticamente igual para as duas metodologias. Ele propõe que *templates* de código, formatações, padrões para interfaces, entre outras coisas, devem ser especificadas pelo time. Isto também se aplica para outras práticas, praticadas individualmente por cada papel no processo, visto que nem a *Crystal Clear*, nem a *Crystal Orange* as definem;

[8] Um evento importante ou estágio no desenvolvimento. (Dictionary.com, 2007)

- Uso de Ferramental: Este é o aspecto em que as metodologias possuem mais diferenças, pois na *Clear* o ferramental de apoio é muito menor que na *Orange*. Na *Clear* é exigido compilador, sistema de versões, ferramenta para gerência de configuração e *Printing Whiteboards*[9]. Já no *Orange* tipos de ferramentas mais especializadas como: ferramentas para testes, comunicação, programação, desenho, entre outras, se fazem necessárias;
- Padrões: Na *Orange* é proposto o uso de padronização também nas convenções de projeto, notação, formatação e qualidade.

4.4.4.4 **Conclusões**

O autor Alistair Cockburn se destaca no mundo acadêmico por seu grande *know-how* em metodologias e processos de desenvolvimento, e aplicou, como continua aplicando até hoje, todo este conhecimento para desenvolver sua família de metodologias *Crystal*.

Pode-se notar que são metodologias que podem e devem ser ajustadas para se adequar ao projeto, evitando ter que adaptar o projeto à metodologia. Além disto, Cockburn ainda propõe o uso de determinada metodologia para determinado tipo de projeto, dando uma metodologia padrão para cada tipo como ponto de partida. Esta metodologia deve então ser aplicada ao projeto e, através de reuniões de acompanhamento e revisões, ajustada para o mesmo.

Esta idéia dá as metodologias da família *Crystal* um enorme dinamismo, e uma rápida resposta a mudanças.

Porém, exige dos envolvidos no projeto certa experiência no uso de processos e metodologias, pois não há como saber o que deve ser alterado em uma metodologia com clareza sem nenhuma experiência no assunto.

Além de tudo isto, a família *Crystal* é ainda uma família de metodologias em desenvolvimento, o que provoca uma relativa escassez de documentação e

[9] Um quadro branco, interativo, para anotações, que possibilita impressão. Neste contexto é utilizado para substituir documentos formais na forma de anotações que podem ser guardadas para posterior visualização. (*Wikipédia*, 2007b)

pesquisas sobre a mesma, se comparado com metodologias mais conhecidas como o XP e o *Scrum*.

4.4.5 EASYPROCESS (YP)

Em uso na Universidade Federal de Campina Grande (UFCG) desde maio de 2003, o *EasyProcess* (ou YP) foi desenvolvido por uma equipe de 9 pessoas, graduandos(as) do curso de Ciências da Computação da UFCG, num trabalho orientado pela Profa. Francilene Procópio Garcia. O projeto nasceu da observação de uma falta de metodologias de fácil aprendizado e projetadas para trabalhos acadêmicos, normalmente projetos de curta duração e baixa complexidade.

Mas, apesar de ter sido desenvolvido para a academia e para projetos curtos e simples, o YP é uma metodologia robusta e completa, com processo bem definido e bastante simples de aprender e implantar.

Esta subseção aborda com mais detalhes o YP: seu processo, papéis, práticas e princípios, com base em Garcia et al. (2007) e PET/UFCG (2007). Logo após, são discutidas as primeiras impressões sobre a metodologia.

4.4.5.1 O Processo

O YP é fortemente baseado em metodologias como RUP, XP e *Agile Modeling*, e seu processo é bastante parecido com a maioria dos processos ágeis existentes. Porém, como nenhum processo anterior conseguia se adaptar com perfeição ao ambiente acadêmico, se desenvolveu, então, um processo do zero.

4.4.5.1.1 As Fases do Processo

O processo do YP é composto de oito fases bem definidas. Este processo foi projetado para ser, acima de tudo, simples e completo. As fases do YP são, em sua maioria, pequenas e simples, e é exatamente essa coesão que o faz um processo tão simples. A figura 4.8 mostra uma síntese do fluxo do YP com mais detalhes.

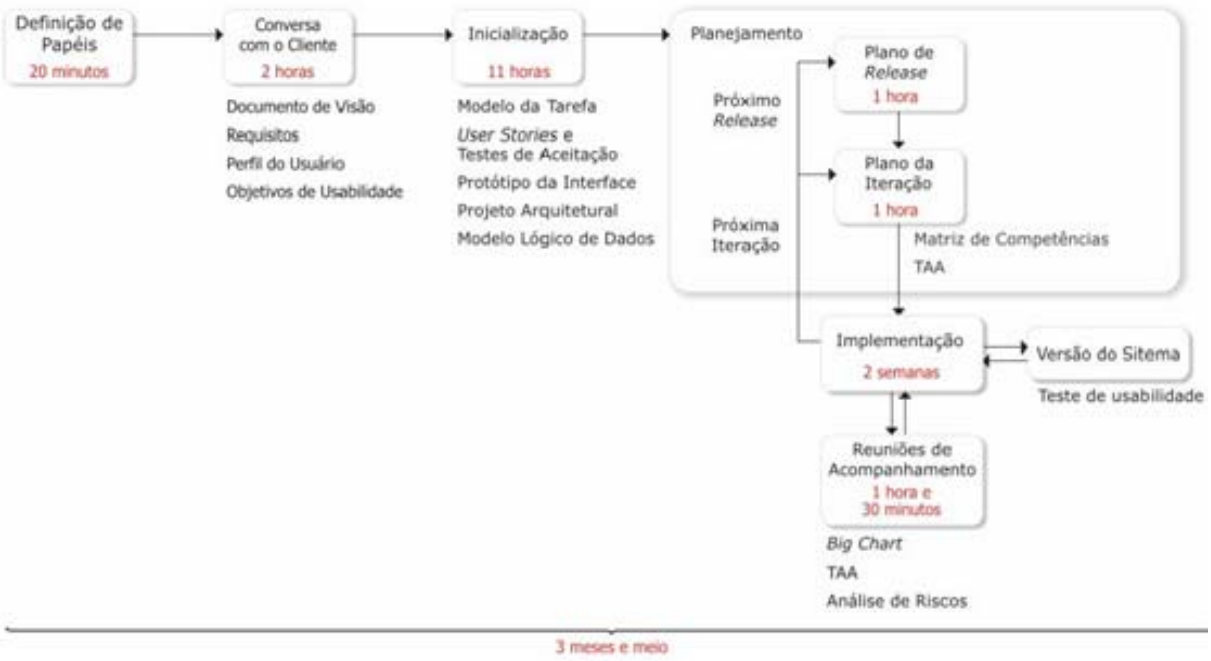


Figura 4.8: Síntese do Fluxo do Processo YP
Autor: PET/UFCG em (PET/UFCG, 2007).

A seguir, explicam-se cada uma das fases com mais detalhes:

4.4.5.1.1.1 Identificação do Escopo do Problema

Esta é a fase inicial do processo, e é nela que se pesquisa e se adquire conhecimentos necessários sobre o problema para que se possa então preparar um roteiro de perguntas para a Conversa com o Cliente. Esta etapa tem um período relativamente curto, e pode durar apenas o suficiente para que a equipe se situe em relação ao problema, podendo ser realizada em uma única reunião juntamente com a etapa seguinte.

4.4.5.1.1.2 Definição de Papéis

Nesta etapa é definido quem faz o quê dentro do processo. O YP possui diversos papéis (apresentados com mais detalhes adiante) que podem ser assumidos pelos membros da equipe. Estes papéis são cumulativos, ou seja, um membro pode ter mais de um papel no processo. Isto foi feito para contemplar

equipes pequenas. Esta fase é relativamente simples e por isso não costuma durar muito tempo, sendo que o tempo proposto pelos autores é de 20 minutos.

4.4.5.1.1.3 Conversa com o Cliente

Esta fase tem o objetivo de fazer com que os desenvolvedores e os clientes tenham um conhecimento em comum do sistema a ser desenvolvido. É nesta primeira reunião com o cliente que se captam informações como: requisitos funcionais e não-funcionais, perfil do usuário, objetivos de usabilidade, testes de aceitação, entre outros. Esta fase é caracterizada pela realização de uma reunião, que tem a duração de apenas algumas poucas horas e que serve de “pontapé inicial” para o projeto. Após esta etapa é então criado o Documento de Visão, primeiro artefato do processo (explicado com mais detalhes na seção seguinte).

4.4.5.1.1.4 Inicialização

Nesta etapa, similar a etapa de Elaboração do XP, é realizado um modelo da tarefa do sistema, para que então seja feito um protótipo da interface do sistema. Após a aceitação desta interface do sistema, são definidas, então, as *User Stories* com seus respectivos testes de aceitação. Enfim é desenvolvido um projeto arquitetural do sistema, bem como um possível modelo lógico, no caso de haver persistência de dados envolvida. Esta etapa gera diversos artefatos e conta com a presença do cliente na sua maior parte, durando então o suficiente para que se tenha uma idéia das tarefas a serem desenvolvidas pelos desenvolvedores com bastante clareza, bem como um protótipo da interface e um modelo de dados bem definido. O tempo proposto pelos autores é de 11 horas, mas possivelmente aumenta (ou diminui) de acordo com a complexidade do projeto.

4.4.5.1.1.5 Planejamento

Esta etapa, como pode ser vista na figura 4.8, é dividida em duas sub-etapas: o planejamento de *Release* e o planejamento de Iteração. Um *release* e uma iteração ocorrem em paralelo, sendo um *release* composto de “n” iterações. Os

autores propõem um *release* de quatro semanas dividido em duas iterações de duas semanas cada. Estas sub-etapas de planejamento têm propósitos diferentes:

- Planejamento de *Releases*: o planejamento de *release* consiste em definir o período e o escopo de um *release*, e então dividi-lo em iterações. Nesta sub-etapa são definidos quais *User Stories* serão implementados no *release* planejado.
- Planejamento de Iterações: no planejamento de iterações, as *User Stories* definidas no planejamento de *release* são divididas em atividades menores (quando necessário) e alocadas para os membros envolvidos. Nesta etapa também é montada/atualizada a Tabela de Alocação de Atividades (ou TAA).

4.4.5.1.1.6 Implementação

Esta é a etapa de criação do principal artefato de todo o projeto: o código-fonte do sistema. Nesta etapa são aplicadas as principais práticas usadas no desenvolvimento com metodologias ágeis. Estas práticas e princípios serão abordados mais a frente.

4.4.5.1.1.7 Reunião de Acompanhamento

Esta reunião, que ocorre semanalmente, serve apenas para o acompanhamento do projeto pelo Gerente. Similar à reunião do *Scrum*, mas um pouco mais formal, ela tem a participação de toda a equipe de desenvolvimento, e, quando coincidir com o final de uma iteração ou *release*, conta também com a presença dos clientes. Esta reunião serve também para atualizar o estado da Tabela de Alocação de Atividades e do *Big Chart* (explicados mais adiante).

4.4.5.1.1.8 Versão do Sistema

Similar às etapas finais de praticamente todos os processos estudados neste trabalho, esta etapa serve para a finalização e testes finais de uma eventual versão do sistema, que entrará em produção. Nesta etapa que são desenvolvidos

eventuais manuais de utilização ou documentação, executados testes de integração, entre outras tarefas relacionadas. É também nesta etapa que são realizados os testes de usabilidade na versão do sistema.

4.4.5.1.2 Principais Artefatos do Processo

Nesta seção são mostrados os principais artefatos utilizados durante o processo do YP. Se comparada com outras metodologias, o YP possui um número relativamente pequeno de artefatos, priorizando mais a comunicação entre os membros da equipe e a interação com o cliente. O conjunto de artefatos do YP é basicamente composto de elementos de controle, para facilitar a gerência de todo o processo, e o acompanhamento preciso do estado do desenvolvimento. O uso de cada artefato está relacionado com as etapas do processo na Figura 4.8.

4.4.5.1.2.1 Documento de visão

Construído após a conversa inicial com o cliente, este documento concentra as idéias gerais sobre o que o sistema se propõe a fazer.

Este documento deve ser escrito utilizando pouco, ou nenhum, linguajar técnico, de forma que seja de fácil entendimento, pois é algo como um contrato informal entre o cliente e os desenvolvedores, Ele deve sinalizar tudo o que foi combinado com o cliente no contato inicial.

Este documento deve conter informações como o perfil do usuário final do sistema, uma análise inicial de riscos do projeto, requisitos funcionais e não-funcionais, características gerais do produto, uma matriz inicial de competências dos envolvidos na equipe (feita a partir da definição de papéis realizada), objetivos de usabilidade (para a geração do teste final de usabilidade), e mais outras informações que possam ajudar a equipe.

Este documento pode ser usado também como um comparativo ao final do projeto, para saber o quão distante do planejado ficou o que foi concretizado.

4.4.5.1.2.2 Modelo da tarefa

A partir da análise da(s) tarefa(s) que o sistema deve executar, é gerada esta representação hierárquica de como a tarefa deve ser executada pelo usuário. Este modelo já deve levar em conta possíveis objetivos de usabilidade e perfis de usuário.

Esta modelagem facilita aos desenvolvedores a implementação da tarefa, pois simplifica sua divisão em sub-tarefas.

Os autores incentivam o uso de formalismos para a realização desta modelagem, sugerindo o uso do TAOS (*Task and Action Oriented System*, ou Sistema Orientado a Ações e Tarefas).

4.4.5.1.2.2.1 *Task and Action Oriented System* (TAOS)

O TAOS é um formalismo inicialmente criado para modelagem de Sistemas Baseados em Conhecimento, e foi validado na área de Biologia Molecular (no campo da Inteligência Artificial).

No modelo, as tarefas são decompostas em sub-tarefas ou ações de forma hierárquica. Cada sub-tarefa/ação na árvore é unicamente identificado e possui relacionamentos com outras sub-tarefas ou possui sub-tarefas filhas. Estes relacionamentos são regrados por operadores lógicos simples como: *OR*, *XOR*, *AND*, e operadores mais específicos: *SEQ* (indica que uma sub-tarefa ou ação deve ser executada em seqüência à outra), *SIM* e *PAR* (Simultânea e Paralela, respectivamente).

No exemplo da Figura 4.9 se pode observar com mais clareza a utilização do TAOS para modelagem de tarefas.

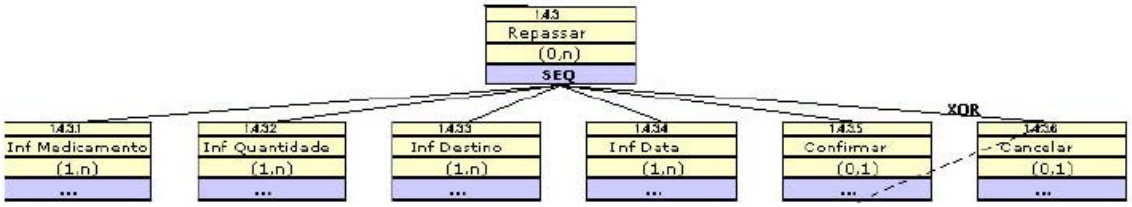


Figura 4.9: Exemplo do TAOS em uso.
Autor: PET/UFCG em (PET/UFCG, 2007)

4.4.5.1.2.3 *User Story* e Testes de aceitação

Similar aos *Story Cards* do XP, as *User Stories* (ou Histórias dos Usuários) se propõem a “contar uma hustória”, que seriam as funcionalidades do sistema. É com base nessas *User Stories* que são definidas as tarefas a serem desenvolvidas pela equipe de desenvolvimento do projeto.

É interessante que as principais *User Stories* do sistema sejam definidas logo no início do projeto, Porém, podem surgir durante o desenvolvimento diversas outras. Estas devem então ser definidas para que em iterações seguintes sejam divididas, se necessário, em tarefas que devem ser executadas pelos desenvolvedores.

Cada *User Story* deve ter no mínimo um Teste de Aceitação definido. Estes testes definem um conjunto de situações definidas pelo cliente para medir o sucesso ou não de uma determinada *User Story*. Só quando passar nos testes de aceitação é que uma *User Story* pode ser declarada finalizada.

4.4.5.1.2.4 Protótipo da Interface

O protótipo de uma interface consiste em uma representação gráfica de uma, ou mais, telas do sistema. No YP esses protótipos não passam de esboços simples, normalmente feito a lápis em uma folha de papel, para que não se dispense esforço na criação de uma interface que não contemple as funcionalidades desejadas pelo cliente.

Esses protótipos auxiliam também na comunicação entre os desenvolvedores e os clientes, pois mostram de uma maneira gráfica o que o sistema deve fazer.

4.4.5.1.2.5 Projeto Arquitetural

É um artefato (com gráficos e descrições relacionadas) que serve para modelar a estrutura do sistema, seus relacionamentos internos e externos (com outros sistemas e/ou subsistemas e módulos).

Toda a equipe de desenvolvedores deve participar da construção deste artefato, e ele só é considerado pronto quando validado pelo cliente. Esta estrutura projetada, se bem feita, dificilmente será modificada durante o projeto. Mas, se isto acontecer, o projeto arquitetural sempre poderá ser alterado, devendo ser validado novamente pelo cliente.

4.4.5.1.2.6 Modelo Lógico de Dados

Um artefato que mostra de maneira gráfica o modelo lógico dos dados utilizados em determinada tarefa, *User Story*, ou até mesmo na arquitetura do sistema. Este modelo é normalmente representado por um diagrama de classes (de análise, ou seja, sem métodos, apenas atributos) ou de banco de dados.

4.4.5.1.2.7 Tabela de Alocação de Atividades (TAA)

Esta tabela, como o nome diz, representa a alocação de atividades para cada desenvolvedor. É ela quem sinaliza quem faz o quê durante uma iteração.

Ela mostra uma descrição simples da atividade, quem é o responsável pela sua implementação, qual a estimativa de tempo, qual o status da tarefa, e, quando finalizada a tarefa, o tempo real despendido para a realização da mesma. A TAA mostra também quais os testes de aceitação associados à quais tarefas e o *status* de cada um destes testes (passou, não passou).

Esta tabela é o principal termômetro do gerente durante a reunião de acompanhamento mensal, pois é a partir dela que é possível diagnosticar tarefas problemáticas e/ou desenvolvedores com problemas para cumprir suas tarefas, por qualquer motivo que seja.

4.4.5.1.2.8 Big Chart

O *Big Chart* é o principal artefato gerencial de todo o processo.

Trata-se de um acumulador, em forma de gráfico, de métricas de todo o desenvolvimento do projeto. Ele marca a evolução das principais métricas que podem ser capturadas em cada reunião de acompanhamento, como por exemplo: número de classes, testes de aceitação contemplados, *User Stories* finalizadas, testes de unidades, interfaces construídas, entre tantas outras.

4.4.5.1.2.9 Teste de Usabilidade

Estes testes não são exatamente artefatos, pois sugerem uma prática. Porém, por serem planejados e executados, e como a aceitação do sistema depende dos resultados destes testes, ele é considerado um artefato importante.

Estes testes são a prática de entrevistas, questionários e observações de uma situação de uso do sistema, em uma simulação com usuários reais que pertencem ao perfil levantado no documento de visão do projeto. Estes usuários selecionados devem então seguir um roteiro planejado, e a eficácia nas utilizações do sistema é a métrica para o sucesso do teste. Esta eficácia é medida através de um questionário pós-teste, e este *feedback* é utilizado para a manutenção de eventuais problemas de usabilidade.

4.4.5.1.2.10 Matriz de Competências

Este artefato é uma tabela simples que mostra as competências de cada desenvolvedor do projeto. Estas competências devem ser relacionadas às tecnologias utilizadas no projeto.

Esta matriz serve também como métrica do aprendizado e evolução dos desenvolvedores, pois se realizada ao início e ao fim do projeto, pode ser utilizada para a comparação das competências de cada membro em dois tempos diferentes, constatando, então, quem aprendeu o quê.

A utilidade desta Matriz em um ambiente de produção pode parecer muito pequena, já que o YP foi desenvolvido para o uso em um ambiente acadêmico, e nesse caso, a avaliação do aprendizado dos envolvidos é algo fundamental. Porém, a Matriz representa, também, um dado interessante sobre os membros da equipe, podendo ajudar, inclusive, o gerente no momento em que as tarefas são alocadas para os desenvolvedores, já que, sabendo de antemão as habilidades de cada um, pode-se “driblar” as dificuldades de um ou outro, diminuindo os riscos de um desenvolvedor ter que aprender uma nova tecnologia.

4.4.5.1.2.11 Código Fonte

Este artefato é o objetivo de todo o processo: o código fonte do sistema. Ele deve ser sucinto e claro o suficiente para que sirva de comunicação entre os desenvolvedores. Como o YP incentiva a prática da Propriedade Coletiva de Código, todos os desenvolvedores terão acesso a qualquer parte dele, logo, é interessante que o código fonte seja de fácil entendimento.

4.4.5.1.2.12 Análise de Riscos

A Análise de Riscos, desenvolvida pelo Gerente, é um documento que mostra os riscos associados a determinados aspectos do projeto, como por exemplo: o aprendizado de uma nova tecnologia. Em um projeto já em andamento, a implantação do próprio processo YP pode representar um risco.

Este documento é normalmente uma planilha simples, com a descrição dos riscos, prioridade, responsável, possível solução ou providência para o mesmo, e um estado (ou *status*) do problema, que pode assumir valores como: “Resolvido”, “Superado”, “Vigente”, “Abortado”, entre outros.

4.4.5.1.3 Conclusões Sobre o Processo

Como visto nesta seção, o processo do YP é bem definido, com etapas e reuniões com tempo e escopo determinados. Viu-se também que todo o processo é acompanhado semanalmente através da reunião semanal, uma prática similar à

utilizada no *Scrum*, proporcionando uma grande flexibilidade e adaptabilidade ao processo, prevenindo com antecedência erros e diminuindo potenciais riscos.

Outro ponto interessante do YP é a quantidade de artefatos, que além de relativamente pequena, possui um bom número de artefatos gerenciais, o que auxilia e incentiva o controle do processo.

O processo é também relativamente simples de aprender. As etapas, em geral, não possuem artefatos complexos e priorizam a comunicação entre os membros da equipe e entre desenvolvedores e clientes. A única exceção a isto, do ponto de vista do autor deste trabalho, é o formalismo TAOS, que é relativamente complexo e um pouco mais difícil de aprender que o restante do processo. O YP não exige o uso do TAOS como única forma de modelagem de tarefas, e por isso seria interessante sua possível substituição por outro modo mais simples de representação.

4.4.5.2 **Papéis e Responsabilidades**

O YP possui um número pequeno de papéis, o que facilita na formação de equipes pequenas de desenvolvimento.

Cada papel possui responsabilidades diferentes associadas, e como visto anteriormente, um papel não é exclusivo de uma única pessoa, sendo que uma pessoa pode ter mais de um papel dentro da equipe.

Certos papéis possuem maior importância em determinadas etapas do que em outras. Essa distribuição de importância e responsabilidades de cada papel em etapas do processo pode ser vista na Figura 4.10.

Esta seção tem por objetivo explicar com mais detalhes as funções e responsabilidades de cada um dos papéis do YP, de acordo com o descrito em (PET/UFCG, 2007).

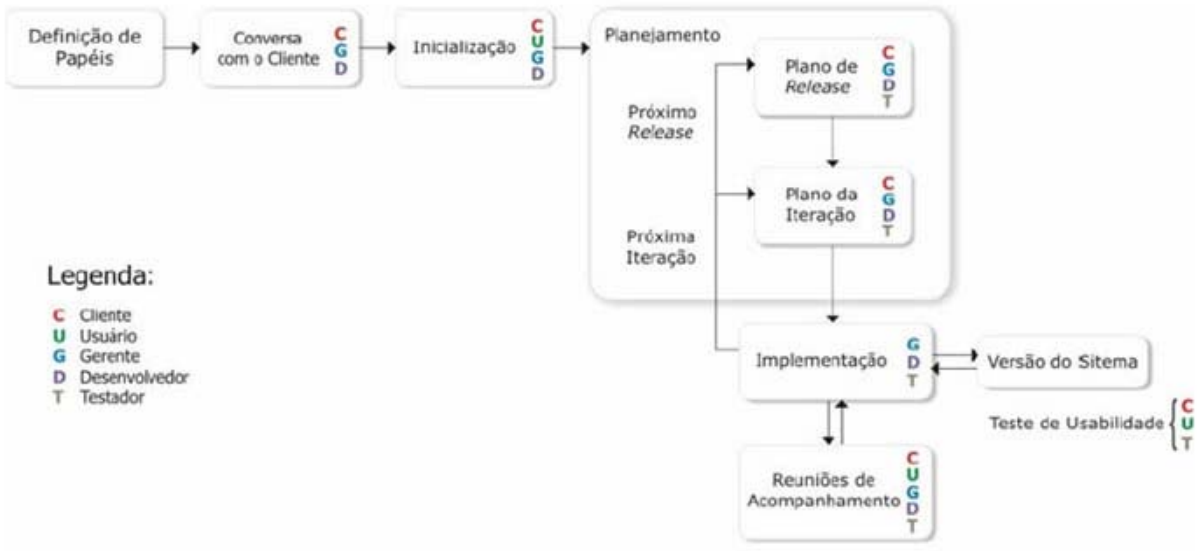


Figura 4.10: Representação dos Papéis no Processo YP
Autor: PET/UFCG em (PET/UFCG, 2007)

4.4.5.2.1 Cliente

O papel de Cliente é desempenhado por quem solicitou o desenvolvimento do sistema, ou seja, o cliente não necessariamente será aquele que efetivamente utilizará o produto. Porém, é o cliente que sabe como o *software* deve ser e o que deve desempenhar. Portanto, sua participação ativa no projeto é fundamental.

Suas principais responsabilidades são:

- Definir os requisitos do sistema: como já explicado acima, o cliente deve saber o que o seu *software* deverá fazer, e a transmissão destas informações para a equipe de desenvolvimento de forma correta é fundamental;
- Priorizar as funcionalidades: o cliente deve especificar quais funcionalidades do seu sistema são mais urgentes, para que sejam implementadas nos primeiros *releases*, facilitando a captação e resolução de erros nos setores mais críticos do sistema, e permitindo o planejamento adequado das tarefas. O cliente ajuda também, dessa forma, no planejamento dos *releases*;

- Elaboração dos testes de aceitação: os testes de aceitação dizem se um aspecto do sistema está aceitável ou não, cabendo ao cliente sua elaboração;
- Identificar o perfil do usuário e os objetivos de usabilidade: mesmo se o cliente não for o usuário final do sistema, ele deve saber que tipo de usuário o utilizará. Esse perfil ajuda na elaboração dos testes de usabilidade, indicando uma categoria de potenciais usuários do sistema, que devem ser então recrutados para as entrevistas e questionários. Junto com essa descrição sobre os usuários, o cliente deve também listar um conjunto de objetivos mensuráveis, na forma de requisitos de desempenho e usabilidade;
- Validação de documentos: o cliente valida documentos como o protótipo de interface e o projeto arquitetural. Só após o aval do cliente é que essas etapas do sistema podem continuar, ou ser declaradas como terminadas.

Como pôde ser visto, o Cliente desempenha um papel muito importante no desenvolvimento do sistema, pois toda a parte de planejamento e inicialização depende dele, bem como toda a parte de aceitação das tarefas já realizadas.

4.4.5.2.2 Usuário

É quem utilizará o sistema. É de acordo com as preferências do usuário que o sistema deve ser desenvolvido, por isso é tão importante que o cliente dê para os desenvolvedores o perfil de usuário correto, pois é a partir do usuário que a interação humano-*software* é idealizada e implementada.

O usuário possui as seguintes responsabilidades:

- Ajudar a definir os testes de aceitação e objetivos de usabilidade: juntamente com o cliente, o usuário pode ajudar na definição dos testes de aceitação/objetivos de usabilidade, de um ponto de vista mais prático;

- Validar o protótipo da interface e acompanhar continuamente a interface do sistema: o usuário valida o protótipo da interface do sistema, de um ponto de vista de quem efetivamente usará o sistema, tecendo críticas e sugestões de melhorias para a mesma, e, durante a implementação da interface, ele deve continuar em contato com a interface, para novamente poder sugerir mudanças na mesma.

Apesar de ligeiramente menos importante para o desenvolvimento do que o cliente, o usuário pode, em diversos momentos, dar sugestões aos desenvolvedores sobre como a interface pode melhorar, do ponto de vista de um potencial usuário do sistema. Isto ajuda muito, pois normalmente o desenvolvedor não enxerga o sistema do mesmo jeito que um usuário final, o que pode tornar o *software* difícil de utilizar e/ou entender.

4.4.5.2.3 Gerente

O gerente do projeto é o principal responsável por coordenar todo o processo, ou seja, todas as atividades de todos os envolvidos no mesmo. De posse da Análise de Riscos e utilizando as informações adquiridas nas Reuniões de Acompanhamento, o gerente é capaz de tomar decisões que podem mudar os rumos do projeto.

O gerente é o papel dentro do YP com mais responsabilidades, sendo as principais delas:

- Presidir todas as reuniões: o gerente deve conduzir os rumos de todas as reuniões previstas no processo do YP. Nas Reuniões de Acompanhamento, que são semanais, o gerente deve avaliar o andamento do projeto, com base nas métricas recolhidas, na análise de riscos, e no *Big Chart*;
- Conduzir os planejamentos e as ações dos desenvolvedores: é papel do gerente garantir que todos os desenvolvedores estão cumprindo seus prazos e metas, e que estejam trabalhando corretamente em suas tarefas. Isto é controlado pelas reuniões de acompanhamento, mas é no ambiente de trabalho que este controle sobre os

desenvolvedores deve ser mais rígido, para assegurar o bom andamento de todo o projeto;

- Elaborar o plano de desenvolvimento (*releases* e iterações): é o gerente que faz o planejamento de todas as atividades a serem executadas em uma iteração, e conseqüentemente, em todos os releases;
- Avaliação dos riscos envolvidos: a análise de riscos é fundamental para que se evitem surpresas no desenvolvimento. É papel do gerente estar constantemente atualizando a Análise de Riscos e avaliando cada risco descoberto, categorizando-os por prioridade, e procurando possíveis soluções para os mesmos;
- Gerência de Configurações: já visto em outras metodologias, esta atividade consiste em manter-se a consistência dos artefatos, para que um desenvolvedor não esteja utilizando um artefato diferente de outro desenvolvedor. Isto é feito, geralmente, utilizando-se ferramentas de controle de versão. Esta atividade é responsabilidade do gerente;
- Coleta e análise de métricas: As métricas são informações muito importantes para o controle do processo, pois são a matéria-prima dos artefatos de gerência do YP. É dever do Gerente a coleta e análise destas métricas, bem como sua utilização no *Big Chart*;
- Alocação de testadores: Como é o gerente que aloca as tarefas para cada membro da equipe de desenvolvimento, é também dever do gerente alocar quem irá testar a tarefa desenvolvida por este desenvolvedor. É importante que um desenvolvedor nunca teste sua própria atividade, pois uma pessoa estranha à atividade irá captar erros com mais facilidade que a pessoa que a programou;
- Resolver conflitos internos: Em uma equipe, bem como em qualquer grupo de pessoas com certo período de convivência, conflitos podem acontecer, e é dever do gerente minimizá-los e/ou resolvê-los,

mantendo o ambiente de desenvolvimento o mais agradável possível para todos os envolvidos;

- Tornar a documentação do projeto sempre acessível: Além de manter tudo sempre atualizado, é papel do gerente manter a informação o mais acessível possível, para que possa ser vista por todos os membros da equipe de desenvolvimento e, até mesmo, pelo cliente, quando possível.

Com todas estas responsabilidades, o Gerente é a peça chave de todo o processo, e o seu comprometimento com o projeto e o cumprimento do processo é fundamental para o sucesso de toda a empreitada.

4.4.5.2.4 Desenvolvedor

O papel de desenvolvedor consiste em modelar os requisitos do sistema, e posteriormente produzir um código eficiente e correto que corresponda a estes requisitos, fazendo uso das práticas e princípios do YP (que serão discutidos mais adiante).

As principais responsabilidades de um desenvolvedor são:

- Levantar requisitos funcionais e não funcionais, junto ao cliente, e elaborar projeto arquitetural: é o desenvolvedor que possui o fundamental papel de extrair do cliente todas as funcionalidades do sistema, e a partir delas ser capaz de montar um projeto arquitetural, escrever *User Stories*, entre outras informações importantes para o desenvolvimento do sistema;
- Auxiliar o gerente na elaboração dos planos de desenvolvimento: mesmo de posse da Matriz de Competências, o gerente ainda pode ter dúvidas quanto à capacidade de geração de código de sua equipe, podendo até mesmo atribuir prazos muito longos, ou muito curtos, para a realização de determinadas tarefas. Cabe ao desenvolvedor informar ao gerente suas estimativas para o cumprimento de suas

atividades, facilitando no processo de planejamento de todo o processo de desenvolvimento;

- Análise e modelagem da tarefa: o desenvolvedor deve modelar as tarefas a serem executadas pelo usuário, de acordo com algum formalismo para tal. Através desta modelagem é que se pode ter uma idéia da interface do sistema, bem como de uma possível arquitetura. E é também através desta modelagem que se pode fazer a divisão de uma tarefa em *User Stories*;
- Gerar o protótipo da interface, e identificar os objetivos de usabilidade: Com base na modelagem da tarefa, o desenvolvedor deve gerar um protótipo da interface do sistema. Este protótipo deve ser feito na forma de esboço, de preferência com lápis e papel, para que o usuário possa ter uma idéia geral da interface a ser desenvolvida. É também utilizando esta técnica, em conjunto com a opinião do cliente e do usuário, que o desenvolvedor deve ser capaz de identificar os principais objetivos de usabilidade. Após este esboço obter o aval do cliente e do usuário, a interface deve então ser implementada pelo desenvolvedor;
- Gerar testes de unidade: cada desenvolvedor deve gerar testes de unidade para o código por ele produzido. Isto provê um maior entendimento do código, e diminui a incidência de erros no desenvolvimento;
- Construir o Modelo Lógico de Dados: se o sistema possuir um banco de dados, ou outro tipo de persistência de dados, ou qualquer estrutura de dados relevante, esta deve ser modelada pelo Desenvolvedor, utilizando um modo de representação apropriado;
- Manter a integração contínua do código: em projetos com mais de um desenvolvedor, este deve ser responsável pela constante integração do código desenvolvido com o restante do código do sistema. Isto pode ser realizado com o uso de ferramentas de controle de versão, como o CVS (Capítulo 2).

O desenvolvedor representa no projeto a força-tarefa de todo o processo, e sua conduta correta, com o cumprimento de prazos e metas e utilização das práticas e princípios do YP, fazem com que a produtividade de todo o projeto aumente, diminuindo custos e gerando *software* de qualidade com eficiência.

4.4.5.2.5 Testador

O Testador é o responsável por revisar o código escrito pelos desenvolvedores, e realizar os diversos testes no sistema, como o teste de aceitação e testes de integração, usabilidade e funcionalidade.

Suas principais responsabilidades são:

- Efetuar revisões no código dos desenvolvedores, e refazer quando necessário: o testador, como já exposto, é responsável pela constante revisão e refatoração do código, quando necessário. Esta refatoração não deve alterar a funcionalidade do código, apenas melhorá-lo ou corrigi-lo;
- Efetuar testes sobre o código de outro desenvolvedor: apesar de aparentemente óbvio, cabe ao testador executar os testes planejados sobre o código de um desenvolvedor. Mas além de tudo isto, o testador deve ser capaz de identificar trechos de código ainda não testados, e de aprimorar os testes já existentes. Como já dito anteriormente, o testador de um código não deve ser o desenvolvedor do mesmo;
- Gerar os testes de aceitação: Antes da entrega de uma versão do sistema, os testadores devem implementar os testes de aceitação propostos pelo cliente. Cabe também aos testadores a execução destes testes;
- Elaborar o material para os testes de usabilidade: O testador deve preparar o material para o teste de usabilidade, que é composto de questionários, roteiros de utilização do sistema, além de recrutar usuários para os testes.

O testador é então o responsável pela validação do funcionamento do sistema, realizando testes que verificam se tudo está funcionando de acordo com as preferências do cliente. São também responsáveis pela otimização de código e correção de *bugs* no sistema.

4.4.5.3 **Práticas e Princípios**

Como a maioria das metodologias ágeis, as práticas e princípios da metodologia são a chave para o sucesso em sua aplicação. Com o YP não é diferente. Mesmo tendo um processo forte e controlado (principal influência do RUP), é nas práticas e princípios que se podem ver com bastante clareza as influências do XP e do *Agile Modeling*, E são exatamente estas práticas que “dão o tom” de uma metodologia ágil ao YP.

Nesta seção se discutirá os principais princípios e as práticas mais importantes da metodologia, destacando, principalmente, sua importância dentro da metodologia e no processo de desenvolvimento do YP. Ao final, se concluirá sobre os mesmos.

4.4.5.3.1 Número Reduzido de Artefatos

Este prática, vinda do XP, é levada muito a sério pelo YP. O número de artefatos do YP é bastante reduzido, para que se incentive a comunicação entre todos os envolvidos, e se evite a “comunicação via papel” que em projetos pequenos não faz sentido e só diminui a produtividade de todo o processo (Cockburn, 2002).

A maioria dos artefatos no YP serve apenas para a gerência do projeto, e não servem como meio de comunicação entre a equipe, como o faz, por exemplo, um Diagrama de Seqüência da UML no RUP.

Esta idéia faz com que o processo do YP flua com mais rapidez, dado que a criação dos artefatos é muito simples, sendo que a grande maioria deles é desenvolvida em conjunto nas fases iniciais do projeto e durante reuniões de acompanhamento.

4.4.5.3.2 Propriedade Coletiva de Código

Esta prática, vinda do XP e do *Agile Modeling* (e já explicada anteriormente quando foi discutido o XP), diz que o código é de propriedade de toda a equipe de desenvolvimento, ou seja, todos conhecem todo o código da aplicação e têm o direito e o dever de alterá-lo quando necessário.

Esta prática faz com que o código esteja em um processo de constante otimização, mantendo a aplicação com menos erros. Mas, para que isto aconteça, deve haver um eficiente controle de versões do sistema, para que não haja conflito entre versões do código.

4.4.5.3.3 Integração Contínua

O princípio da Integração Contínua do YP propõe que toda vez que uma atividade for programada pelo desenvolvedor, e não possuir erros de compilação, esta deve ser mandada para o repositório central do código, ou integrada ao restante do código, mantendo sempre o sistema integrado.

Esta idéia facilita muito o desenvolvimento em equipe, principalmente quando a equipe não possui horários de trabalho em comum, pois, se todos os desenvolvedores estiverem sempre de posse de uma versão atualizada de todo o código, a comunicação via código aumenta e a incidência de potenciais erros de integração diminui.

Esta prática facilita também o acompanhamento de todo o projeto pelo Gerente, visto que, como ele terá sempre a versão mais recente do código, terá a capacidade de captar métricas adequadas, realizando um controle mais preciso das atividades e prevendo possíveis riscos associados a atrasos na etapa de implementação.

4.4.5.3.4 Participação do Cliente e do Usuário nas reuniões

A prática de *“Client on Site”* (ou Cliente no Local), utilizada largamente no XP, propõe a presença constante dos clientes e potenciais usuários do sistema durante todo o processo de desenvolvimento, tirando dúvidas, dando sugestões aos desenvolvedores, entre outras possíveis participações.

No YP isto é feito de maneira mais moderada, através das reuniões de acompanhamento e planejamento, e através das validações de tarefas e testes.

A comunicação entre o cliente, usuários e desenvolvedores é incentivada por esta prática, mas a presença dos clientes e usuários não é necessária sempre. Isto facilita mais quando cliente e usuários não dispõem de muito tempo para acompanhar o processo.

4.4.5.3.5 Boas Práticas de Codificação

Estas práticas, usadas na etapa de implementação do sistema, são muito populares na grande maioria das metodologias ágeis. Elas pregam a simplicidade, a facilidade do entendimento e a refatoração constante do código. Diversas boas práticas de codificação podem ser usadas na atividade de codificação.

Esta seção aborda as principais práticas de codificação sugeridas pelos autores do YP.

4.4.5.3.5.1 *Design* Simples

Muito utilizada no XP, esta prática consiste em programar o código da maneira mais simples possível, da forma mais legível possível. O código tem de servir de meio de comunicação, sendo praticamente auto-explicativo. A idéia é programar somente o necessário, sem deixar “esperas” para funções futuras. Estas devem ser programadas quando necessário.

4.4.5.3.5.2 Padrões de Codificação

Esta idéia sugere que a equipe entre em um consenso sobre a forma de apresentação do código, para que todos vejam o código da mesma forma: com a mesma identação, colocação de chaves, nomes de funções com o mesmo formato, etc.

4.4.5.3.5.3 Padrões de Projeto

Um padrão de projeto é uma pequena arquitetura de classes que pode ser integrada ao sistema sem alterar sua estrutura já existente. Para vários tipos de problemas simples existem padrões já documentados que podem resolvê-los.

A utilização destes Padrões de Projeto pode agilizar na codificação, resolvendo os problemas mais comuns com rapidez e mantendo um padrão em todo o código, facilitando o seu entendimento.

4.4.5.3.5.4 Refatoração

A refatoração prega a reescrita do código sem alterar suas funcionalidades. Isto é feito para melhorar o código, para adequá-lo ao padrão utilizado pela equipe, para deixá-lo mais claro, entre outros aspectos não-funcionais.

Com esta prática, o código fica num processo de constante evolução, sempre melhorando, e ficando cada vez mais simples, legível e otimizado.

4.4.5.3.6 Testes

Os testes têm como finalidade a validação do sistema. Eles que conseguem provar que determinada funcionalidade, ou determinado aspecto do sistema, funcionam da maneira como deveriam.

Existem diversos tipos de testes, que testam diferentes aspectos do sistema: funcionalidade, unidade, integridade, usabilidade, carga, controle de acesso, desempenho, entre outros. Porém, o YP sugere que sejam utilizados três: Aceitação, Usabilidade e Unidade. Estes serão abordados com mais detalhes nesta seção.

4.4.5.3.6.1 Testes de Unidade

Estes testes garantem o funcionamento das menores partes de todo o sistema, classes, métodos ou funções, trechos de código e afins. Os testes de unidade são rodados exaustivamente durante o desenvolvimento, mantendo a integridade do sistema constante.

O YP sugere também o uso da técnica *Test Driven Development* (Desenvolvimento através de Testes), ou TDD, que propõe a escrita do teste antes da escrita do código que ele deve testar. Apesar de aparentemente abstrato, esta é uma boa prática de programação, pois facilita o entendimento do código, e faz com que o *software* esteja sempre funcionando sem erros, visto que o mesmo deve sempre passar nos testes.

Os desenvolvedores devem escrever seus próprios testes, porém estes devem ser revisados pelos testadores, para assegurar sua funcionalidade.

Esta prática, apesar de parecer simples, exige um grande esforço de programadores que não têm o costume de programar testes antes. Isto ocorre porque o TDD prega um paradigma muito diferente do clássico *"Code and Fix"*, visto que tudo deve ser montado a partir do teste, e nada pode ser programado que não seja causado (ou justificado) por um teste.

O YP sugere uma alternativa ao TDD, que é a programação paralela dos testes com o código. Esta prática não facilita tanto o entendimento do código quanto o TDD, mas ajuda também a manter a integridade do sistema.

4.4.5.3.6.2 Testes de Aceitação

Estes testes são definidos pelos desenvolvedores em conjunto com o cliente e o usuário, e estão sempre associados às *User Stories*. Eles devem ser definidos da forma mais simples possível, visto que o cliente deve especificá-los e entendê-los. Após a implementação de uma *User Story*, o testador então realizará estes testes.

Testes de aceitação são um conjunto de possíveis cenários e situações, que exigem um determinado comportamento do sistema que deve ser observado. Estes testes devem ter resultado positivo sempre que executados.

Esta prática facilita no processo de validação do sistema, e já que os testes vem de uma especificação simples, o cliente não tem dificuldade em dizer o que quer da funcionalidade e como ela deve funcionar, facilitando a comunicação entre desenvolvedor, cliente e usuário final.

4.4.5.3.6.3 Testes de Usabilidade

O teste de usabilidade serve para avaliar o desempenho de um usuário utilizando o sistema: se ele consegue realizar as tarefas, se não tem problemas com a interface do sistema (interação humano-computador), entre outros. Este teste procura também validar os objetivos de usabilidade e desempenho definidos no início do projeto com o cliente e usuários.

Este teste é realizado normalmente através de entrevistas, questionários e na prática do laboratório, que consiste em pôr potenciais usuários reais frente ao sistema, com um roteiro de atividades pré-determinado e avaliar seu desempenho, para depois captar seu *feedback* para que se efetuem possíveis melhorias no sistema.

Esta prática faz com que o sistema seja avaliado antes de sua versão final por um grupo de potenciais usuários, o que permite a reavaliação de alguns conceitos antes de entrar em produção. Isto faz com que o software, que eventualmente irá para as mãos do usuário final, tenha qualidade e seja fácil de usar e aprender, causando maior satisfação ao cliente e, principalmente, aos seus usuários.

4.4.5.3.7 Pequenos Releases

O YP propõe que o planejamento do software seja feito na forma de pequenos *releases*, com duas iterações de duas semanas cada. Ao final de cada iteração, um novo encontro é realizado com o cliente para o acompanhamento do sistema e validação das *User Stories* implementadas.

Esta prática, além de simplificar o planejamento, faz com que se diminua o impacto causado por mudanças nos requisitos, já que as *User Stories* são declaradas como finalizadas apenas após o aval do cliente. Deste modo, possíveis mudanças são captadas antes de ter-se um *release* mais longo, facilitando a adequação das *User Stories* implementadas aos novos requisitos, além de aumentar a comunicação e interação com o cliente durante o processo.

4.4.5.3.8 Conclusões sobre Práticas e Princípios do YP

O que se pode observar analisando-se as práticas e princípios do YP, é que todo o dinamismo da metodologia está embasado nelas. Estas práticas e princípios devem ser observados por todos os envolvidos, e seu uso correto aumenta as chances de sucesso no projeto.

Algumas práticas, entretanto, podem representar um pequeno risco ao desenvolvimento, visto que podem acarretar em aprendizado de novas técnicas e tecnologias por parte dos desenvolvedores menos experientes. Estes riscos devem ser avaliados no momento da implantação da metodologia.

A idéia de reuniões constantes com o cliente deve ser avaliada também, de acordo com a disponibilidade do cliente para o projeto. Ela deve ser incentivada, quando possível.

4.4.5.4 Primeiras Impressões sobre o YP

Como foi abordado, o YP, apesar de desenvolvido para uso na academia, possui um processo robusto e completo. É também uma metodologia leve e simples, com reduzido número de papéis e artefatos, e faz uso da maioria das práticas e princípios que consagraram diversas metodologias ágeis, como o XP.

O fato de ter sido planejado para academia que fez do YP um processo muito simples e prático. Como foi feito para o uso por equipes com nenhum conhecimento ou prática no uso de metodologias de desenvolvimento e processos, ele foi desenvolvido para ser simples, mas completo o suficiente para que o aluno tenha um contato com uma metodologia séria de desenvolvimento (Garcia, 2004).

O processo do YP possui um bom número de etapas, sendo estas curtas e bem definidas. Isto foi feito para que nada fique subentendido no processo, ficando muito claro para a equipe quando uma reunião deve ser feita, quando deve ser feito o planejamento das tarefas, quando o cliente deve interagir com a equipe, etc. Esta organização facilita o aprendizado.

O YP possui apenas cinco papéis. Esta simplificação do número de papéis vem do uso de uma metodologia simplificada para pequenas equipes. Porém, com esta prática, o desenvolvedor pode vir a funcionar como programador, arquiteto,

analista de requisitos e administrador. O baixo número de papéis pode ajudar em equipes pequenas, mas em projetos com equipes maiores e projetos mais complexos, isto pode prejudicar a organização, ou sobrecarregar membros da equipe. Entretanto, como o YP foi desenvolvido para o uso em projetos de tamanho pequeno/médio com equipes pequenas, isto não deve ser um problema.

O uso de práticas e princípios ágeis, utilizadas principalmente no XP e no *Agile Modeling*, traz o dinamismo e velocidade do mundo ágil ao YP. As práticas de Refatoração, Pequenos *Releases*, Integração Contínua, Propriedade Coletiva de Código, Testes, entre outras, são práticas, em sua maioria, simples, mas com ótimos resultados quando aplicadas corretamente. Porém, algumas são relativamente difíceis de aprender, como por exemplo: a prática dos testes antes, pois envolvem um possível estudo, por parte dos desenvolvedores, de tecnologias que desconhecem, ou que não usam, como o *JUnit*[10] ou o *HttpUnit*[11] por exemplo, bem como o uso de um paradigma de programação ao qual não estão acostumados.

Outras práticas, entretanto, incentivam a interação com o usuário e a comunicação entre membros da equipe. Estas costumam ser simples, facilitam o trabalho dos desenvolvedores, e devem ser aplicadas sem medo de possíveis riscos associados à sua implantação.

Com tudo isto, o YP parece inicialmente uma metodologia madura o suficiente para o uso em ambiente de produção, apesar de possuir apenas alguns anos de existência. Entretanto, como nunca foi usada em um ambiente de produção real, apenas na academia, sua implantação em um ambiente diferente deve ser estudada e possíveis adaptações podem vir a ser feitas para adequá-la.

4.5 CONCLUSÕES

Este capítulo mostrou um pouco sobre o que é uma metodologia para desenvolvimento de *software*, diferenciando-a de um processo e apresentou as duas categorias de metodologias hoje existentes. Com relação à escolha de uma

[10] O JUnit é um *framework* para criação e execução de testes unitários (ou testes de unidade) na linguagem Java. (*Wikipédia*, 2007a)

[11] O HttpUnit é um *framework* de código aberto para realizar testes em sites da *WEB* sem a necessidade de um *Browser* (ou navegador). (*Wikipédia*,2007f)

metodologia, vimos também, com bases nas análises em Abrahamson et al. (2002), diversos aspectos das metodologias RUP, XP, *Scrum*, *Crystal Clear*, *Crystal Orange* e YP.

Esta análise consistiu de descrições sobre o processo, papéis e práticas envolvidas, bem como primeiras impressões sobre as mesmas.

Os maiores problemas encontrados no uso de metodologias ágeis são a necessidade de experiência no uso de metodologias (para que se possa adaptar uma metodologia ao projeto), experiência em programação (para o uso de técnicas aprimoradas de desenvolvimento, como o TDD), e, no caso das metodologias *Crystal*, escassez de documentação.

Contudo, estas metodologias carregam, tanto em seus processos iterativos quanto em suas práticas, conceitos que agilizam o desenvolvimento, incentivam a comunicação, aceleram a resposta a potenciais problemas, simplificam o código, diminuem o número de artefatos necessários e ajustam a metodologia ao processo.

Uma metodologia que propõe um equilíbrio entre estes aspectos é a *EasyProcess* (ou YP). A YP é baseada em metodologias como o RUP e o XP, balanceando as vantagens das metodologias tradicionais com as ágeis, podendo ser classificada como uma metodologia híbrida.

No capítulo seguinte as metodologias analisadas neste capítulo serão comparadas. Com base nesta análise, é possível realizar a escolha da metodologia mais apropriada para o problema do LATIN, sendo ela tradicional, ágil ou híbrida.

5 A IMPLANTAÇÃO DE UMA METODOLOGIA

O processo de implantar uma metodologia em um ambiente de trabalho não é uma tarefa trivial e, portanto, deve ser pensado e analisado. Deve-se escolher uma metodologia apropriada e, depois de implantada, fazer com que as mudanças permaneçam após a aplicação.

Neste capítulo inicialmente se discute a escolha de uma metodologia para o ambiente do LATIN e são propostas adaptações a ela de acordo com o ambiente. Em seguida são discutidos aspectos da implantação, seguido do levantamento de possíveis erros e dificuldades que este processo pode acarretar. Finalmente, propõem-se métricas e meios para a avaliação do sucesso de uma implantação.

5.1 A ESCOLHA DA METODOLOGIA

De acordo com o que foi visto até este ponto sobre metodologias e processos de desenvolvimento, pode-se afirmar que não há uma metodologia que sirva para todo e qualquer projeto. A escolha de uma metodologia deve ser feita então com base nos projetos que a utilizarão.

O capítulo 2 fala sobre o Laboratório de Aplicação de Tecnologia da Informação, descrevendo seu ambiente, equipe e projetos em andamento. Com base nestas informações, é feito o estudo de caso para a escolha da metodologia mais adequada para a categoria de projetos lá desenvolvidos.

Os projetos do LATIN são, em sua totalidade, de médio a grande porte, de baixo risco e desenvolvidos por equipes pequenas de no máximo seis pessoas com pouca experiência em programação. Em uma primeira análise, pode-se dizer que a escolha mais óbvia seria escolher uma metodologia tradicional, com processo bem definido e grande cerimônia, para que tudo seja documentado. Porém, o LATIN não possui funcionários que possam definir com clareza o processo utilizado. Deste modo impossibilitando o uso de uma metodologia tradicional. Metodologias ágeis também não são as mais indicadas, pois, como já citado, o grupo de desenvolvedores não possui experiência. Para que se possa então escolher a

metodologia mais adequada ao LATIN, estes aspectos devem ser levados em consideração, e um meio termo deve ser encontrado.

Neste trabalho estudaram-se diversas metodologias ágeis: *XP, Scrum, Crystal Clear, Crystal Orange* (que não serve para o laboratório, pois necessita de uma equipe composta de vários times), uma tradicional: RUP, e uma híbrida: YP. As subseções seguintes mostram como foi realizado o processo de comparação e escolha da metodologia mais adequada para o problema.

5.1.1 COMPARANDO METODOLOGIAS

Em Abrahamson et al. (2002) os autores fizeram estudos sobre diversas metodologias e, baseados nas teorias de Song e Osterweil (1991) de comparação quasi-formal[12], compararam as metodologias umas com as outras com relação às categorias: processo, papéis e responsabilidades, práticas, adoção e experiências, escopo de uso e pesquisas.

Neste trabalho foi utilizada uma simplificação deste estudo, mas com a inclusão do YP para comparação com outras metodologias já existentes no estudo original.

Esta subseção apresenta algumas definições do estudo, comparação de cinco metodologias (RUP, XP, *Scrum*, Família *Crystal* e YP), e algumas conclusões obtidas.

5.1.1.1 **Definição do método de comparação**

A tarefa de comparar qualquer metodologia é muito difícil e o resultado muitas vezes é influenciado por experiências ou intuições do autor das comparações. Contudo, existe uma abordagem que supera essas limitações de uma comparação informal, chamada de comparação quasi-formal. (Abrahamson et al., 2002).

O autor Hank G. Sol (1983) propôs diversas abordagens para comparações quasi-formais, e a escolhida pelos autores do estudo foi: “Definir uma

[12] Uma comparação quase-formal se propõe a ser o meio termo entre uma comparação informal e uma formal. Deste modo, ela contém uma certa formalidade, mas não segue todos os princípios de uma comparação formal propriamente dita. Hank G. Sol (1983)

metalinguagem como veículo de comunicação e um quadro de referência contra o qual são descritos diversos métodos". Esta idéia foi aplicada ao descrever as metodologias que serão comparadas, pois já foram categorizadas em processo, papéis, práticas e conclusões quanto apresentadas nos capítulos anteriores.

5.1.1.2 **Comparação de Metodologias**

As comparações realizadas no estudo não buscam dar valor a atributos e/ou aspectos das metodologias estudadas. São apenas definidas diferenças e similaridades entre elas, para que se conclua qual a melhor para o uso a partir destes dados. Essas comparações foram divididas em etapas, contemplando cada uma determinados aspectos das metodologias.

5.1.1.2.1 Pesquisas Relacionadas

No campo de pesquisa as metodologias são categorizadas como "Nascendo", "Em crescimento" e "Ativa". Estas categorias denotam a quantidade de pesquisas desenvolvidas que abordam o uso e/ou adoção de determinada metodologia, bem como existência de comunidades de usuários e literatura relacionada. A tabela 5.1 relaciona os resultados obtidos.

Tabela 5.1: *Status* de metodologias quanto a pesquisas relacionadas

Status (em 2007)	Descrição	Metodologia
Nascendo	Metodologia criada e/ou disponível recentemente, pouca ou nenhuma pesquisa existente e pouca ou nenhuma experiência de uso existente	YP
Em crescimento	Metodologia reconhecida amplamente, com relatórios de uso e comunidade de usuários começando a aparecer, e pesquisas sobre o assunto identificáveis	Família *Crystal*
Ativa	Metodologia largamente utilizada, detalhada em vários estudos e bibliografias, comunidade de usuários e pesquisas relacionadas ativas	XP, RUP, *Scrum*

No estudo original (Abrahamson et al., 2002) a metodologia *Scrum* estava categorizada como "Em crescimento". Porém, deste então (o estudo data de Agosto de 2002) a metodologia evoluiu bastante, criando uma larga comunidade de

usuários e praticantes, bem como pesquisas e bibliografias relacionadas. Decidiu-se então passá-la para categoria “Ativa” neste trabalho.

O YP ainda não possui uso fora da academia, e por isso não há relatos de uso em ambientes de produção, bem como de uma comunidade de usuários. Porém, pesquisas são desenvolvidas em cima do uso da metodologia, com diversos trabalhos relacionados à metodologia sendo realizados atualmente.

5.1.1.2.2 Diferenciais e desvantagens

Como visto anteriormente, cada método de desenvolvimento ágil procura ter uma metodologia apenas suficiente para o problema de desenvolver *software*, e propõem soluções para este problema com abordagens diferentes. Sendo assim, cada método tem suas peculiaridades, diferenciando-o dos demais. Estas peculiaridades devem ser levadas em conta na escolha de uma metodologia, bem como os diferenciais e desvantagens individuais de cada uma.

A tabela 5.2 lista os métodos estudados e relaciona peculiaridades, diferenciais e desvantagens identificadas em cada método.

Tabela 5.2: Avaliação de peculiaridades, diferenciais e desvantagens de metodologias.

Método	Peculiaridades	Diferenciais	Desvantagens
XP	Desenvolvimento acompanhado pelo Cliente, Equipes pequenas, *builds* diários, TDD.	Refatoração – o constante re-design do sistema para aumento de desempenho e aceleração da resposta a mudanças.	Necessita de uma equipe experiente em técnicas de programação, e não dá atenção a parte de gerência do processo.
Crystal	Família de métodos. Todos possuem a mesma base de práticas e princípios, variando apenas os papéis, técnicas e ferramentas utilizadas.	A escolha da metodologia mais adequada é feita de acordo com o tamanho e risco do projeto.	Apenas duas das metodologias da família foram desenvolvidas, não havendo como analisá-las todas.
RUP	Modelo de desenvolvimento de *software* completo, incluindo conjunto de ferramentas. A definição dos papéis é	Modelo de negócios. Suporte a família de ferramentas da *Rational*.	O RUP não possui limitações no escopo de uso, sendo “adequado” a qualquer projeto. Um meio de como

	feita de acordo com a atividade a ser desenvolvida.		adaptar o método para um projeto específico não existe
Scrum	Times de desenvolvimento pequenos e com organização própria. Ciclo de *Release* com 30 dias.	Força um novo paradigma de visão do produto de acordo com a metodologia, invés do "Definido e repetível".	O *Scrum* detalha em específico como gerenciar o ciclo de um release, mas não detalha como deve ser feito os testes de aceitação e integração. O *Scrum* é fraco em técnicas e práticas de codificação, sendo normalmente complementado com práticas de outras metodologias.
YP	Processo robusto e completo. Práticas de codificação e gerência inspiradas em metodologias já consagradas.	Metodologia de fácil aprendizado. Gerência do projeto cobre todo o projeto. Interação com o cliente através da definição e homologação de testes de aceitação e usabilidade.	Metodologia desenvolvida para uso na academia, e, portanto, voltada para a mesma. Pode necessitar de adaptações para o uso em ambiente de produção.

Como pode ser visto, existem grandes diferenças entre os métodos estudados, cada um tendo peculiaridades bem diferentes uns dos outros.

O XP, por exemplo, é um método baseado em práticas (como o TDD, por exemplo), e por isso depende muito da perícia da equipe de desenvolvimento. Porém, o XP não possui boas práticas para controle e gerência do projeto, faltando um modo de se obter uma visão geral do processo.

Já com o *Scrum* é o contrário. O método possui diversas práticas para gerência, tendo um controle constante do "onde estamos agora" e do "para onde iremos deste ponto em diante", mas não possui práticas definidas para codificação. Como suas equipes são auto-organizadas e independentes, a escolha de práticas neste sentido é realizada localmente.

Entretanto ambos os métodos possuem uma grande interação com o cliente, tornando a resposta a mudanças (seja nos requisitos, em tecnologias, recursos, etc.) muito mais rápida.

A família *Crystal* é a única que incentiva a adaptação da metodologia ao problema, tendo, inclusive, em seu próprio processo, etapas para o ajuste da metodologia.

O RUP possui um processo completo e não é necessariamente uma metodologia leve, mas pode ser usado como uma, se adequadamente adaptado. Porém, ele se destaca por ter um conjunto de ferramentas comerciais que suportam todo seu processo, algo que as outras metodologias não possuem.

E, finalmente, o YP que, por ser baseado no RUP, possui um processo completo e robusto. Mas seu maior diferencial é a facilidade de aprendizado do processo, já que, por ser desenvolvido para o uso com iniciantes no mundo das metodologias e processos, usa um número reduzido de artefatos e possui etapas bem definidas e de curta duração. O processo é iterativo e o controle gerencial é feito semanalmente, fazendo com que se tenha uma visão geral e constante do projeto. Porém, o YP foi desenvolvido para uso na academia, o que pode acarretar em adaptações para o uso em um ambiente de produção.

5.1.1.2.3 Visão do desenvolvimento

O estudo original inicia com uma comparação entre visões no desenvolvimento, traçando um paralelo entre as visões metodológicas clássicas com o que ocorre na prática. Isto é então comparado com o que pregam os métodos ágeis. Isto foi feito para que se saiba qual abordagem está mais próxima do que realmente ocorre na prática.

Este paralelo é traçado em relação a diversos aspectos de um desenvolvimento. A tabela 5.3 mostra estas comparações.

Tabela 5.3: Comparação de visões de desenvolvimento de *software* e métodos ágeis

	Visão metodológica	O que ocorre na prática	Visão das metodologias ágeis
Atividades	Tarefas discretas	Projetos pessoais inter-relacionados	Projetos pessoais inter-relacionados
	Duração previsível	Completude imprevisível	Completude do próximo release previsível
	Repetível	Dependente do contexto	Muitas vezes depende do contexto
Desempenho do Processo	Confiável	Dependente de condições contextuais	Desempenho atada a um release pequeno, portanto confiável
	Interações específicas	Inerentemente interativo	Interações especificadas incentivando a natureza inerentemente interativa
	Pequenas tarefas em seqüência	Muitas tarefas são realizadas em paralelo	Tarefas simples normalmente não utilizadas por causa de sua natureza dependente de contexto
Esforço dos desenvolvedores	Dedicados a projetos de *software*	Comuns a todas as atividades (projeto, não-projeto, pessoal, rotinas)	Desenvolvedores estimam o esforço necessário
	Não diferenciado	Específico do indivíduo	Específico do indivíduo
	Totalmente disponível	Totalmente utilizado	Totalmente utilizado
Controle do trabalho	Regularidade	Oportunismo, improvisação e interrupção	Apenas controles mutuamente concordados[13]. Respeito pelo trabalho alheio
	Milestones, planejamento e controle gerencial	Preferência individual e negociação mútua	Preferência individual e negociação mútua

[13] Com exceção dos métodos *Crystal* que definem os controles que devem ser utilizados.

O que pode-ser concluído desta análise é que a visão metodológica tradicional de desenvolvimento de *software* difere muito do que é feito na prática, enquanto que as metodologias ágeis procuram se aproximar disto.

5.1.1.2.4 Abrangência do método

Os métodos ágeis procuram se aproximar do que realmente acontece na prática em desenvolvimento de *software*, entretanto, normalmente não contemplam todo o ciclo de vida do desenvolvimento.

Como vimos que cada método aborda o problema do desenvolvimento de um ponto de vista diferente, é de se esperar que algum ponto do processo não receba tanta atenção quanto os outros.

A figura 5.1 mostra com detalhes esta abrangência das metodologias estudadas.

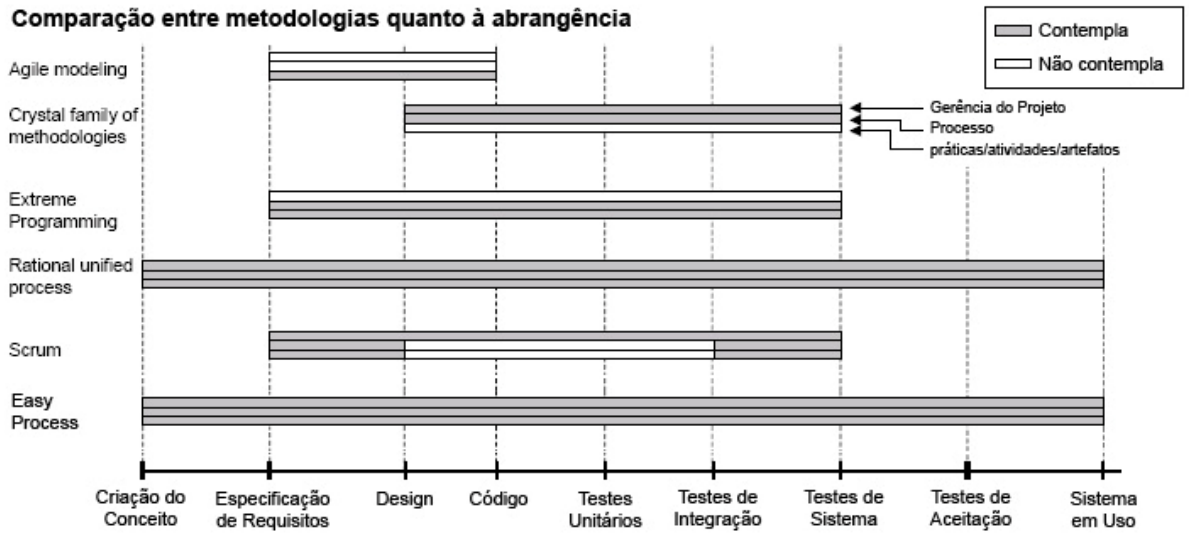


Figura 5.1: Comparação entre metodologias quanto à abrangência.
Autor: Adaptado de (Abrahamson et al., 2002)

Na figura pode-se notar a comparação de abrangência em três aspectos: Gerência do projeto, Processo e práticas/atividades/artefatos, sendo que um retângulo cinza quer dizer que a metodologia determina com detalhes métodos de gerência, etapas do processo, práticas, princípios, atividades e/ou artefatos utilizados para um ponto do ciclo de vida do desenvolvimento.

O método *Agile Modeling*, que não foi um método estudado neste trabalho, consta no gráfico por ser parte da base do YP. Pode-se notar que o mesmo possui apenas práticas para um pequeno trecho do desenvolvimento (indo da especificação de requisitos até o código apenas). E foi exatamente esta parte de modelagem do problema que foi utilizada no YP.

Nota-se também que o XP e o *Scrum* são métodos complementares, pois o que falta na abrangência de um, o outro cobre. E ambos possuem a mesma abrangência do ciclo de vida do desenvolvimento, que vai da especificação de requisitos até o teste do sistema.

Os únicos métodos que cobrem todo o ciclo de vida do desenvolvimento são o RUP e o YP. Isto é algo bastante óbvio, visto que o YP “herdou” boa parte da organização de seu processo do RUP, simplificando apenas no uso de artefatos e incentivando o uso de práticas de metodologias ágeis, como o *Agile Modeling* e o XP. Estes dois métodos possuem também cobertura completa na parte de gerência do projeto, mantendo controle absoluto sobre todo o processo de desenvolvimento.

5.1.1.2.5 Tamanho da equipe

Este é um aspecto que pesa muito na escolha de uma metodologia específica. Das metodologias estudadas, YP, XP e *Scrum* são desenvolvidas para times pequenos, na média de até dez desenvolvedores, enquanto que RUP e *Crystal* são escaláveis para até cem desenvolvedores. Claro que, quanto maior o time, maior as dificuldades de comunicação, sendo necessários mecanismos de comunicação e de controle gerencial mais apurados, deixando o processo mais pesado.

5.1.1.2.6 Facilidade no aprendizado

Este aspecto impacta bastante na implantação de uma metodologia, pois quanto mais fácil seu aprendizado e utilização, menos tempo os desenvolvedores ficam sem produzir (ou produzindo menos) para aprendê-la.

O estudo mostra que o RUP possui um processo completo, porém complexo, sendo provavelmente a metodologia estudada mais difícil de aprender. O

XP e o *Scrum* possuem técnicas que quebram o paradigma normal de desenvolvimento (o primeiro com a prática de testes antes, e o segundo com o modelo de negócios do produto). As metodologias da família *Crystal* são desenvolvidas para a implantação gradual, visto que devem ser adaptadas ao uso, e esses ajustes são feitos pela equipe. Porém, a equipe deve ter experiência no uso de metodologias, para saber o que deve ser ajustado e por que. E, finalmente, o YP, que apesar de possuir um processo completo como o RUP, é a metodologia mais fácil de aprender, pois é suficiente e simples, e desenvolvida para o uso por iniciantes.

5.1.1.3 **Conclusões levantadas nas comparações**

De posse agora dos estudos comparativos, pode-se visualizar com bastante clareza os diferenciais de cada metodologia, bem como as desvantagens de cada uma. É possível também levantar potenciais riscos associados à implantação e utilização destes métodos.

Uma metodologia com atividades e/ou práticas muito diferentes do que realmente é utilizado na prática de desenvolvimento pode não ser aceita pela equipe, que a utilizará em um primeiro momento, mas muito provavelmente a abandonará no futuro, ou não a utilizará corretamente.

Vimos que, neste caso, as metodologias ágeis se aproximam bastante do que acontece na prática, mas que nem sempre cobrem todo o desenvolvimento. Isto pode não ser um problema dependendo do tipo de projeto, já que um projeto com poucos riscos associados não precisa de uma fase de testes de sistema muito apurada, bem como um projeto pouco complexo não necessita de uma etapa de Criação de Conceito completa.

Outro ponto importante é a adaptabilidade da metodologia. Muito comum nas metodologias ágeis, este aspecto possibilita a resposta a mudanças no projeto ou nos recursos de forma dinâmica. Incentiva também o melhor ajuste da metodologia, para que seu uso seja o mais adequado possível.

As comparações mostraram que o *Scrum* e o XP realmente se complementam, o que explica o uso cada vez mais comum das duas metodologias

em conjunto. O XP possui o maior número de práticas e técnicas para o desenvolvimento, enquanto o *Scrum* possui maior ênfase na gerência do projeto.

A análise mostrou também que as metodologias da família *Crystal* são as mais adaptáveis, e que as metodologias RUP e YP possuem o processo mais completo e robusto.

A principal conclusão obtida nestas comparações foi a de que cada metodologia tem suas vantagens e desvantagens específicas, e que estas diferenças vêm do fato de que cada metodologia foi desenvolvida através de um ponto de vista diferente.

A análise destas vantagens e desvantagens deve ser levada em conta na escolha de uma metodologia, para que se escolha a mais adequada ao problema que se deseja resolver.

5.1.2 ESCOLHENDO UMA METODOLOGIA

Depois de realizadas as comparações entre as metodologias estudadas, pode-ser ter uma noção das peculiaridades de cada uma. Com bases nestes dados é possível escolher uma metodologia adequada às necessidades do LATIN.

Como foi visto no capítulo 2, o LATIN possui uma equipe com pouca experiência em programação e nenhuma experiência no uso de processos e metodologias.

Estas características dificultam muito o uso das metodologias XP, *Scrum* e RUP, pois são ligeiramente mais complexas e difíceis de aprender e seguir com eficiência.

A abrangência do processo também é algo importante nesta escolha, já que, de acordo com o que foi discutido em reuniões pela equipe, foi decidido que é fundamental o acompanhamento gerencial de todo o processo, para que a equipe saiba sempre em que ponto está, e para onde deve ir.

Com tudo isto analisado, optou-se por escolher a metodologia YP. Apesar de esta ser a mais recente, e de não ter sido desenvolvida para o uso em ambiente de produção, ela é, de longe, a mais fácil de aprender e uma das mais completas e

robustas do estudo. Além disto, ela abrange todo o processo, possui um excelente mecanismo de gerência, e seus artefatos, mesmo sendo poucos, podem servir de documentação para o projeto. Tudo isto é fruto do fato de ela ser uma metodologia baseada em outras metodologias tradicionais e ágeis, sendo assim híbrida, criando um meio-termo muito conveniente para este trabalho.

5.2 O QUE DEVE SER MUDADO INICIALMENTE NO YP

O autor Alistair Cockburn (2002) propõe que toda metodologia deve ser adaptada para o uso de acordo com o projeto em que deve ser aplicada. Deste modo, partindo de uma metodologia "base" deve-se chegar a uma metodologia adaptada que é utilizada no projeto. Ele propõe também que estas adaptações continuem sendo realizadas durante o uso da metodologia, no decorrer do projeto.

No caso do YP, vimos que a metodologia necessita de alterações antes do início do uso, principalmente pelo fato de não ter sido criada para um ambiente de produção real. O problema é que os projetos desenvolvidos pelo LATIN são comerciais e/ou institucionais. Analisando as categorias de projetos acadêmica e comercial/institucional, algumas diferenças importantes foram constatadas:

- Complexidade e Tamanho do projeto: um projeto acadêmico é normalmente muito mais simples e menor que um projeto real, contendo reduzido número de funcionalidades (já que muitas vezes a funcionalidade em si não importa para o projeto, e sim a tecnologia utilizada, teorias envolvidas, etc.);
- Escopo e Prazos: a execução de um projeto acadêmico é algo com começo, meio e fim bem definidos. Já um projeto comercial não tem fim definido, podendo o mesmo continuar indefinidamente em desenvolvimento, através de manutenções, correções de *bugs* e melhorias. Inclusive, podem-se criar novos módulos para um projeto, criando subprojetos agregados ao projeto principal. Um projeto acadêmico possui também um prazo específico para entrega, já um projeto contínuo, como os desenvolvidos e mantidos pelo LATIN, não possuem prazos finais, apenas prazos para *releases* e implantações.

- Definição de funcionalidades: projetos acadêmicos possuem desde o início seus requisitos bem definidos, enquanto que um projeto comercial sofre mudanças nos requisitos com freqüência.

No capítulo seguinte são propostas as alterações iniciais na metodologia, e elas levam em consideração o que foi levantado nesta seção. Estas alterações são ligeiramente simples, e não devem desvirtuar a metodologia do seu propósito original.

5.3 IMPLANTANDO UMA METODOLOGIA

O autor Kent Beck (1999b) propôs, para o XP, que a metodologia seja implantada gradativamente, facilitando sua compreensão e reduzindo os impactos que um novo método de trabalho possa acarretar na equipe.

Isto deve ser feito também com o YP, já que possui algumas das práticas mais inovadoras do XP, como a refatoração e o desenvolvimento dirigido a testes.

Esta implantação deve ser planejada antes de executada. Deve também se acompanhada constantemente, para que se descubram potenciais riscos e problemas com rapidez.

O autor Alistair Cockburn (2002) sugere a realização de *Workshops* de Reflexão com a equipe de desenvolvimento durante o uso de uma metodologia. Nesses *workshops* são discutidos os prós e contras da metodologia em uso, bem como são relacionados possíveis desconfortos no seu uso ou sugeridas possíveis modificações em sua estrutura e práticas. Esta prática dá ao processo de implantação um maior dinamismo. Dá também uma maior adaptabilidade à metodologia, que será moldada de acordo com o projeto e à equipe que o desenvolve.

5.4 PREVENDO ERROS E DIFICULDADES

Mudar todo um modo de trabalho em uma equipe de desenvolvimento que já está habituada a trabalhar de outro modo não é algo fácil, e dificuldades e

erros ao seguir este novo modo com certeza surgirão. Deve-se então prever estes erros e dificuldades e solucioná-los antes de acontecerem, ou com rapidez o suficiente para que suas conseqüências não sejam sentidas.

A etapa de implantação que utiliza um projeto piloto ou um projeto de teste é uma arma para a prevenção de erros e previsão de dificuldades na implantação da metodologia. De posse dos dados coletados neste pequeno projeto é possível ajustar a metodologia para que se diminuam as dificuldades para os desenvolvedores.

Porém, ainda assim podem surgir erros no decorrer da implantação final. Neste caso, é fundamental a comunicação com a equipe durante o processo e, principalmente, durante as reuniões de acompanhamento e *workshops* de reflexão, pois se estas dificuldades forem relatadas rapidamente podem também ser propostas soluções para elas com rapidez.

O artefato mais precioso para previsão de dificuldades e prevenção de erros dentro da metodologia é a Análise de Riscos. Portanto, sua manutenção e constante atualização são fundamentais na implantação da metodologia.

5.5 COMO AVALIAR O SUCESSO DE UMA IMPLANTAÇÃO

Esta seção levanta um dos maiores problemas de implantar uma metodologia com sucesso: como saber quando se tem sucesso. Para isso são discutidos tópicos como: como medir o sucesso de uma implantação, que métricas devem ser utilizadas, como obtê-las, e como saber se são precisas e confiáveis o suficiente.

Um processo de desenvolvimento, como foi visto anteriormente, não é algo regular, pois é baseado em pessoas, e pessoas são inconstantes. Uma metodologia tenta regrar e disciplinar um processo definido. Portanto, uma metodologia estará implantada com sucesso se conseguir disciplinar e regrar o desenvolvimento do modo como ela se propõe a fazer. No entanto, isto não é de forma alguma algo facilmente mensurável. Não é algo simples poder dizer “A Metodologia está 76,4% implantada com sucesso!”. E não precisa ser.

Propõe-se para este trabalho que a medição do sucesso da implantação da metodologia YP seja a constatação do seguimento correto dos passos da metodologia e do conforto dos envolvidos ao utilizá-la. Isto é uma atividade que deve ocorrer nos *workshops* de reflexão. Quando for constatado que a equipe está seguindo os passos da metodologia sem dificuldades, e que já está acostumada às atividades ao ponto de realizar etapas e práticas com clareza, pode-se dizer que a implantação foi um sucesso.

6 A METODOLOGIA APLICADA AO LATIN

Este capítulo se propõe a relatar os acontecimentos, dificuldades e constatações que ocorreram durante o processo de implantação da metodologia YP no Laboratório de Aplicação de Tecnologia da Informação. Inicialmente são mostrados os planejamentos para esta implantação e, em seguida, são relatados os acontecimentos que ocorreram durante a implantação gradual nos projetos do laboratório. Finalmente, se propõem outras mudanças na metodologia e são analisadas as dificuldades e problemas encontrados neste processo.

6.1 PLANEJAMENTO DA IMPLANTAÇÃO

Uma metodologia de desenvolvimento não pode ser implantada em um ambiente de trabalho sem qualquer tipo de planejamento prévio. No caso do YP no LATIN o proposto foi que esta implantação ocorresse em etapas simples e bem-definidas. São estas etapas:

- Apresentação da metodologia à equipe de desenvolvimento: os desenvolvedores conhecem a metodologia por completo, com ênfase no processo e nas práticas. Cada um fica a par de seus papéis e suas responsabilidades. Nesta etapa todas as dúvidas quanto à metodologia devem ser sanadas.
- Propostas de adaptação: Após a apresentação da metodologia, a equipe propõe mudanças e adaptações à mesma. Estas mudanças são avaliadas e, se aceitas, assimiladas.
- Projeto piloto: Um projeto simples e pequeno, de no máximo duas semanas, é desenvolvido utilizando a metodologia. Após o final deste projeto, é analisado o impacto no desenvolvimento e na gerência das atividades. Nova reunião de propostas de adaptação é realizada.
- Implantação gradual: Neste ponto é implantada a metodologia em seu formato final, com adaptações, em todos os projetos do laboratório. Mas esta implantação é gradual, deixando para adaptar novas técnicas de programação, ou novas atividades mais complexas no

processo quando a equipe já estiver acostumada com as práticas e atividades mais simples. O único aspecto que não deve ser gradual é o processo. Este deve ser implantado desde o início, pois ele representa a base da metodologia.

- *Workshops* de Reflexão: descritos em um capitulo anterior, estes *workshops* permitem um acompanhamento de como os desenvolvedores se sentem com o uso da metodologia, e possibilitam ajustes na metodologia.

A implantação gradual de aspectos da metodologia só deve ser declarada finalizada quando constatado este fato em um *Workshop* de Reflexão por toda a equipe. Após este passo, devem ser decididos quais novos aspectos da metodologia devem ser implantados. Este processo se repete até que toda a metodologia esteja implantada. A figura 6.1 ilustra o processo.

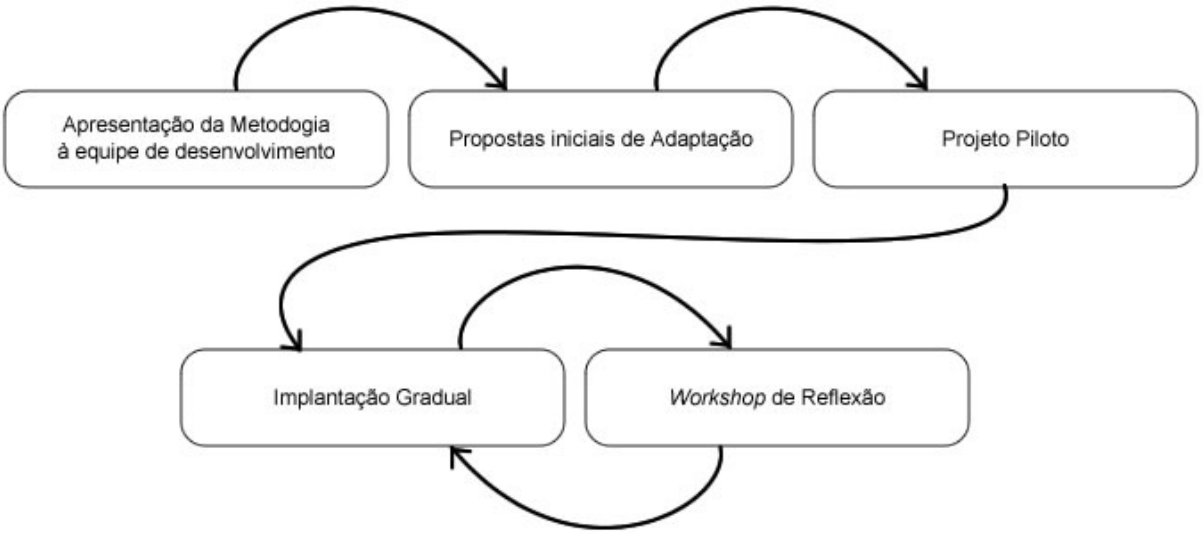


Figura 6.1: Etapas da implantação do YP
Fonte: Autor

Se corretamente seguidas estas etapas até o final, e se todos os problemas encontrados forem solucionados, a implantação da metodologia estará terminada.

6.2 COMO OCORREU A IMPLANTAÇÃO

Esta seção relata os acontecimentos ocorridos após a escolha da metodologia. São discutidos aqui aspectos da implantação do YP no LATIN como adaptações propostas e soluções tomadas para os impasses encontrados.

A parte prática deste trabalho começa neste ponto, com a implantação da metodologia em projetos reais. Esta implantação ocorre em etapas, e a seguir são relatados os acontecimentos ocorridos em cada uma delas.

6.2.1 APRESENTAÇÃO DA METODOLOGIA À EQUIPE

Nesta etapa foi apresentada a metodologia a equipe, como descrito anteriormente. A equipe assimilou bem o conhecimento, e as únicas dúvidas que surgiram foram em relação ao formalismo TAOS, como esperado.

6.2.2 AJUSTES INICIAIS AO YP

Esta etapa ocorre na mesma reunião da etapa anterior. Isto é realizado desta maneira para que o conhecimento adquirido na etapa anterior não seja esquecido no intervalo entre estas etapas.

Aqui são sugeridas as alterações básicas e iniciais na metodologia. São também definidas quais práticas e/ou atividades serão implantadas primeiro, e quais são deixadas para iterações futuras na implantação gradual proposta.

As primeiras alterações sugeridas pela equipe na metodologia YP foram relativas ao processo, visto que, por exemplo, o sugerido de dois *releases* com duas iterações cada não se aplica a um projeto contínuo. São estas adaptações:

- Aumento do número de *releases*: esta adaptação é relativamente simples: a idéia é não se limitar a dois *releases*, mas manter o número de iterações para um *release* em dois. Deste modo, a cada duas iterações deve-se ter um *release* funcional pronto para implantação;
- *User Stories* de Manutenção: Quando relatado um *bug*, ou funcionalidade que deve ser mudada, isto deve virar uma *User Story*

de Manutenção, e deve ser adicionada à iteração atual, ou às seguintes, de acordo com prioridade estabelecida pelo cliente;

- Inicialização dentro da iteração: esta alteração propõe que uma etapa similar a etapa de inicialização, mas simplificada, deve existir dentro da iteração, para que se montem protótipos de interface e modelos de tarefa para *User Stories* definidas durante o processo. Esta inicialização deve ocorrer junto com a reunião de acompanhamento, e deve abranger apenas as novas *User Stories* que serão implementadas na iteração que segue à reunião;
- Publicação para validação do usuário: Por não possuirmos contato direto e constante com os clientes/usuários do sistema, este será publicado em um servidor de testes (no caso de aplicações *WEB* somente) e então validado pelos usuários. Isto implica na montagem de um *release* antes da validação de testes de aceitação pelos clientes. Estes testes devem então ser analisados pelos testadores do laboratório antes do *release*. Se os clientes/usuários discordarem de alguma funcionalidade ou encontrarem *bugs* no sistema, estes problemas devem virar *User Stories* de Manutenção, e alterados no próximo *release*;
- *Release* de produção: Por utilizar a prática de publicar os *releases* em servidor de testes, a maioria destes *releases* pode não estar de acordo com o que querem os clientes. Porém, se isto não acontecer, estes *releases* de teste podem ser utilizados no servidor de produção, virando assim *releases* de produção. O sugerido para contemplar esta característica do desenvolvimento é que se utilizem duas categorias de *releases*: RC (*Release Candidate*, ou Candidato a Release) e FR (*Final Release*, ou Release Final/de Produção). A idéia é que um FR sempre venha da aceitação e validação completa de um RC;
- Subprojetos ou módulos: Foi decidido que um subprojeto ou módulo de um sistema deve ser tratado como um novo projeto, possuindo todas as etapas de um projeto comum;

- Extinção do artefato Matriz de Competências: Por possuir uma equipe pequena, onde todos conhecem as habilidades de todos e a comunicação entre desenvolvedores é algo comum, não vimos necessidade em utilizar a Matriz de Competências, pois seria um artefato que, por ser pouco utilizado, não possuiria grande valor para o projeto;
- Uso opcional do TAOS: Por ser um formalismo um pouco complicado de utilizar e aprender, a equipe decidiu que o uso de um formalismo para modelar a tarefa é algo opcional, e deve ser utilizado apenas para tarefas mais complexas. Isto ocorreu devido ao fato de que a maioria das telas e tarefas do sistema SINGU (principal projeto do laboratório) possui um padrão de funcionamento. Ou seja, a maioria das telas funciona de modo similar, e uma descrição menos precisa é mais do que suficiente para que o desenvolvedor saiba o que deve ser feito.

Nenhuma mudança nos papéis e responsabilidades foi sugerida.

Quanto à prática de testes unitários, ficou definida a utilização de testes em paralelo. Já os testes de aceitação não serão implementados, sendo apenas descrições de cenários de utilização que devem ser reproduzidos pelos testadores manualmente.

Nesta reunião foi decidido também que o processo deve ser implantado com as alterações propostas integralmente. Quanto às técnicas e práticas, foi sugerido que o uso de formalismos de modelagem de tarefas, testes de unidade e testes de usabilidade fiquem para iterações futuras da implantação gradual.

6.2.3 PROJETO PILOTO

Após a escolha da metodologia, e a definição de suas primeiras adaptações básicas, surgiu um desejo de pôr o YP à prova, para verificar sua eficácia em um projeto real. Foi decidido, então, que o YP seria utilizado como metodologia em um projeto simples e muito pequeno, de curta duração. Isto foi feito

para que se possa analisar a metodologia em uma atividade real antes de implantá-la definitivamente, possibilitando o levantamento de conclusões sobre o uso da mesma e um melhor julgamento sobre as alterações iniciais realizadas.

O projeto escolhido é o desenvolvimento de um subsistema simples, com poucas classes, chamado de *ServletLattes*. Este sistema serve de interface entre o sistema SINGU, em desenvolvimento pelo LATIN, e o sistema de currículos Lattes, do Instituto STELA. A idéia é que quando o ServletLattes receber uma requisição para captura de um currículo Lattes, ele se conecte com o sistema Lattes e capture o currículo desejado, retornando estas informações para o requisitante. Sendo assim, o *ServletLattes* serve de ponte de conexão entre dois grandes sistemas. O desenvolvimento deste subsistema já era algo necessário, e decidiu-se então aproveitar o fato para testar também o YP.

Após esta etapa nova reunião é realizada, para que se debata o YP novamente, mas agora com alguma experiência de uso. Novas adaptações podem ser sugeridas e assimiladas à metodologia.

Todas as etapas do Processo YP foram respeitadas durante o desenvolvimento deste pequeno projeto, e todos os artefatos que se faziam necessários ao projeto foram criados. Este projeto teve a duração de duas semanas, e foi composto de um único *release* com duas iterações de uma semana cada.

O relatório final do desenvolvimento deste projeto, que contém todos os artefatos, pode ser conferido no Apêndice A.

Na reunião que seguiu ao desenvolvimento do pequeno projeto foi analisado o uso da metodologia adaptada, e o constatado foi que uma maior organização do desenvolvimento foi alcançada graças às técnicas de gerenciamento incluídas no YP.

Nenhuma alteração adicional ao YP foi proposta depois do uso no projeto piloto. Isto provavelmente ocorreu por que o projeto era muito simples, e o desenvolvimento muito curto, de apenas duas semanas, e problemas e dúvidas que podem ocorrer durante o uso em longo prazo não foram detectados. Ou seja, novas alterações podem vir a acontecer durante a implantação em projetos maiores.

6.2.4 IMPLANTAÇÃO GRADUAL NOS DEMAIS PROJETOS

Nesta etapa a metodologia é aplicada em todos os projetos do laboratório.

Esta implantação gradual se deu de duas maneiras diferentes: em projetos já em andamento, e em novos projetos (ou novos subprojetos). Isto ocorreu por que uma metodologia é desenvolvida para que seja utilizada em um projeto desde seu início. E implantar uma metodologia em um projeto em andamento é algo relativamente mais complicado que em novos projetos.

A seguir, são descritos os passos realizados para cada tipo de projeto:

- Novos Projetos: Em novos projetos, como o projeto piloto, se segue o processo desde o início, sem nenhuma alteração adicional às já propostas;
- Projetos em Andamento: Já em projetos em andamento, as etapas iniciais são simuladas, e se criam os principais artefatos da etapa de inicialização. São também descritos as principais *User Stories* que ainda não foram implementadas, com protótipos de interface, modelos lógicos, e demais artefatos necessários. Após essa "inicialização simulada" passa-se à etapa de planejamento, e inicia-se o processo iterativo, proposto na metodologia, normalmente.

Como foi planejada anteriormente, a implantação gradual se dá em iterações, e uma vez implantada uma prática que foi deixada para depois, ela deve ser utilizada em todos os projetos do laboratório. Mas no início esta prática pode ser aplicada em apenas um projeto para que se teste sua utilização real. Com isto, pode-se avaliar o conforto no uso de tal prática com mais facilidade, antes de efetivamente utilizá-la em todos os projetos.

Estes "níveis de conforto" devem ser medidos nos *workshops* de reflexão.

6.2.5 *WORKSHOPS* DE REFLEXÃO

Esta etapa propõe uma discussão entre a equipe quanto ao uso da metodologia. Nesta etapa adaptações da metodologia são propostas durante o uso

da mesma em projetos reais. Deste modo, o YP no LATIN deve evoluir constantemente, até que seu uso fique “estável”.

Durante a implantação da metodologia diversas alterações foram propostas nestas reuniões, bem como diversas dificuldades foram solucionadas. Um conhecimento mais apurado do processo e da metodologia também foi adquirido nestes encontros.

Pôde-se concluir com estas reuniões que a equipe estava realmente empenhada e interessada com o uso de uma metodologia de desenvolvimento, e que a organização alcançada com o YP realmente aumenta o entendimento do sistema e das tarefas que devem ser realizadas.

As alterações propostas pela equipe durante o uso do YP são apresentadas e discutidas mais a fundo na seção seguinte.

6.3 REVISANDO O YP: ADAPTAÇÃO DURANTE O USO

Após as primeiras iterações utilizando a metodologia YP nos projetos do LATIN, a equipe já formula suas primeiras impressões sobre o seu uso. Estas impressões são a base para a sugestão das primeiras adaptações durante a utilização.

Todas as adaptações propostas pela equipe foram discutidas e ponderadas nos *workshops* de reflexão realizados, e as decisões de assimilar ou não tais sugestões sugiram de um consenso entre os envolvidos.

6.3.1 UM FORMALISMO PARA MODELAGEM DE TAREFAS

Um dos principais prováveis problemas, previsto logo no início da implantação (quando do estudo da metodologia), foi rapidamente comprovado já nas primeiras iterações da metodologia: o uso do TAOS como formalismo para modelagem de tarefas.

Inicialmente se pensou em utilizar o TAOS, ou qualquer outro formalismo, apenas quando necessário. Porém, logo se concluiu que a falta da modelagem de

uma tarefa prejudica muito o entendimento da mesma. Decidiu-se então pela realização mais freqüente de modelos de tarefas antes do desenvolvimento.

O TAOS foi desenvolvido para a “aquisição e representação de conhecimento em um domínio centrado na análise de tarefas e ações que serão executadas neste domínio” (Medeiros, 2000) e foi criado para o uso em sistemas baseados em conhecimento. Os autores do YP sugerem seu uso (PET/UFCG, 2007), mas não o obrigam.

A equipe do LATIN começou a utilizá-lo na segunda iteração para modelagem de tarefas, e diversos problemas e dúvidas surgiram.

Os principais problemas no uso deste formalismo, constatados pela equipe, foram:

- Dificuldade no entendimento: um diagrama TAOS possui uma visualização fácil de entender quando a tarefa é relativamente simples, com poucos estados da interface[14], e poucas ações. Porém, quando a complexidade da tarefa aumenta, o diagrama do TAOS cresce horizontalmente, e, por ser em forma de árvore, pode ficar muito caótico. A figura 6.2 exemplifica como uma tarefa não muito complexa que utiliza o TAOS como modelo de tarefas pode ficar ilegível. A solução comumente adotada para estes casos é dividir o diagrama em diversas sub-tarefas, entretanto isto toma um tempo muito maior para desenvolver, demandando um grande esforço dos desenvolvedores.

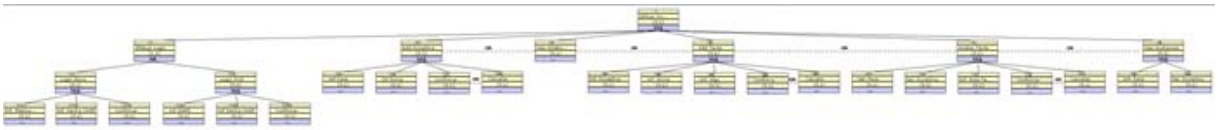


Figura 6.2: O uso do TAOS como representação de uma tarefa complexa.

- Dificuldade no aprendizado: o TAOS é um formalismo complexo, e relativamente difícil de aprender. A equipe tinha uma enorme dificuldade de entendê-lo, e modelar tarefas utilizando-o era uma

[14] Uma interface nos projetos *WEB* pode ter vários estados. Estas interfaces podem possuir botões que realizam o envio da informação. A cada envio de informação a interface pode mudar, causando diversos estados em uma interface.

tarefa complexa e demorada, até mesmo para as tarefas mais simples. Com isto, notou-se que o uso deste formalismo logo seria abandonado pela equipe.

- Ferramentas para criação/edição dos diagramas: a ferramenta indicada para a editoração dos modelos TAOS é o iTAOS[15] (fig. 6.3). Trata-se de uma ferramenta desenvolvida por alunos da UFCG, berço do YP. Ela procura facilitar o desenvolvimento de diagramas TAOS. Porém, a ferramenta possui diversos *bugs*, chegando às vezes a perder funcionalidades em pleno uso. A visualização do modelo também é prejudicada, e a exportação do diagrama em forma de imagem é muito pobre. Foi realizada uma pesquisa por outros editores de diagramas TAOS, mas nenhum outro foi encontrado.

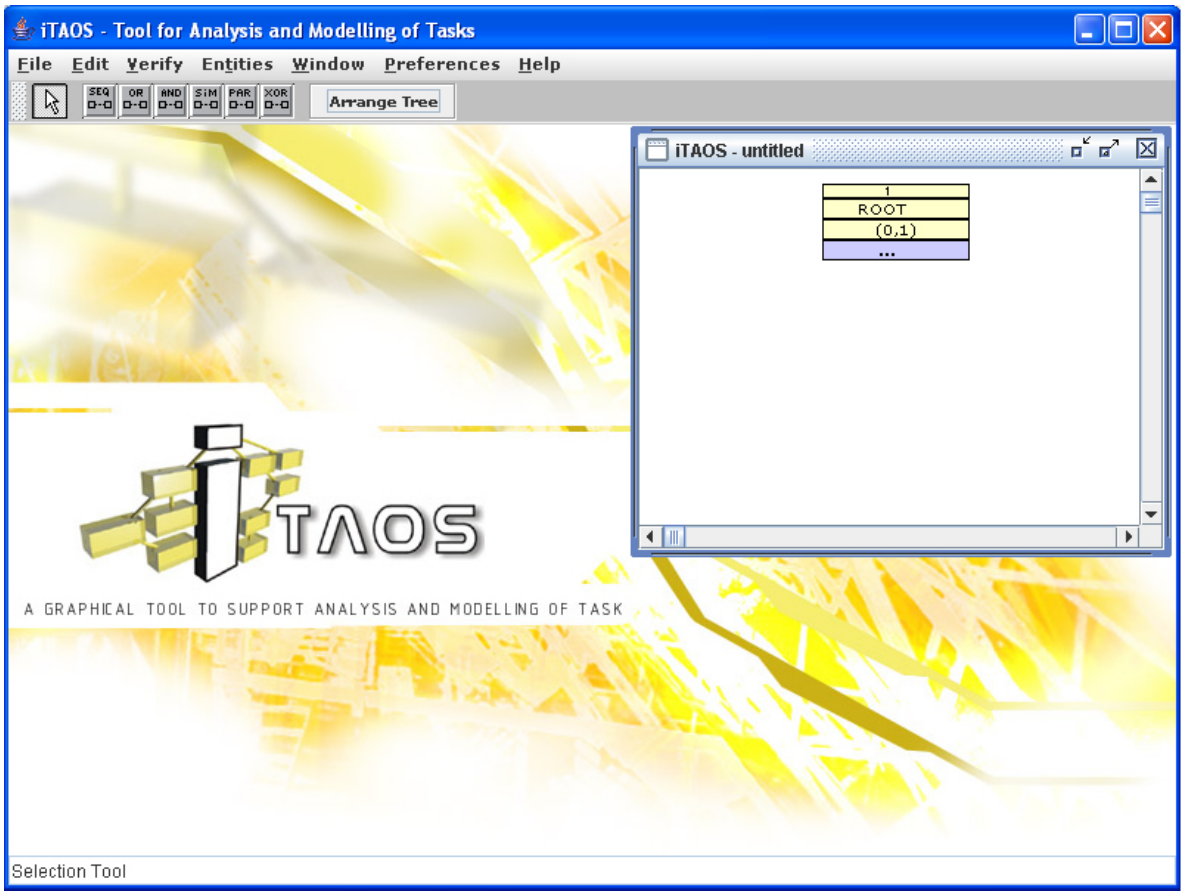


Figura 6.3: iTAOS

15 http://www.dsc.ufcg.edu.br/~itaos/

A solução proposta pela equipe para este problema foi a utilização de outro formalismo para modelar tarefas. E foi na UML que foi encontrado um substituto apropriado: o Diagrama de Atividades.

O Diagrama de Atividades é “utilizado para descrever a seqüência de atividades, utilizando comportamento condicional e paralelo” (Carlos, 2005) e é muito similar a um diagrama de estados comum, e foi escolhido como formalismo para modelagem de tarefas por possuir uma boa visualização (que possibilita um melhor entendimento da tarefa à primeira vista até por pessoas que não o conhecem), por ser muito mais fácil de aprender e utilizar, e por possuir uma enorme gama de ferramentas de editoração, já que a UML é uma linguagem para modelagem reconhecida mundialmente e utilizada por muitos desenvolvedores.

A figura 6.4 mostra uma comparação dos diagramas para o projeto piloto. Pode-se notar que o modelo utilizando diagrama de atividades é muito mais compreensível à primeira vista que o modelo utilizando o TAOS.

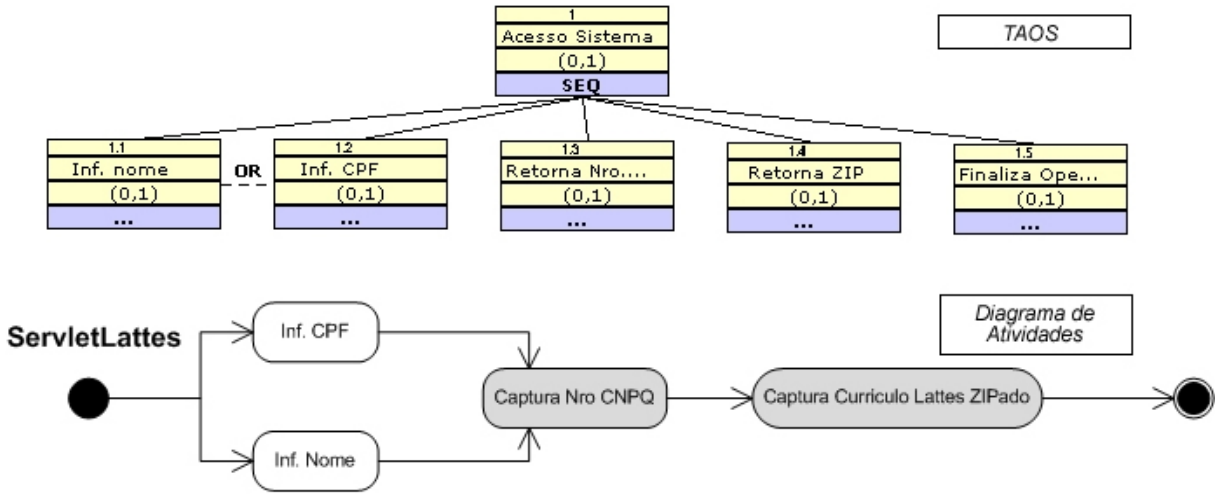


Figura 6.4: Comparação TAOS x Diagrama de Atividades
Fonte: Autor

No LATIN utilizamos o Microsoft Visio (Figura 6.5) para a editoração dos diagramas. Muitas outras ferramentas gratuitas e open-source poderiam ser utilizadas em seu lugar (como o JUDE, por exemplo), porém, como os alunos de computação e sistemas de informação da UFSC possuem licenças para sua utilização, optou-se por utilizar esta ferramenta. Deste modo não foi necessário o

aprendizado do uso de mais uma ferramenta, já que o Visio já era conhecido pela equipe.

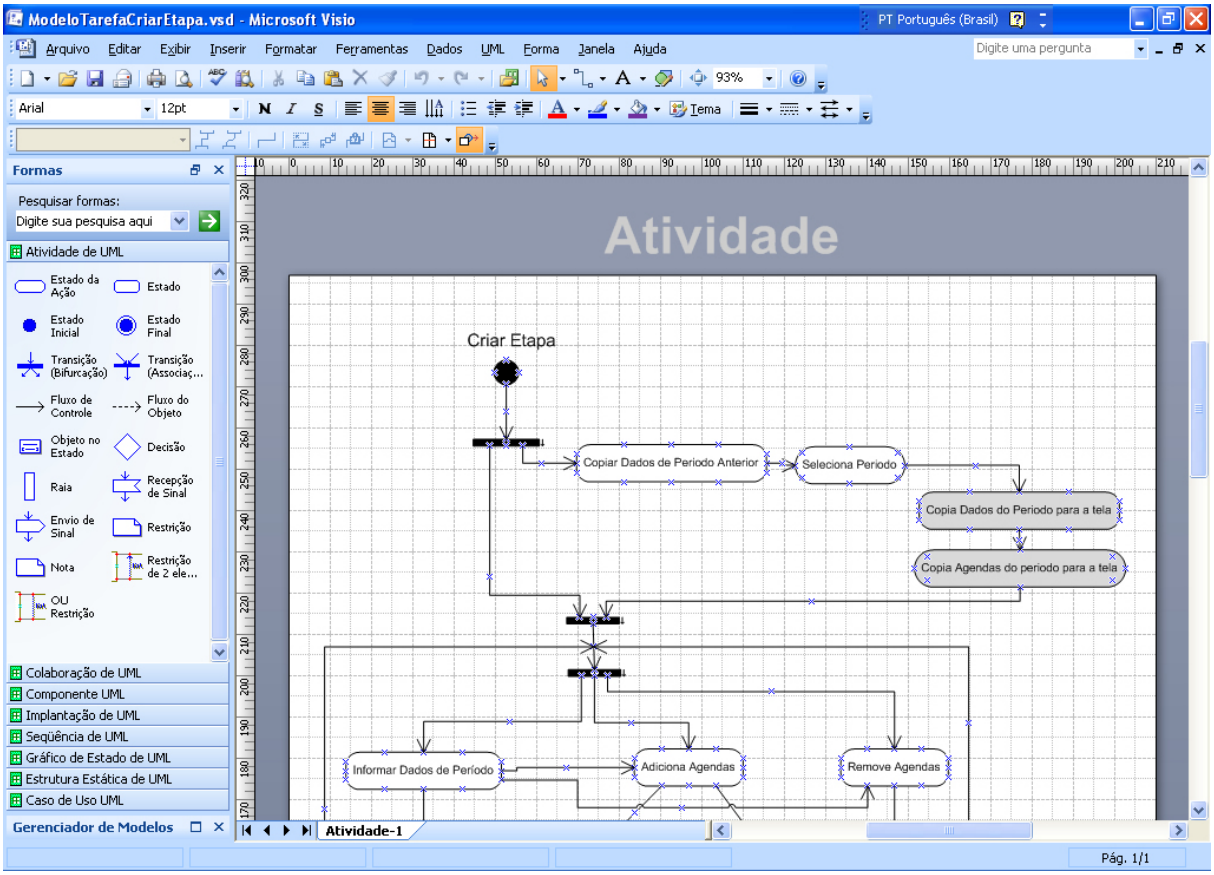


Figura 6.5: Microsoft Visio

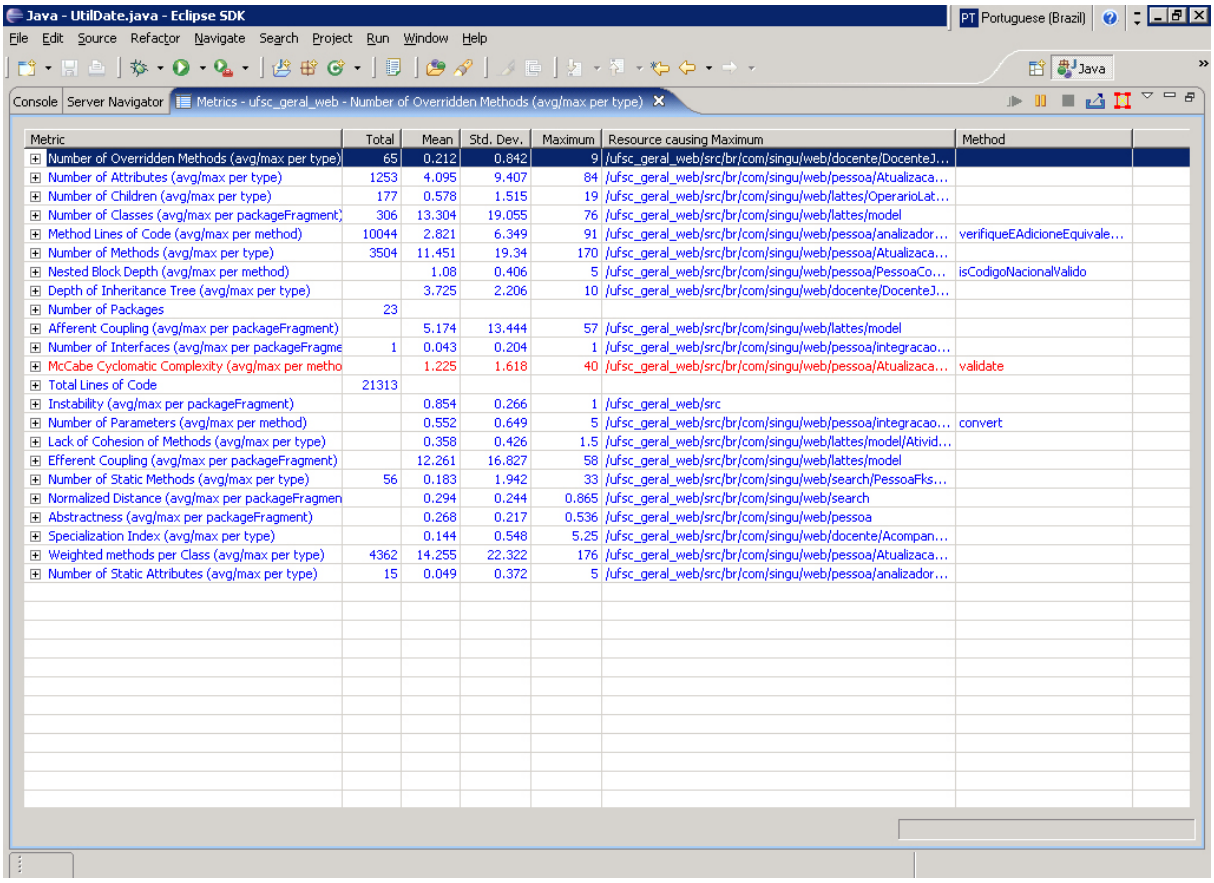
A utilização do Diagrama de Atividades foi aceita com facilidade pela equipe.

6.3.2 CAPTURA DE MÉTRICAS DO PROJETO VIA *SOFTWARE*

O YP sugere a utilização, para a montagem do *Big Chart*, de diversas métricas de código, como, por exemplo: número de classes, linhas de código, entre outras.

A adaptação sugerida para a captura destas métricas de código foi utilizar uma ferramenta, na forma de *plug-in*[16] para o *Eclipse*, para realização automática desta captura.

Após uma breve pesquisa das ferramentas existentes, optou-se pelo uso do *Eclipse Metrics*[17], por realizar também diagnósticos de qualidade de código. A ferramenta em uso é demonstrada na figura 6.6.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	65	0.212	0.842	9	/ufsc_geral_web/src/br/com/singu/web/docente/DocenteJ...	
Number of Attributes (avg/max per type)	1253	4.095	9.407	84	/ufsc_geral_web/src/br/com/singu/web/pessoa/Atualizaca...	
Number of Children (avg/max per type)	177	0.578	1.515	19	/ufsc_geral_web/src/br/com/singu/web/lattes/OperarioLat...	
Number of Classes (avg/max per packageFragment)	306	13.304	19.055	76	/ufsc_geral_web/src/br/com/singu/web/lattes/model	
Method Lines of Code (avg/max per method)	10044	2.821	6.349	91	/ufsc_geral_web/src/br/com/singu/web/pessoa/analizador...	verifiqueEAdicioneEquivale...
Number of Methods (avg/max per type)	3504	11.451	19.34	170	/ufsc_geral_web/src/br/com/singu/web/pessoa/Atualizaca...	
Nested Block Depth (avg/max per method)		1.08	0.406	5	/ufsc_geral_web/src/br/com/singu/web/pessoa/PessoaCo...	isCodigoNacionalValido
Depth of Inheritance Tree (avg/max per type)		3.725	2.206	10	/ufsc_geral_web/src/br/com/singu/web/docente/DocenteJ...	
Number of Packages	23					
Afferent Coupling (avg/max per packageFragment)		5.174	13.444	57	/ufsc_geral_web/src/br/com/singu/web/lattes/model	
Number of Interfaces (avg/max per packageFragme	1	0.043	0.204	1	/ufsc_geral_web/src/br/com/singu/web/pessoa/integracao...	
McCabe Cyclomatic Complexity (avg/max per metho		1.225	1.618	40	/ufsc_geral_web/src/br/com/singu/web/pessoa/Atualizaca...	validate
Total Lines of Code	21313					
Instability (avg/max per packageFragment)		0.854	0.266	1	/ufsc_geral_web/src	
Number of Parameters (avg/max per method)		0.552	0.649	5	/ufsc_geral_web/src/br/com/singu/web/pessoa/integracao...	convert
Lack of Cohesion of Methods (avg/max per type)		0.358	0.426	1.5	/ufsc_geral_web/src/br/com/singu/web/lattes/model/Ativid...	
Efferent Coupling (avg/max per packageFragment)		12.261	16.827	58	/ufsc_geral_web/src/br/com/singu/web/lattes/model	
Number of Static Methods (avg/max per type)	56	0.183	1.942	33	/ufsc_geral_web/src/br/com/singu/web/search/PessoaFks...	
Normalized Distance (avg/max per packageFragmen		0.294	0.244	0.865	/ufsc_geral_web/src/br/com/singu/web/search	
Abstractness (avg/max per packageFragment)		0.268	0.217	0.536	/ufsc_geral_web/src/br/com/singu/web/pessoa	
Specialization Index (avg/max per type)		0.144	0.548	5.25	/ufsc_geral_web/src/br/com/singu/web/docente/Acompan...	
Weighted methods per Class (avg/max per type)	4362	14.255	22.322	176	/ufsc_geral_web/src/br/com/singu/web/pessoa/Atualizaca...	
Number of Static Attributes (avg/max per type)	15	0.049	0.372	5	/ufsc_geral_web/src/br/com/singu/web/pessoa/analizador...	

Figura 6.6: *Eclipse Metrics*

Durante a reunião de acompanhamento, uma amostragem dos dados coletados pelo *Eclipse Metrics* é inserida nos locais apropriados no *Big Chart*, para que então sejam traçados os gráficos pertinentes.

Esta adaptação não é uma alteração na metodologia, apenas a automatização de um aspecto para a melhoria do processo.

---

16 "Na informática, um *plugin* ou *plug-in* é um (geralmente pequeno e leve) programa de computador que serve normalmente para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica." (*Wikipédia*, 2007g)

17 http://metrics.sourceforge.net/.

6.3.3 *RELEASES* COM DURAÇÃO VARIÁVEL

Como foi visto nas adaptações iniciais, o tempo de duração do projeto agora pode variar, não ficando preso ao sugerido de três meses e meio. A duração sugerida para um *release* era de duas iterações, de duas semanas cada, totalizando quatro semanas.

Porém, em determinadas ocasiões, este tempo de quatro semanas é muito longo, principalmente se um *release* planejado possuir poucas *User Stories* para implementar.

A equipe então decidiu que a duração de um *release* também pode ser variável, com um número qualquer de iterações, sendo que cada iteração deve ter no mínimo uma semana de duração. Deste modo é possível realizar um *release* rápido com poucas alterações críticas, que podem ser publicadas e entrar em uso rapidamente.

Entretanto, foi decidido manter, para a maioria dos *releases*, o sugerido de duas iterações, variando de uma a duas semanas cada.

6.3.4 ITERAÇÕES CURTAS PARA MANUTENÇÃO

Anteriormente, quando discutidas as alterações iniciais, foi mostrada a proposta de utilizar duas categorias de *release*: RC e FR. Em determinadas ocasiões um *release* pode precisar entrar em produção (i.e.: utilização pela instituição) com urgência. Um bom exemplo deste tipo de ocasião seria: a correção de um *bug* no *software* do Pedido de Matrícula da UFSC, às vésperas de um início de semestre letivo. Se neste caso, o *release* não fosse aprovado pelos usuários, todos os erros deveriam ser relatados aos desenvolvedores, para que então sejam transformados em *User Stories* de Manutenção, e só então implementados em um *release* tipo RC posterior, que estaria sujeito a novos testes antes de virar FR. Isto poderia tomar tempo demais.

Para solucionar este tipo de problema, a equipe sugeriu, quando em face de problema semelhante, a inclusão de uma nova iteração, porém mais curta (com duração de uma semana), para resolução dos *bugs* constatados pelos usuários (na

forma de *User Stories* de Manutenção). Ao final desta iteração curta, o mesmo *release* é novamente testado, e se aprovado, se transforma em um FR.

A diferença nesta abordagem é que este *release*, sem novas funcionalidades (*User Stories*), é aprovado com mais facilidade, visto que contém apenas correções de *bugs* e pequenos consertos na interface e/ou nos textos.

A inclusão de uma iteração curta em um *release*, entretanto, pode atrasar os *releases* seguintes. Por este motivo, a utilização desta técnica deve ser ponderada e avaliada de acordo com os riscos do projeto. Por este motivo, ficou determinado que a decisão sobre o uso desta técnica deve ser tomada pelo Gerente, em conjunto com os clientes.

6.3.5 *BIG CHART* COM DIVERSOS GRÁFICOS

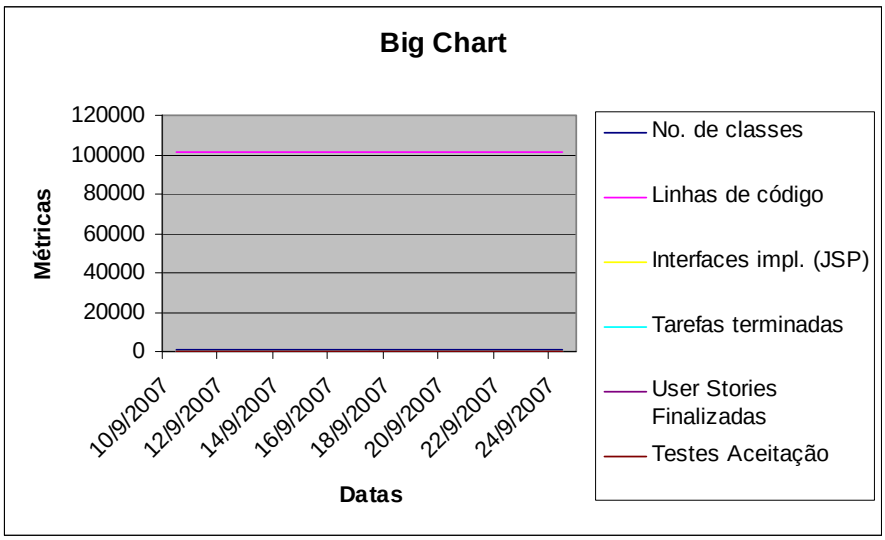
O *Big Chart*, importante artefato gerencial do YP, é um gráfico que mostra a evolução das métricas colhidas ao longo do tempo. De acordo com o proposto pelos autores do YP, o *Big Chart* é um gráfico único, que contém a evolução de todas as métricas nos intervalos das reuniões de acompanhamento.

Durante a primeira reunião de acompanhamento, notou-se, porém, que essa prática de juntar todas as métricas em apenas um gráfico para projetos em andamento causou um efeito indesejado: valores com grandezas desproporcionais. Ou seja, alguns dados com valores muito grandes e outros com valores muito pequenos fizeram com que a visualização da evolução dos dados no gráfico ficasse prejudicada. Isto pode ser notado no gráfico 6.1, onde dados como número de linhas de código e número de classes são muito grandes, e dados como tarefas realizadas e *User Stories* terminadas são muito pequenos.

A solução encontrada foi a de dividir o gráfico em diversos pequenos gráficos. Agora o *Big Chart* se divide em *Big Chart* Gerencial e Gráficos de Medição Auxiliares. Estes gráficos de medição auxiliar abrangem os aspectos mais técnicos do desenvolvimento, como: número de testes unitários implementados, *bugs* encontrados, linhas de código, arquivos em um projeto, número de classes implementadas, entre outras possíveis métricas.

A figura 6.7 mostra um exemplo do *Big Chart* dividido utilizado nos projetos do laboratório.

Gráfico 6.1: *Big Chart*



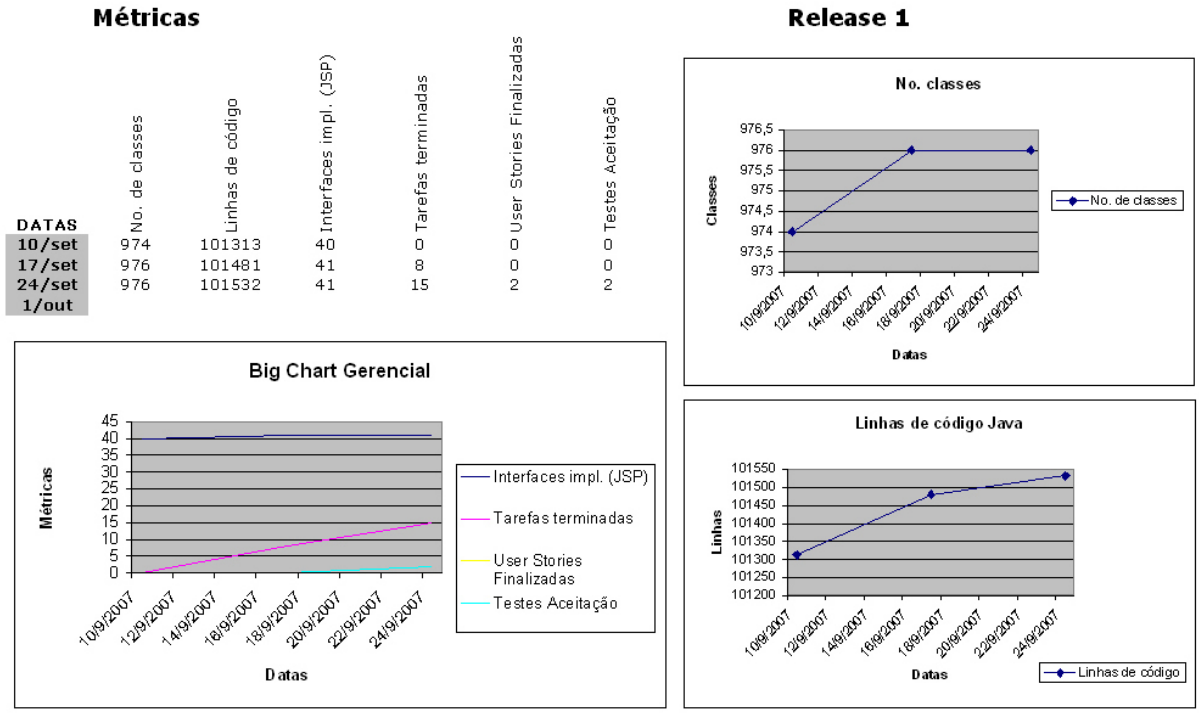


Figura 6.7: *Big Chart* com separação em Gerencial e Auxiliares

Deste modo, o gráfico fica mais organizado e a visualização mais compreensível. Para criar este gráfico pode-se utilizar um Editor de Planilhas Eletrônicas com opção de geração de gráficos qualquer.

6.3.6 ARTEFATOS VISÍVEIS A TODOS

O autor Alistair Cockburn (2002) propõe o uso do que ele chamou de “Radiadores de Informação”. Ele define que “radiadores de informação mostram informações em locais onde transeuntes podem vê-las”, e diz ainda que “Corredores se qualificam como bons Radiadores de Informação. Páginas na *WEB* não.”.

Com base nessa idéia, e na necessidade de realizar a gerência de versões dos artefatos, exigida pelo YP como responsabilidade do Gerente, foi proposto o uso de uma parede do laboratório como mural para exposição dos artefatos, sempre na versão mais recente dos mesmos. A figura 6.8 ilustra esta abordagem.

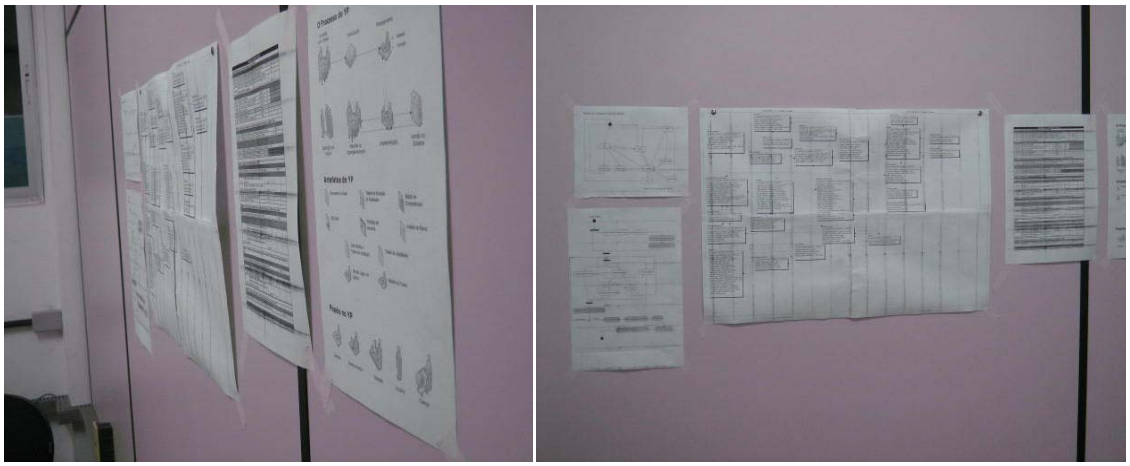


Figura 6.8: Radiador de Informação implantado no LATIN.

Deste modo, as versões mais recentes de todos os artefatos de uso geral do sistema estão sempre disponíveis para todos os membros da equipe. E é responsabilidade do Gerente do projeto manter este mural sempre atualizado.

6.4 PROBLEMAS ENCONTRADOS NA IMPLANTAÇÃO

Além das dificuldades que causaram adaptações na metodologia, descritas na seção anterior, alguns problemas foram encontrados durante a implantação do YP. Estes problemas, em geral, não puderam ser resolvidos com adaptações na metodologia.

Esta seção observa estes problemas, bem como apresenta soluções para os mesmos.

6.4.1 TESTES FUNCIONAIS MANUAIS

Uma das maiores dificuldades encontradas durante a implantação da fase de testes foi a implementação de testes funcionais das *User Stories* terminadas. Isto ocorreu por que os projetos desenvolvidos pelo LATIN são todos baseados em *WEB*, sendo basicamente sites dinâmicos que acessam bases de dados. Deste modo, as tecnologias para desenvolver testes funcionais são mais complexas.

Estes testes funcionais, se implementados, devem simular uma navegação pelo site desenvolvido, e após realizar operações nos cadastros, deve verificar se os dados inseridos nas bases de dados são coerentes com o que foi inserido nas interfaces.

Algumas ferramentas e *frameworks* foram sugeridos, e testados, pela equipe para tentar contemplar estas tarefas. Porém, a programação destes testes se mostrou uma tarefa complexa, e relativamente demorada.

Mas o aspecto que mais pesou na decisão de abandonar a tecnologia estudada foi o tempo de execução dos testes. Para verificar a coerência da base de dados, uma cópia desta base era criada antes da realização do teste, e após a execução esta cópia era comparada com o estado final da base de dados. Se as alterações na base não fossem exatamente as que foram determinadas no código do teste, este falharia. Isto demorava diversos minutos para rodar, para cada teste funcional implementado. Esta situação era inaceitável, pois além da demora para escrever os testes, ainda gastar-se-iam vários minutos para executá-los. Para realizar estes testes manualmente se gastam poucos minutos, muitas vezes segundos.

Desta forma a equipe decidiu que os testes de aceitação (normalmente compostos de testes funcionais) deveriam ser descritos pelos usuários e clientes como cenários de uso com resultados esperados. E esta descrição deveria guiar os testadores na realização dos testes. Esta prática, porém, está mais sujeita a falhas que testes implementados, pois conta com o fator humano como motor para os

testes. Portanto, os testes devem ser executados várias vezes pelos testadores, para que haja um alto grau de certeza quanto à avaliação dos mesmos.

6.4.2 UTILIZAÇÃO DE PADRÕES DE PROJETOS

A metodologia YP propõe, em suas boas práticas de programação, que se instaure no desenvolvimento a utilização de um *design* simples e de padrões de projeto na codificação.

Porém, como explicado no capítulo 2, a equipe de desenvolvimento tem muitas dificuldades em utilizar artifícios mais rebuscados de Engenharia de *Software* em seus códigos, e muitas vezes nem conhecem um padrão de projeto utilizado por outros membros mais experientes.

A utilização destes padrões nos projetos, causou, então, um pouco de dificuldade no entendimento de tarefas já implementadas, tanto para manutenção das mesmas quanto para eventuais refatorações.

Porém, a equipe decidiu continuar utilizando-os, se comprometendo a pesquisar mais sobre o assunto em bibliografias especializadas. Esta decisão veio da constatação de que estes padrões realmente deixam o código mais simples e dão ao *software* uma arquitetura mais elaborada. Desta forma, a equipe estará também evoluindo seus conhecimentos em técnicas de programação constantemente.

6.4.3 FALTA DE COMUNICAÇÃO COM OS USUÁRIOS

Um dos problemas que mais dificultaram a implantação da metodologia foi a falta de comunicação direta com os usuários dos sistemas em desenvolvimento.

Isto acontece por se tratarem de *softwares* desenvolvidos para a instituição, que serão utilizados por membros de diversos departamentos, e nos mais diferentes cargos dentro da instituição.

A cultura de desenvolvimento de *softwares* dentro da UFSC, e principalmente no LATIN, não é baseada na presença direta do cliente durante o desenvolvimento, sendo este um paradigma muito difícil de quebrar.

Mesmo com as adaptações realizadas inicialmente, de lançar *releases* de teste para capturar *feedback* dos usuários, ainda há uma falta de comunicação, e esta falta dificulta o entendimento de certas tarefas, e pode afetar, principalmente, a qualidade da implementação de interfaces e funcionalidades mais elaboradas, que exigem um alto grau de interação humano-computador.

Esta dificuldade encontrada apenas prova a importância da presença do cliente e de usuários no processo de desenvolvimento.

6.4.4 POUCA EXPERIÊNCIA NO DESENVOLVIMENTO DE TESTES UNITÁRIOS

Existe uma grande variedade de tecnologias disponíveis para a geração de testes unitários atualmente. Estas tecnologias possibilitam a implementação destes tipos de teste com muito mais rapidez. Porém, a técnica e a prática de desenvolver testes unitários não dependem da tecnologia.

A equipe não possuía, antes do YP, nenhuma experiência com testes unitários, ou qualquer outro tipo de implementação de testes, e teve certa dificuldade em aprender a técnica e a tecnologia. Muito tempo foi gasto inicialmente desenvolvendo-se testes, e muitas vezes estes testes acabam por serem insuficientes, ou pouco confiáveis. Isto ocorria por que os desenvolvedores não sabiam discernir o que deveria ser testado e por que.

A prática de testes em paralelo se mostrou adequada neste ponto, pois se os testes fossem desenvolvidos antes, possivelmente este processo de assimilar o desenvolvimento de testes seria muito mais árduo.

Eventualmente a equipe conseguiu superar esta dificuldade, e a prática de desenvolver testes unitários passou a ser algo comum.

6.4.5 ALTA ROTATIVIDADE DA EQUIPE

Como dito no capítulo 2, o LATIN possui uma equipe formada por estagiários. Na UFSC um estágio dura, normalmente, de 6 a 12 meses. Fora o fato de que, o mercado atual é muito favorável para desenvolvedores de *software*, principalmente para programadores com os conhecimentos adquiridos no laboratório. Estas características fazem com que haja uma constante troca na

equipe de desenvolvimento, gerando o efeito conhecido como Alta Rotatividade da equipe.

Apenas para se ter uma idéia, desde o início deste trabalho, a equipe mudou três vezes.

O YP foi desenhado para ser uma metodologia simples e fácil de aprender. Mas ainda assim, é algo que deve ser aprendido.

Um dos problemas que isto causa é o fato de que novos recursos humanos devem primeiro aprender a metodologia, para então começarem a trabalhar. E mesmo sendo isto um processo relativamente rápido (bastando apenas uma rápida introdução à metodologia e às práticas envolvidas), este novato ainda leva certo tempo para assimilar tudo que aprendeu.

Nestes casos, para facilitar esta assimilação, o Gerente deve tentar tirar ao máximo as dúvidas sobre a metodologia. Foi colocado também um documento, que resume de forma gráfica o YP, fixado ao Radiador de Informação do LATIN (este documento é mostrado no Apêndice B).

6.5 CONSIDERAÇÕES FINAIS SOBRE A IMPLANTAÇÃO

Após diversas semanas utilizando-se a metodologia YP no laboratório, e com base em tudo que foi descrito neste texto, algumas conclusões são obtidas quanto ao processo de implantação realizado.

O aspecto mais notado e comentado pela equipe foi o fato de que todo o processo de desenvolvimento ficou mais organizado, graças às práticas gerenciais do YP e do seu processo muito bem definido. Processo esse que foi implantado sem nenhum problema, já que a equipe conseguiu seguir todas as etapas com facilidade.

A modelagem das tarefas mais complexas utilizando formalismos, ao contrário do ceticismo inicial por parte dos desenvolvedores, acabou se mostrando uma poderosa ferramenta para o entendimento correto das atividades, e se tornou rapidamente uma atividade importante durante o planejamento de uma *User Story*, ou de uma tarefa mais complexa. A equipe aprendeu a desenvolver os diagramas de atividades com rapidez, e a utilizá-los para definir o funcionamento dos aspectos que

deveriam desenvolver. Desta forma, uma tarefa deveria realmente ser entendida antes de ser implementada, diminuindo dúvidas e impasses na codificação.

O uso de ferramentas para criação de artefatos e captura de métricas ajudou a acelerar o processo, e a equipe já pensa atualmente em outras formas de automatização de atividades relacionadas ao processo.

As dificuldades encontradas na implantação, em sua grande maioria, foram superadas, com ou sem adaptações na metodologia. Porém, alguns aspectos inerentes ao ambiente, como a alta rotatividade da equipe e a falta de experiência em programação por exemplo, ainda são problemáticos, e um meio de resolvê-los deve ser estudado.

De todos os aspectos da metodologia YP, a implantação gradual no laboratório até agora só não cobre os testes de usabilidade. Entretanto, a utilização destes testes já é estudada pela equipe, pois é um meio de se conseguir uma maior aproximação com os usuários do sistema.

Com tudo isto, pode-se dizer que a implantação do YP no Laboratório de Aplicação de Tecnologia da Informação vem sendo um sucesso, pois os resultados são visíveis: a gerência de tarefas consegue criar prazos realistas para os *releases,* os envolvidos relatam que possuem um maior entendimento das tarefas que devem realizar, tarefas mais simples estão sendo realizadas em algumas poucas horas, e tarefas mais complexas, que agora são divididas entre diversos membros, são desenvolvidas mais rapidamente em relação ao processo utilizado antes deste trabalho. E, além de todas estas melhorias, a metodologia está sendo bem aceita pela equipe, e sua utilização atualmente ocorre sem maiores dúvidas e dificuldades.

7 CONCLUSÕES

Neste capítulo serão apresentadas as conclusões para este trabalho, com base em todo o conteúdo estudado, nas técnicas e práticas utilizadas, nas experiências realizadas e nos conhecimentos adquiridos.

7.1 QUANTO AOS OBJETIVOS

Os objetivos específicos serão tratados inicialmente. Cada um deles será relacionado com os resultados obtidos na execução deste trabalho.

7.1.1 OBJETIVO ESPECÍFICO I

> *I – Realizar uma análise dos problemas existentes no processo de desenvolvimento atual, justificando e motivando a implantação de uma metodologia de desenvolvimento;*

Esta análise dos problemas no processo de desenvolvimento utilizado no LATIN, realizada em conjunto com toda a equipe, conseguiu de fato levantar os maiores problemas existentes. Esta prática fez com que a equipe conhecesse os seus problemas, e possibilitou a busca de soluções e a realização deste trabalho.

De todos os problemas levantados, de fato a equipe optou por tentar solucionar os relacionados à falta de uma metodologia.

7.1.2 OBJETIVO ESPECÍFICO II

> *II – Realizar uma revisão teórica sobre processo de desenvolvimento e metodologias de desenvolvimento, realizando uma pequena pesquisa sobre as mais utilizadas atualmente no mercado, como o XP, RUP, Scrum, Família Crystal e YP. Subseqüentemente deve-se realizar uma comparação entre as metodologias analisadas para que se possa escolher a mais adequada para a o problema.*

Esta pesquisa foi realizada com sucesso, e foi feita em bibliografia especializada no assunto, de autores conhecidos mundialmente pelos seus trabalhos com processos e metodologias, principalmente as ágeis. As metodologias foram estudadas de acordo com suas principais características, porém de forma

mais resumida, apenas para que se conhecessem seus princípios e teorias. A pesquisa sobre o YP foi realizada com base no material disponível no *web site*[18] da metodologia. Este material, apesar de possuir alguns erros, foi suficiente para um bom entendimento geral da metodologia.

Esta pesquisa proporcionou um maior entendimento do assunto deste trabalho, facilitando o seu desenvolvimento. Os exemplos contidos no material ajudaram também a entender melhor os artefatos que deveriam ser produzidos, facilitando a criação destes.

Foram utilizados também artigos e outros trabalhos sobre as metodologias de desenvolvimento para aprofundar ainda mais os conhecimentos e entender melhor as práticas relacionadas e técnicas utilizadas.

Com base em um trabalho acadêmico que tinha a proposta de comparar metodologias, uma base para comparação entre metodologias foi estudada e especificada. O estudo feito em (Abrahamson et al, 2002) foi expandido para que incluísse a metodologia YP, e foram abordadas neste trabalho apenas as metodologias estudadas e analisadas anteriormente: XP, *Scrum*, Família *Crystal* e RUP. A comparação foi realizada e possibilitou a escolha de uma metodologia, baseada nos dados obtidos, e não na experiência individual ou preferências do autor deste trabalho e da equipe do LATIN.

7.1.3 OBJETIVO ESPECÍFICO III

> *III – Planejar uma implantação da metodologia escolhida e em seguida implantar esta metodologia em projetos do LATIN. Durante o uso da metodologia os principais problemas ocorridos, bem como as melhorias obtidas, devem ser observados.*

A equipe planejou em conjunto, em reunião presidida e dirigida pelo autor deste trabalho, o processo de implantação da metodologia. Sugestões foram analisadas e assimiladas a este processo, que chegou ao seu formato final com cinco etapas bem definidas, sendo que duas delas formam iterações. Este planejamento se mostrou muito simples e fácil de ser seguido, e bom o suficiente

[18] http://www.dsc.ufcg.edu.br/~yp

para que o processo fique bem organizado. Por isso o processo de implantação foi realizado de acordo com o planejado sem maiores problemas ou dificuldades.

A metodologia foi implantada com sucesso no LATIN, sendo utilizada em todos os projetos até o fim deste trabalho, e possivelmente continuará sendo utilizada enquanto for adequada para os fins do laboratório. Esta implantação, além de proporcionar aos envolvidos uma maior experiência em metodologias e processos, também deu ao laboratório uma maior organização e uma capacidade de gerência dos projetos que o mesmo não possuía antes. A implantação contou com a cooperação de todos os envolvidos, que se interessaram pelo assunto e pela metodologia, e que, estudando e utilizando o YP nos projetos, possibilitaram o êxito deste objetivo.

Durante toda a implantação, foram recebidos constantes *feedbacks* dos membros da equipe de desenvolvimento do LATIN quanto ao uso da metodologia, principalmente durante as reuniões de acompanhamento e *workshops* de reflexão. Estes dados recolhidos possibilitaram a descoberta dos problemas ocorridos, e eventuais propostas de soluções para estes. Estes problemas, e soluções adotadas são descritos em detalhes no capítulo anterior. E foram também com base nestas informações (*feedback*) que se constataram as principais melhorias obtidas com o uso do YP: maior e melhor gerência do processo e das atividades, maior entendimento das tarefas do sistema e uma melhor previsão de riscos e dificuldades nos projetos.

7.1.4 OBJETIVO ESPECÍFICO IV

> *IV - Obter, após algumas semanas de uso da metodologia em projetos do laboratório, métricas ou dados para avaliar o impacto da implantação da metodologia no ambiente do LATIN. E, de posse destas métricas, realizar uma análise de possíveis melhoras no processo de desenvolvimento, decorrente da aplicação da solução implantada.*

Os apêndices C e D apresentam, respectivamente, o planejamento da primeira e da segunda iteração no principal projeto do LATIN. Analisando estes artefatos, que mostram um planejamento das tarefas a ser realizadas, e seus respectivos testes de aceitação, pode-se perceber que as duas primeiras semanas

(primeira iteração) foram planejadas para que houvesse um menor esforço para a equipe, para que se pudessem assimilar os conhecimentos necessários para o uso da metodologia. Já nas duas semanas seguintes, a equipe já planejou a realização de um número muito maior de tarefas, segura de que o entendimento da metodologia não seria mais um impasse na produção das mesmas. Com base nestes dados, pode-se concluir que o impacto da implantação na produção causou o planejamento de um número baixo de tarefas a realizar nas primeiras duas semanas, e que nas semanas seguintes este número cresceu bastante, pois a equipe já conhecia a metodologia e também possuía um melhor entendimento das tarefas a realizar. Futuramente, a tendência é que este número de tarefas cresça ainda mais, pois com base no histórico dos artefatos, pode-se estimar com mais clareza o esforço realmente necessário para a realização de tarefas semelhantes às já desenvolvidas no passado, e deste modo, cada envolvido pode produzir mais.

Anteriormente, o processo de desenvolvimento no LATIN era caótico. Nenhum dos envolvidos sabia o que, e como, determinada tarefa deveria fazer e como desenvolvê-la, e, mesmo se soubesse, não saberia o que fazer a seguir. Não havia também um entendimento dos projetos, pois tudo era feito sem planejamento, e só quem possuía alguma noção do propósito dos sistemas desenvolvidos eram os desenvolvedores que mantinham alguma comunicação com os clientes e/ou com os usuários, e, mesmo assim, ainda ocorriam erros de especificação, que conduziam a erros na implementação. Durante as primeiras semanas no YP, se notou um fortalecimento da parte gerencial do processo, e esses problemas diminuíram bastante. As práticas de gerência do YP possibilitaram um planejamento correto das atividades e as práticas de modelagem de tarefas deram um maior entendimento do problema aos desenvolvedores. A comunicação com os clientes se tornou especificação de *User Stories*, fazendo com que este conhecimento fosse passado para toda a equipe, que poderia acessar a informação a qualquer momento no Radiador de Informação implantado. As práticas de programação como: Integração Contínua, Propriedade Coletiva de Código e Padronização de Código já vinham sendo empregadas no desenvolvimento no laboratório há muito tempo, então seu advento oficial não foi muito sentido. O mesmo, porém, não se pode dizer das práticas de testes e modelagem. Inicialmente os desenvolvedores encontraram

dificuldades no seu uso, porém, quando isto se tornou uma rotina, os benefícios no uso destas práticas se tornaram evidentes. A modelagem possibilitou um maior entendimento das tarefas enquanto que a prática de testes acrescentou uma maior qualidade ao código desenvolvido, visto que o número de *bugs* diminuiu muito, e a conformidade com o planejado era verificada constantemente. Quanto à documentação do *software*, tarefa que antes do YP era ignorada por completo, agora é baseada nos artefatos gerados. A idéia é, após um *release*, ou ao final de um projeto, montar um documento com todos os artefatos criados para o projeto, com algumas descrições simples adicionadas. Isto foi feito para o projeto piloto (Apêndice A), e vendo sendo feito para os *releases* do projeto SINGU (o Apêndice E mostra o relatório de desenvolvimento para o primeiro *release* do projeto dentro da metodologia). Esta análise, de uma possível melhora no processo, é difícil de mensurar, sendo baseada somente nas opiniões dos envolvidos. Porém, no geral é possível notar uma grande melhora na produção de *software* no LATIN, devido aos aspectos, decorrentes da solução implantada, mencionados acima.

7.2 RESULTADOS OBTIDOS E EXPERIÊNCIAS ADQUIRIDAS

Nesta seção é apresentado um apanhado geral dos resultados obtidos na realização deste trabalho.

Durante o período de aproximadamente um ano e meio, este projeto vem sendo desenvolvido dentro do LATIN. Inicialmente com uma extensa pesquisa sobre o assunto e sobre metodologias utilizadas no mercado, e finalizando com a implantação de uma metodologia no laboratório.

A aplicação da metodologia YP deu ao processo de desenvolvimento do LATIN um dinamismo e um objetivismo que há muito era necessário, mas que não se sabia como alcançar. Pode-se dizer também que os envolvidos cresceram muito como desenvolvedores ao perceber as falhas no desenvolvimento e ao propor mudanças e adaptações a uma metodologia para melhorar seu trabalho.

Atualmente os projetos no laboratório são planejados, modelados e entendidos com clareza, e a implementação é muito mais precisa, possuindo mais qualidade. Apesar dos problemas que ainda existem, como a falta de treinamento,

de experiência e, muitas vezes, de perícia, pode-se dizer que um grande passo rumo a uma maior produtividade no processo foi tomado.

Mas, o maior benefício deste trabalho, para todos os envolvidos foi sem sombra de dúvida o conhecimento e a experiência adquiridos. Agora, os envolvidos estão habilitados a seguir uma metodologia, pois este trabalho já lhes deu um primeiro contato com uma que, apesar de acadêmica, se assemelha muito às comercialmente utilizadas mundo afora. Estão habilitados também a julgar se uma metodologia, ou determinada prática, é adequada para um determinado projeto, já que tiveram que realizar decisões semelhantes neste trabalho.

Portanto, mesmo se a metodologia YP cair futuramente em desuso no LATIN, o conhecimento que foi obtido com este trabalho e as experiências adquiridas no decorrer destes meses ficarão para sempre com os envolvidos neste projeto. Só por este fato já se pode concluir que este trabalho foi um sucesso.

7.3 QUANTO ÀS PERSPECTIVAS DE CONTINUIDADE

A implantação da metodologia YP foi um grande passo para o LATIN. Porém, definitivamente não deve ser o último passo tomado, já que o propósito do laboratório é a constante pesquisa de práticas e técnicas para o desenvolvimento de tecnologia na UFSC.

A metodologia YP também é uma grande fonte de pesquisas, com inúmeros trabalhos desenvolvidos, ou em desenvolvimento, visto que é uma metodologia com poucos anos de existência e ainda há muitas experiências com a mesma que podem ser relatadas em trabalhos científicos.

A seguir, podem-se listar diversas propostas de continuação deste trabalho, dentro ou fora do LATIN:

- Continuação da implantação do YP no LATIN, mas com aplicação de diferentes técnicas como Testes antes de codificar, ou diferentes formalismos para modelagem de tarefas;
- Criação de ferramentas para automatização e controle de tarefas e artefatos, como por exemplo: a criação automática do *Big Chart*

através de *plugins* de Captura de Métricas (como o *Eclipse Metrics*) e de Controle de Tarefas (como o Mylyn);

- Adaptação e utilização do YP em um ambiente comercial de desenvolvimento de *software*, ou em projetos *Open-Source*;
- Utilização e Adaptação de outras metodologias em projetos do LATIN em que o YP não é adequado;
- Análise e uso de ferramentas de gerência de projetos como *MS Project*, ou outras ferramentas *open-source*, para gerenciar as tarefas e recursos humanos, adequando-as ao YP, ou a outra metodologia qualquer.

Estas são apenas algumas poucas sugestões de trabalhos futuros, mas como a área da informática está sempre em constante evolução e crescimento, muitos outros trabalhos podem ser desenvolvidos nos assuntos que este trabalho aborda. Porém, se analisadas as necessidades atuais do LATIN, pode-se afirmar que as duas primeiras sugestões acima seriam as mais importantes.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, Pekka, SALO, Outi, RONKAINEN, Jussi, WARSTA, Juhani. **Agile software development methods. Review and analysis**. Espoo 2002. VTT Publications 478. 107 p.

AMBLER, Scott. **Agile Modeling: Effective Practices for eXtreme Programming and Unified Process.** John Wiley & Sons, 2002.

BECK, Kent: **Embracing Change with Extreme Programming**. IEEE Computing: 1999.

BECK, Kent. **Extreme Programming Explained: Embrace Change**. Addison-Wesley: 1999.

CARLOS, José. **Diagramas: Sequência e Atividades.** In: iMasters: Programação. 2005. Disponível em: <http://www.imasters.com.br/artigo/3004/uml/diagramas_seq%C3%BCencia_e_atividades/>. Acessado em 1 out. 2007.

COCKBURN, Alistair. **Agile Software Development: The Cooperative Game.** Addison-Wesley: 2002.

COCKBURN, Alistair. **Characterizing People as Non-linear, First-order components in Software Development**, 1999. Disponível em: <http://alistair.cockburn.us/index.php/Characterizing_people_as_non-linear,_first-order_components_in_software_development>. Acessado em: 20 jun 2007.

DIJKSTRA, Edsger W. **The Humble Programmer**. 1972. Disponível em: <http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html>. Acessado em: 19 jun 2007.

GARCIA, Francilene Procópio et al. **easYProcess: um processo de desenvolvimento de software para uso no ambiente acadêmico**. Campina Grande: UFCG, 2004. Disponível em: <http://www.dsc.ufcg.edu.br/~pet/Artigos/ARTIGO_YP.pdf>. Acessado em: 30 jun. 2007.

HAUNGS, Jim. **Pair Programming on the C3 project.** IEEE Computer: 2001.

HUMPHREY, Watts S. **Managing the Software Process**. Addison-Wesley: 1989.

IEEE, **IEEE Standard Glossary of Software Engineering Terminology**, IEEE std 610.12-1990, 1990.

FOWLER, Martin, **The New Methodology**, 2003. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acessado em: 18 jun 2007.

KROLL, Per. **Rational Unified Process Made Easy, The: A Practitioner's Guide to the RUP.** Addison-Wesley Professional: 2003.

KRUCHTEN, Phillipe. **The Rational Unified Process: An Introduction.** Addison-Wesley Professional: 1998.

LARMAN, Craig**. Utilizando UML e Padrões: Uma Introdução a Análise e ao Projeto Orientado a Objetos**. Prentice Hall PTR: 2000.

MEDEIROS, J. H. KAFURE, I. M. LULA, B. Jr. **TAOS: a Task-and-Action Oriented Framework for User's Task Analysis in the Context of Human-Computer Interfaces Design**. IEEE: 2000.

MURPHY, Craig. **Adaptive Project Management Using Scrum**. 2007. Disponível em <http://www.methodsandtools.com/archive/archive.php?id=18>. Acessado em 10 ago. 2007.

PET/UFCG. **easYProcess: Um processo de desenvolvimento de software.** Apostila. 2007. Disponível em <http://www.dsc.ufcg.edu.br/~yp/Download/YP%20Completo%20-%20%202007.pdf>. Acessado em: 12 jun. 2007.

SCHWABER, Ken, BEEDLE, Mike. **Agile Method Development With Scrum**. Prentice-Hall, 2002.

SCHWABER, Ken. **Scrum Development Process.** OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag, 1995.

SOL, Hank G. **A feature analysis of information systems design methodologies: Methodological considerations**. In: Olle, T. W., Sol, H. G. and Tully, C. J. (eds.). Information systems design methodologies: A feature analysis. Amsterdam, Elsevier: 1983.

SONG, Xiping, OSTERWEIL, Leon J. **Comparing design methodologies through process modeling**. 1st International Conference on Software Process, Los Alamitos, Calif., IEEE CS Press: 1991.

TAKEUCHI, Hirotaka, NONAKA, Ikujiro. **The New New Product Development Game.** Harvard Business Review: 1986.

___. IBM buys Rational for $2.1bn. **The Register**. 2002. Disponível em <http://www.theregister.co.uk/2002/12/09/ibm_buys_rational/>. Acessado em: 18 jun. 2007.

___. **JUnit.** Wikipedia, 2007. Disponível em <http://en.wikipedia.org/wiki/JUnit>. Acessado em 13 ago. 2007.

___. **Interactive Whiteboard**. Wikipedia, 2007. Disponível em <http://en.wikipedia.org/wiki/Interactive_whiteboard>. Acessado em 10 set. 2007.

___. **Agile software development**. Wikipedia, 2007. Disponível em <http://en.wikipedia.org/wiki/Agile_software_development>. Acessado em 10 jun. 2007.

___. **Metodologia (Engenharia de Software)**. Wikipédia, 2007. Disponível em <http://pt.wikipedia.org/wiki/Metodologia_%28engenharia_de_software%29>. Acesado em 16 jun. 2007.

___. **Dynamic Systems Development Method**. Wikipedia, 2007. Disponível em <http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method>. Acesado em 18 jun. 2007.

___. **HttpUnit**. Wikipedia, 2007. Disponível em <http://en.wikipedia.org/wiki/HttpUnit>. Acesado em 10 set. 2007.

___. **Plug-in.** Wikipédia, 2007. Disponível em <http://pt.wikipedia.org/wiki/Plugin>. Acessado em 1 out. 2007.

___. **Bug**. Wikipédia, 2007. Disponível em <http://pt.wikipedia.org/wiki/Bug>. Acessado em 1 out. 2007.

___. **Milestone.** Dictionary.com: 2007. Disponível em <http://dictionary.reference.com/search?q=milestone>. Acessado em 10 set. 2007.

APÊNDICES

APÊNDICE A

RELATÓRIO DE DESENVOLVIMENTO DO PROJETO SERVLETLATTES

Projeto ServletLATTES

1 Definição de Papéis

Equipe	Papéis
Elton Andrade dos Santos	Gerente, Testador, Usuário
Filipe Ferreira	Gerente, Desenvolvedor
Márcio Clemes	Cliente

2 Documento de Visão

2.1 Descrição do Sistema

O sistema deverá realizar a recuperação dos currículos LATTES dos docentes, através de um parâmetro fornecido ao sistema. O parâmetro pode ser o número do CPF do docente ou o nome completo do mesmo. Após o processamento, o sistema retornará o currículo LATTES de acordo com o parâmetro utilizado.

2.2 Requisitos Funcionais

Componente 01: Recuperar o currículo LATTES do docente:

- Recuperar currículo via número do CPF do docente
- Recuperar currículo via nome completo do docente

2.3 Requisitos Não Funcionais

- HTTP
- Tomcat 5.5
- Servlet
- JSP

2.4 Perfil do Usuário

O usuário do sistema é um sistema externo. A comunicação do sistema externo com este sistema é feita através de requisições HTTP, e possuem sempre a mesma sintaxe.

2.5 Objetivos de Usabilidade

Objetivo	Mensuração
Resposta rápida	Tempo de resposta do sistema quando da requisição de captura de um currículo qualquer

3 Modelo da Tarefa

Figura 1 – Modelo da tarefa do Sistema

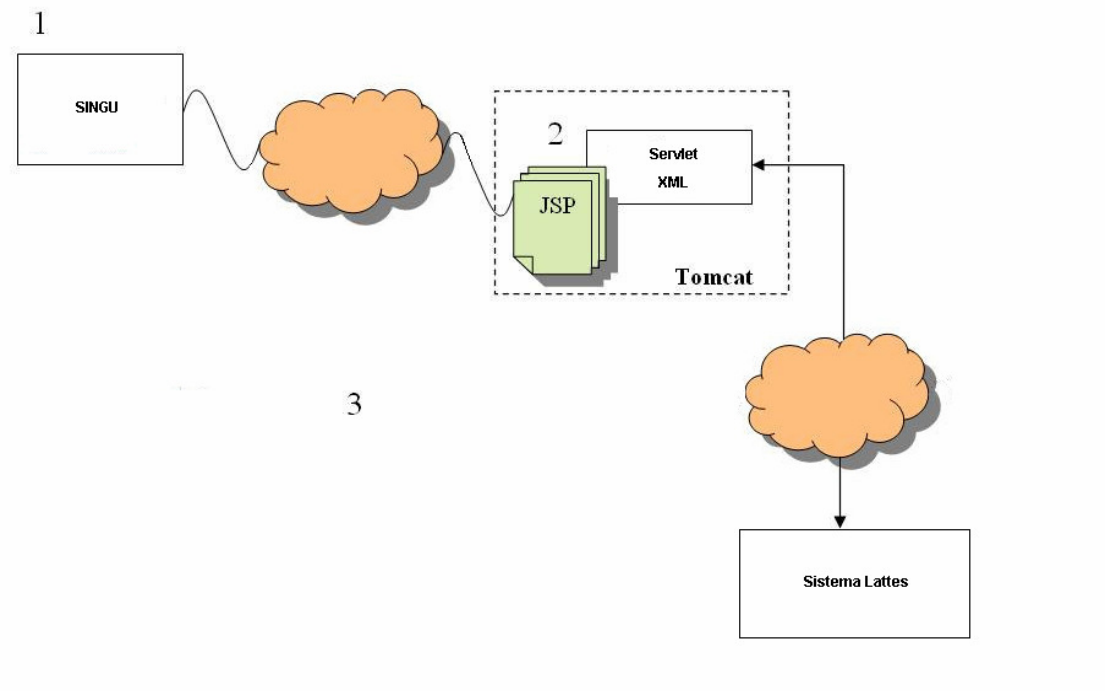
4 User Stories e Testes de Aceitação

US01	Estudar as tecnologias utilizadas: JSP, Tomcat, Servlet **Estimativa inicial: 20h**
TA1.1	Construir um exemplo utilizando as tecnologias
US02	Implementar funcionalidade de retorno dos currículos **Estimativa inicial: 20h**
TA2.1	Construir um exemplo que retorne o currículo de um docente qualquer
TA2.2	Construir um exemplo utilizando um parâmetro inválido (não pode retornar erros).
TA2.3	Construir um exemplo que retorne o currículo do desenvolvedor

5 Protótipo da Interface

Como o sistema é acessado via HTTP, e sua comunicação é estritamente sistema-sistema, não há necessidade de interface gráfica.

6 Projeto Arquitetural



6.1 Descrição do Projeto Arquitetural

A idéia da arquitetura do sistema é bastante simples... Trata-se de um sistema entre dois sistemas externos que não tem comunicação entre si. O sistema ServletLattes cria uma ponte entre estes dois sistemas, e possui uma arquitetura simples, apenas dois servlets que recebem a requisição de um sistema, repassa para o outro e retorna a resposta para o sistema requisitante.

7 Modelo lógico de Dados

Este projeto não possui dados que precisem de um modelo.

8 Plano de Release

8.1 Release 1

Release 1: 3/09 - 17/09	**Gerente: Elton**	
Iteração	**User Stories**	**Período**
Iteração 01	1	03/09 – 10/09
Iteração 02	2	10/09 - 17/09

8.1.1 Iteração 1

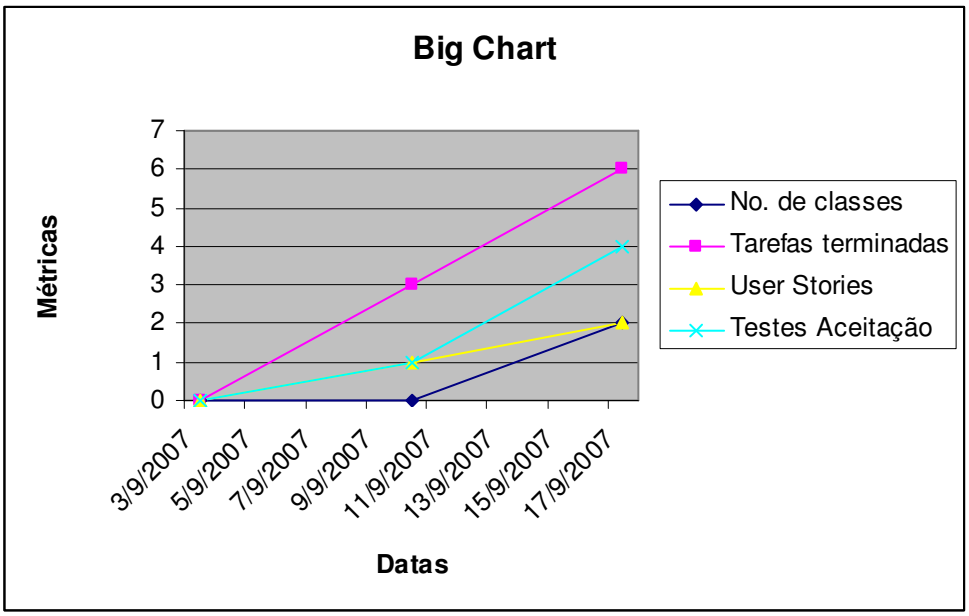
Iteração 01 - (03/09 – 10/09)					
US01 - Estudar as tecnologias utilizadas: JSP, Tomcat, Servlet, HTTPUnit					
Testes de aceitação				Resp.	Status
TA1.01	Construir um exemplo utilizando as tecnologias			Elton	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A1.01	Estudar as tecnologias faltantes (Servlet, Tomcat, HTTPUnit)	Filipe	10h	7h	OK
A1.02	Instalação e configuração do Tomcat	Filipe	2h	2h30	OK
A1.03	Criação de um exemplo	Filipe	4h	5h	OK

8.1.2 Iteração 2

Iteração 02 - (10/09 – 17/09)					
US02 - Implementar funcionalidade de retorno dos currículos					
Testes de aceitação				Resp.	Status
TA2.01	Construir um exemplo que retorne o currículo de um docente qualquer			Elton	OK
TA2.02	Construir um exemplo utilizando um parâmetro inválido (não pode retornar erros).			Elton	OK
TA2.03	Construir um exemplo que retorne o currículo do desenvolvedor			Elton	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A1.01	Desenvolvimento do ServletLattes com parâmetro CPF	Filipe	10h	8h	OK
A1.02	Desenvolvimento do ServletLattes com parâmetro Nome	Filipe	2h	1h	OK
A1.03	Publicação no Servidor 4.5	Filipe	2h	2h	OK

9 Big Chart

DATAS	No. de classes	Tarefas terminadas	User Stories	Testes Aceitação	Observações
3/set	0	0	0	0	Não houve nenhum tipo de reuso de código/páginas JSP Não foram contadas as classes escritas nos exemplos
10/set	0	3	1	1	
17/set	2	6	2	4	



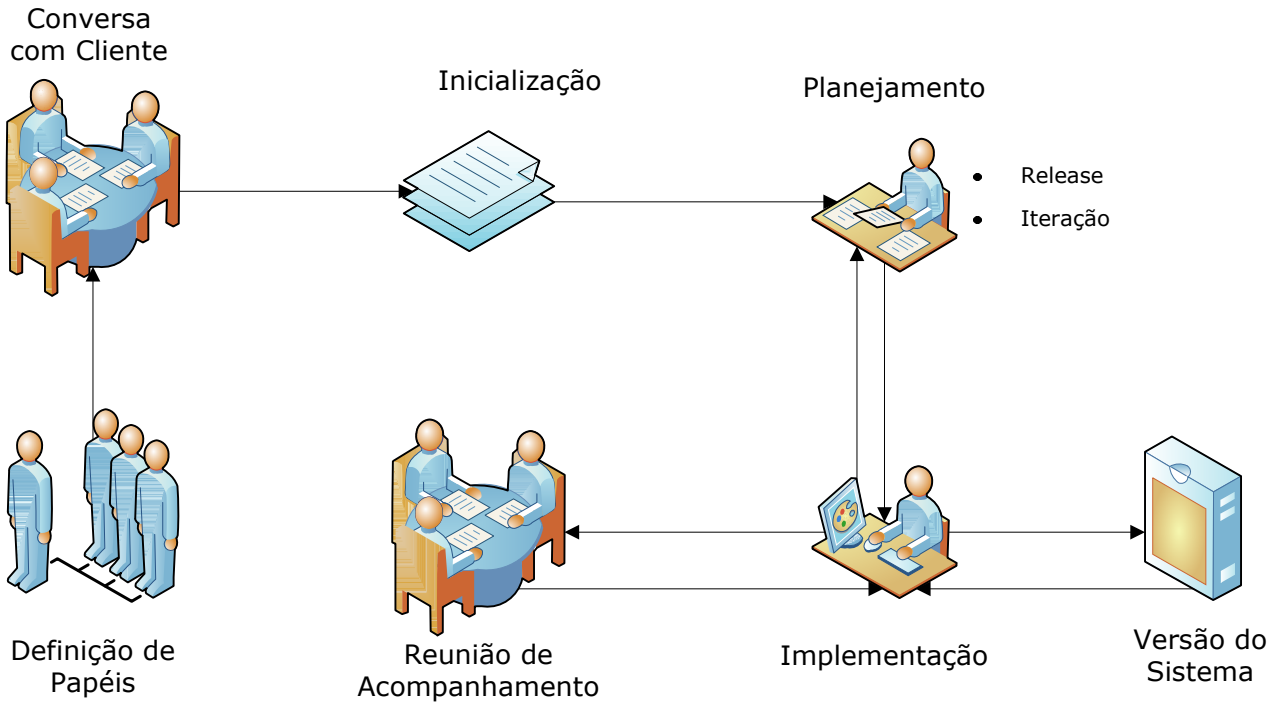
10 Análise de Riscos

Data	Risco	Prior.	Resp.	Status	Providência/Solução
03/09	Desconhecimento da metodologia YP	Alta	Todos	Superado	Realização de uma apresentação da metodologia aos envolvidos
04/09	Desconhecimento das tecnologias utilizadas	Média	Filipe	Superado	Estudar o assunto, e criar exemplos. Isto foi previsto no plano da iteração

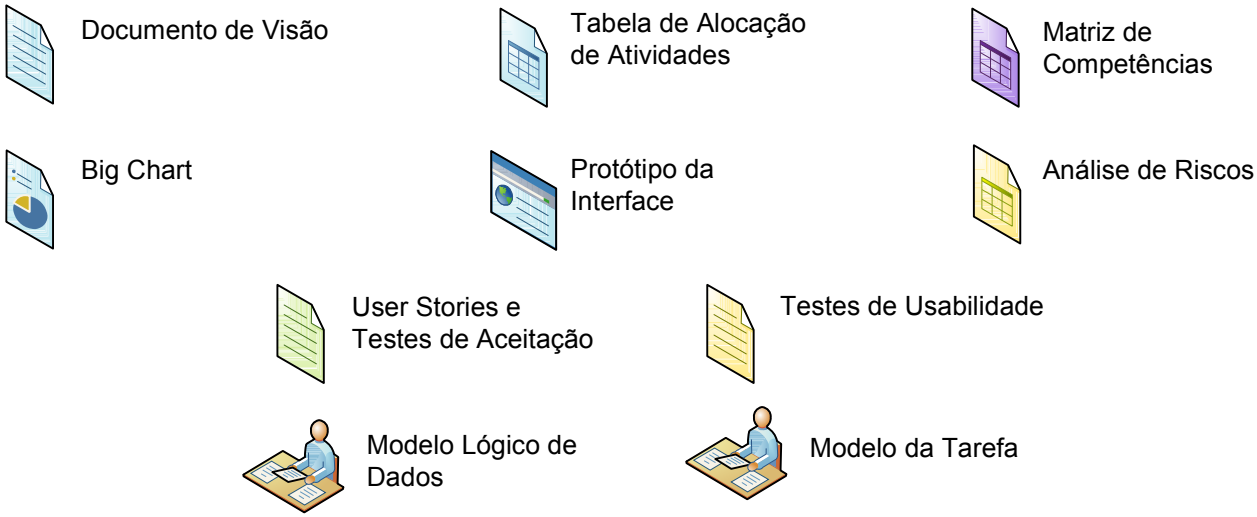
APÊNDICE B

RESUMO GRÁFICO DO YP (PARA O MURAL DO LATIN)

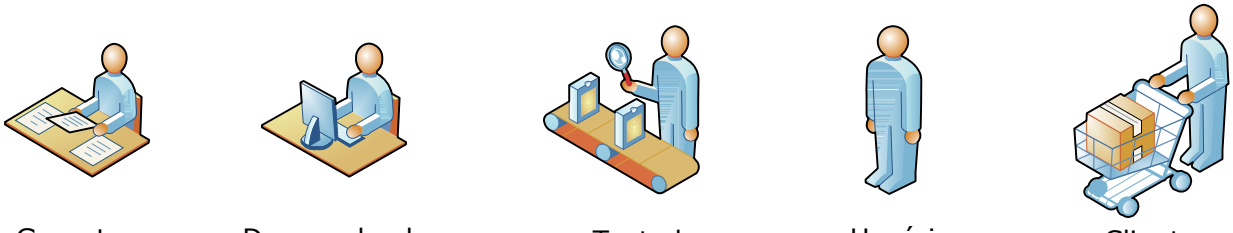
O Processo do YP



Artefatos do YP



Papéis no YP



APÊNDICE C

PLANEJAMENTO DA PRIMEIRA ITERAÇÃO – PROJETO SINGU/UFSC

Iteração 01 - (10/09 - 24/9)

US01 - Alteração da interface para o layout da UFSC

Testes de aceitação		Resp.	Status
TA1.01	Verificar se o novo layout satisfaz os clientes e usuários	Marcio	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A1.01	Estudar o manual de identidade visual da UFSC	Luis	2h		
A1.02	Estudar os arquivos de template do SINGU	Luis	4h		
A1.03	Alteração dos arquivos do template	Luis	8h		
A1.04	Realinhamento do Menu	Luis	4h		
A1.05	Verificação do padrão das páginas	Luis	4h		
A1.06	Testes nas interfaces	Luis	2h		
A1.07	Cores das interfaces	Luis	2h		

US02 -Implementar funcionalidade Criar nova Etapa

Testes de aceitação		Resp.	Status
TA2.01	Criar uma nova etapa com todos os dados informados (sucesso)	Luis	
TA2.02	Criar uma nova etapa com nenhum dado informado (erro)	Luis	
TA2.03	Criar uma nova etapa sem agendas copiadas (deve alertar o usuário)	Luis	
TA2.04	Criar uma nova etapa com todos os dados e agendas copiadas de um periodo anterior (sucesso)	Luis	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A2.01	Estudo do problema	Elton	2h		
A2.02	Criação da interface a partir do prototipo	Leonardo	4h		
A2.03	Lógica dos componentes da interface	Leonardo	3h		
A2.04	Estudo do modelo	Elton	4h		
A2.05	Lógica de Modelo (BD)	Elton	12h		
A2.06	Testes com interfaces dependentes	Elton	4h		

US03 - Relatórios

Testes de aceitação		Resp.	Status
TA3.01	Verificar se o novo layout satisfaz os clientes e usuários	Marcio	
TA3.02	Verificação dos dados na interface	Elton	
TA3.03	Verificação dos dados de um docente no relatório gerado	Elton	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A3.01	Verificação das funcionalidades atuais	Filipe	1h30		
A3.02	Verificação dos dados impressos	Filipe	3h		
A3.03	Mudança na organização e layout dos dados	Filipe	6h		
A3.04	Testes com diversos docentes e deptos.	Filipe	1h		

APÊNDICE D

PLANEJAMENTO DA SEGUNDA ITERAÇÃO – PROJETO SINGU/UFSC

Iteração 02 - (24/09 - 8/10)

US02 -Implementar funcionalidade Criar nova Etapa

Testes de aceitação		Resp.	Status
TA2.01	Criar uma nova etapa com todos os dados informados (sucesso)	Luis	
TA2.02	Criar uma nova etapa com nenhum dado informado (erro)	Luis	
TA2.03	Criar uma nova etapa sem agendas copiadas (deve alertar o usuário)	Luis	
TA2.04	Criar uma nova etapa com todos os dados e agendas copiadas de um periodo anterior (sucesso)	Luis	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A2.01	Estudo do problema	Elton	2h		
A2.02	Criação da interface a partir do prototipo	Leonardo	4h		
A2.03	Lógica dos componentes da interface	Leonardo	3h		
A2.04	Estudo do modelo	Elton	4h		
A2.05	Lógica de Modelo (BD)	Elton	10h		
A2.06	Lógica de Interface	Elton	4h		
A2.07	Testes com interfaces dependentes	Elton	4h		

US04 - Captura dos dados de produção bibliográfica do Lattes para o relatório final

Testes de aceitação		Resp.	Status
TA4.01	Verificar Funcionamento da tecnologia para captura dos currículos	Elton	
TA4.02	Verificação da correta captura dos dados de um docente	Elton	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A4.01	Estudo do problema e do subsistema do Lattes	Filipe	20h		
A4.02	Integração Singu/subsistema ServletLattes	Filipe	6h		
A4.03	Alteração das classes e interfaces necessárias	Filipe	12h		

US05 - Criação da interface principal (HOME)

Testes de aceitação		Resp.	Status
TA5.01	Verificar se o novo layout satisfaz os clientes e usuários	Marcio	-

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A5.01	Estudar quais tarefas são relevantes para os usuários	Luis	2h		
A5.02	Estudar como diferenciar o acesso ao sistema para Home pages diferentes	Luis/Leon.	4h		
A5.03	Prototipagem das novas páginas	Luis	2h		
A5.04	Implementação das novas páginas	Luis/Leon.	8h		
A5.05	Testes nas interfaces	Luis	2h		

US06 - Autenticação SSL

Testes de aceitação		Resp.	Status
TA6.01	Realizar login e verificar se o SSL está funcionando de acordo	Marcio	

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A6.01	Estudar a tecnologia SSL	Elton	4h		
A6.02	Estudar a integração SSL com Jboss	Elton	2h		
A6.03	Criação do Certificado	Elton	4h		
A6.04	Instalação no Jboss	Elton	4h		

USM01 - Manutenção interface Relatorio Plano Institucional / Departamental (US03)

Testes de aceitação				Resp.	Status
TAM1.01	Verificar novamente teste de aceitacao TA03.02			Elton	
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM1.01	Adicionar o status do plano do departamento na interface do relatorio departamental (visao da instituicao)	Leonardo	1h		
AM1.02	Adicionar o status do plano do departamento na interface do relatorio departamental (visao do depto.)	Leonardo	1h		

USM02 - Adição de link para visualização direta do relatorio individual de um docente apartir do PIA

Testes de aceitação				Resp.	Status
TAM2.01	Verificação da existencia dos links nos lugares corretos			Elton	
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM2.01	Adicionar link para acesso direto ao relatorio individual de atividades de um docente a partir de seu PIA (visao docente)	Leonardo	1h		
AM2.02	Adicionar link para acesso direto ao relatorio individual de atividades de um docente a partir de seu PIA (visao depto.)	Leonardo	2h		

USM03 - Alteração da interface principal para adequar ao padrao UFSC

Testes de aceitação				Resp.	Status
TAM3.01	Verificar se o novo layout satisfaz os clientes e usuários			Marcio	
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM3.01	Modificar barra superior do menu	Luis	8h		

APÊNDICE E

RELATÓRIO DE DESENVOLVIMENTO – PROJETO SINGU/UFSC (*RELEASE 1RC*)

Projeto SINGU-UFSC

1. Definição de papéis

Equipe	Papéis
Elton A. dos Santos	Gerente, Desenvolvedor, Testador
Douglas Schoeder (saiu da equipe)	Desenvolvedor, Testador
Filipe Ferreira	Desenvolvedor, Testador
Leonardo D'Agostini (saiu da equipe)	Desenvolvedor, Testador
Luís Artur Ribeiro de Lima	Desenvolvedor, Testador
Márcio Clemes	Cliente, Usuário
Marlene Costa da Silva	Usuário

2. Documento de Visão

2.1. Descrição do sistema

A Universidade Federal de Santa Catarina possui diversos cadastros de alunos, docentes, servidores, e outras pessoas físicas e jurídicas em geral, para diversos setores e/ou serviços disponíveis pela instituição. Assim, surgiu então, a necessidade de integrar-se esses dados, para que possam ser utilizados em funcionalidades que visam a instituição como um todo.

O projeto SINGU (ou Sistema Integrado de Gestão Universitário) surgiu então para tentar solucionar esta falta de integração entre as diversas bases. Se trata, então, de um sistema integrador, com aplicações específicas para os dados migrados.

Na primeira etapa do projeto, buscou-se meios de integração de dados, através de analisadores com Lógica Fuzzy, para filtrar os dados de todos os sistemas "externos" e migrar as informações para uma base de dados central, a base chamada de "Pessoa". Através de um processo de extração de dados, conseguiu-se atingir esse objetivo.

A segunda etapa, parte de aplicação desta integração, visa montar um sistema para o gerenciamento dos Planos de Atividades de Docentes, Departamentos e da instituição como um todo, gerando diversos dados e relatórios relacionados. É nesta etapa também que outros subsistemas, necessários para a criação dos planos de atividades (como o controle de Orientações dos docentes, por exemplo), serão criados.

2.2. Requisitos Funcionais

Fase 1 – Integração dos dados :

- Extração de dados das bases "externas".
- Analise dos dados extraidos
- Migração dos dados para uma base central

Fase 2.1 – Aplicação :

- Criação e controle de etapas de extração;
- Montagem (Automática) dos planos individual de atividades dos docentes (PIA);
- Montagem (Automática) dos planos de atividades dos departamentos (PAD);
- Montagem (Automática) dos planos de atividades da Instituição (PAI);
- Workflow de Aprovação de atividades entre docentes, departamento e instituição;

Fase 2.2 – Sistemas auxiliares :

- Cadastramento de Pessoas;
- Cadastramento de Órgãos;
- Cadastramento e controle de Orientações dos docentes (por parte dos docentes e do departamento);
- Cadastramento e controle de afastamento dos docentes (por parte dos docentes e do departamento);
- Cadastramento de Representantes da Instituição;

Fase 2.3 – Relatórios :

- Relatório Individual de Atividades dos Docentes (RIA);
- Relatório de Atividades dos Departamentos (RAD);
- Relatório de Atividades da Instituição (RAI);

2.3. Requisitos Não-Funcionais

- Interface WEB (JSP);
- Base de dados Sybase;
- Segurança – Login e senha para acesso ao sistemas e subsistemas;
- Arquitetura N Camadas;
- Utilização do framework SINGU, desenvolvido no próprio Laboratório.
- Padrão de projeto MVC para WEB.
- Javaserver Faces, EJB, iBatis.

2.4. Perfil do Usuário

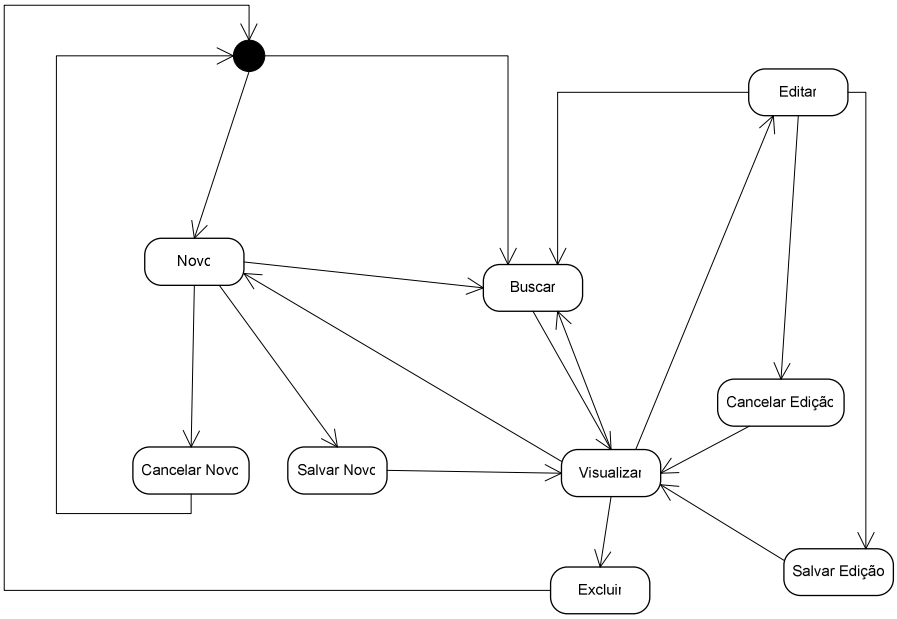
Usuários da UFSC, funcionários do Depto. e docentes. Conhecimento em informática variado.

2.5. Objetivos de Usabilidade

Objetivo	Mensuração
Resposta rápida	Tempo de resposta do sistema quando da requisição de captura de um currículo qualquer.
Facilidade de interação humano-computador	Um usuário não pode necessitar de muita ajuda ao utilizar o sistema. Menus devem ser explicativos

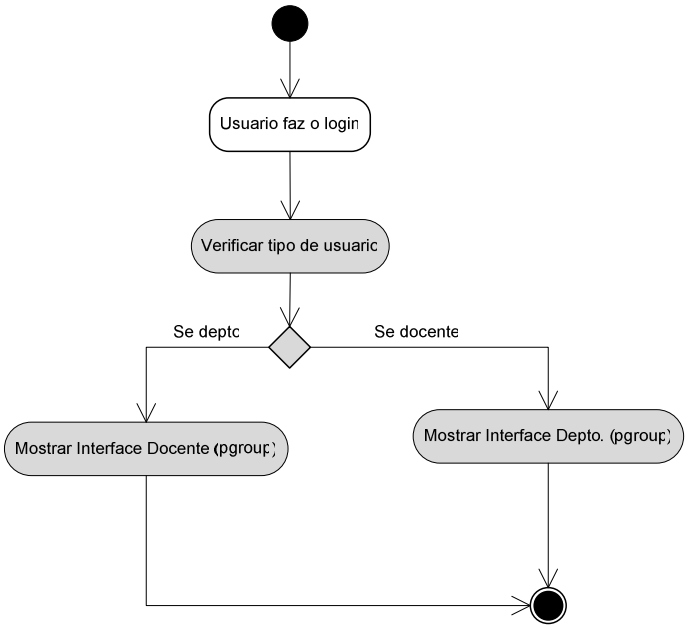
3. Modelos de Tarefas deste Release (RC1)

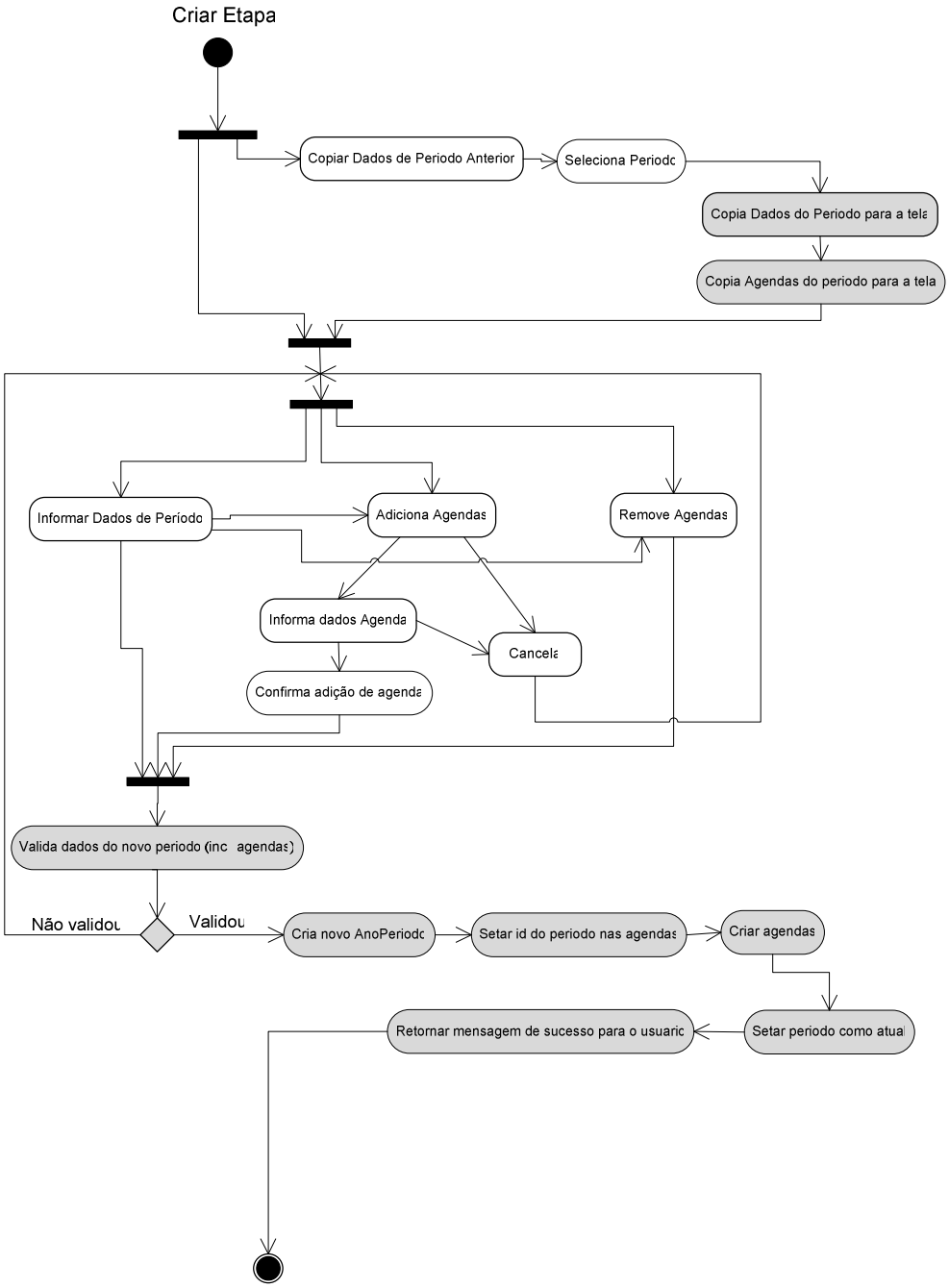
Modelo de Cadastro Simples SINGU

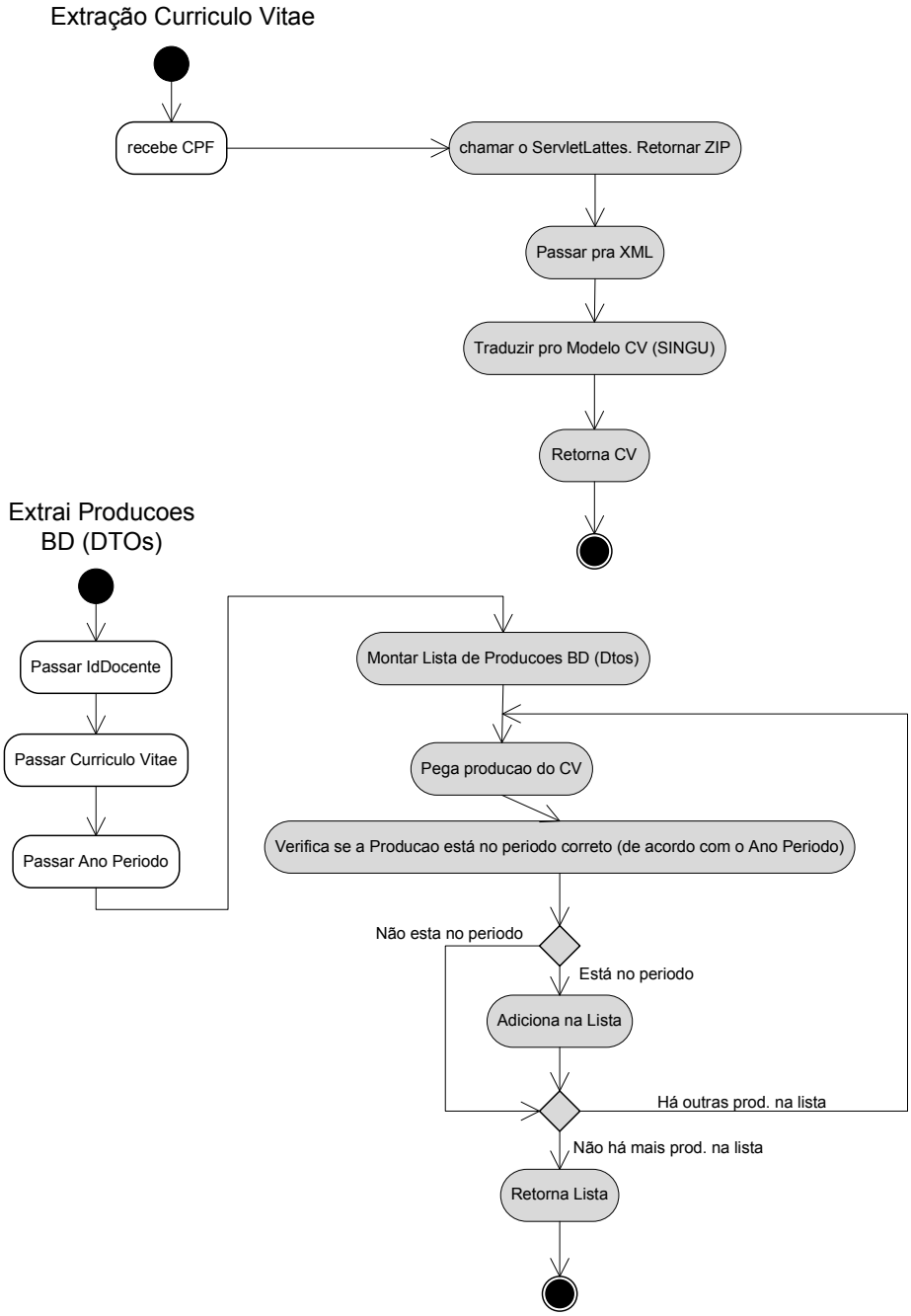


Máquina de estados da Toolbar

Tela Inicial do Sistema




Criar Etapa
Copiar Dados de Periodo Anterior
Seleciona Periodc
Copia Dados do Periodo para a tela
Copia Agendas do periodo para a tela
Informar Dados de Período
Adiciona Agendas
Remove Agendas
Informa dados Agenda
Cancela
Confirma adição de agenda
Valida dados do novo periodo (inc agendas)
Não validou
Validou
Cria novo AnoPeriodc
Setar id do periodo nas agendas
Criar agendas
Retornar mensagem de sucesso para o usuario
Setar periodo como atual


Extração Curriculo Vitae
recebe CPF
chamar o ServletLattes. Retornar ZIP
Passar pra XML
Traduzir pro Modelo CV (SINGU)
Retorna CV
Extrai Producoes
BD (DTOs)
Passar IdDocente
Passar Curriculo Vitae
Passar Ano Periodo
Montar Lista de Producoes BD (Dtos)
Pega producao do CV
Verifica se a Producao está no periodo correto (de acordo com o Ano Periodo)
Não esta no periodo
Está no periodo
Adiciona na Lista
Há outras prod. na lista
Não há mais prod. na lista
Retorna Lista

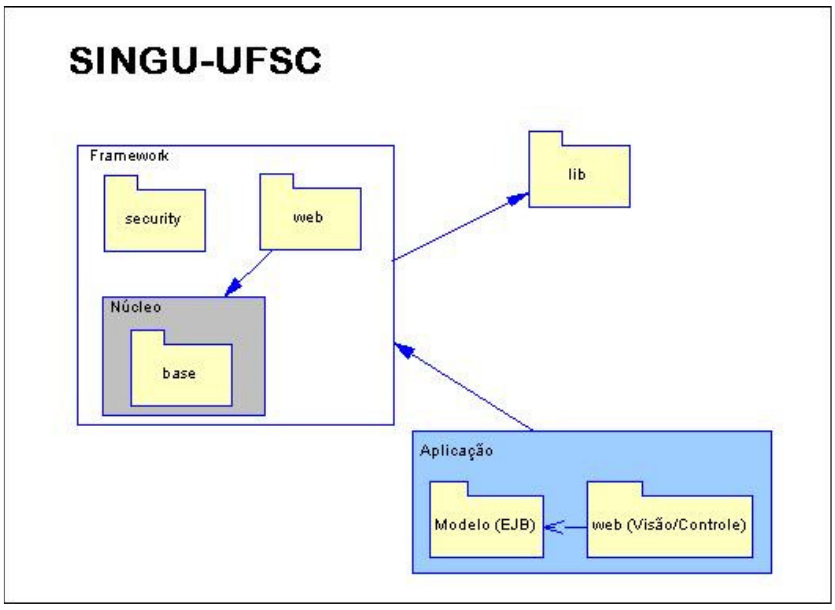
4. User Stories

USER STORIES	
Cod	**Descrição**
US01	Alteração da interface para o layout da UFSC **Estimativa Inicial: 30h**
TA1.01	Verificar se o novo layout satisfaz os clientes e usuários
US02	Implementar funcionalidade Criar nova Etapa **Estimativa Inicial: 30h**
TA2.01	Criar uma nova etapa com todos os dados informados (sucesso)
TA2.02	Criar uma nova etapa com nenhum dado informado (erro)
TA2.03	Criar uma nova etapa sem agendas copiadas (deve alertar o usuário)
TA2.04	Criar uma nova etapa com todos os dados e agendas copiadas de um periodo anterior (sucesso)
US03	Relatórios **Estimativa Inicial: 20h**
TA3.01	Verificar se o novo layout satisfaz os clientes e usuários
TA3.02	Verificação dos dados de um docente no relatório gerado
US04	Captura dos dados de produção bibliográfica do Lattes para o relatório final **Estimativa Inicial: 30h**
TA4.01	Verificar Funcionamento da tecnologia para captura dos currículos
TA4.02	Verificar se o novo layout satisfaz os clientes e usuários
US05	Criação da interface principal (Home) do sistema **Estimativa Inicial: 16h**
TA5.01	Verificar com clientes e usuários se a nova interface principal satisfaz o uso
US06	Adição de autenticação SSL no login do Sistema **Estimativa Inicial: 20h**
TA6.01	Realizar login e verificar se o SSL está funcionando de acordo

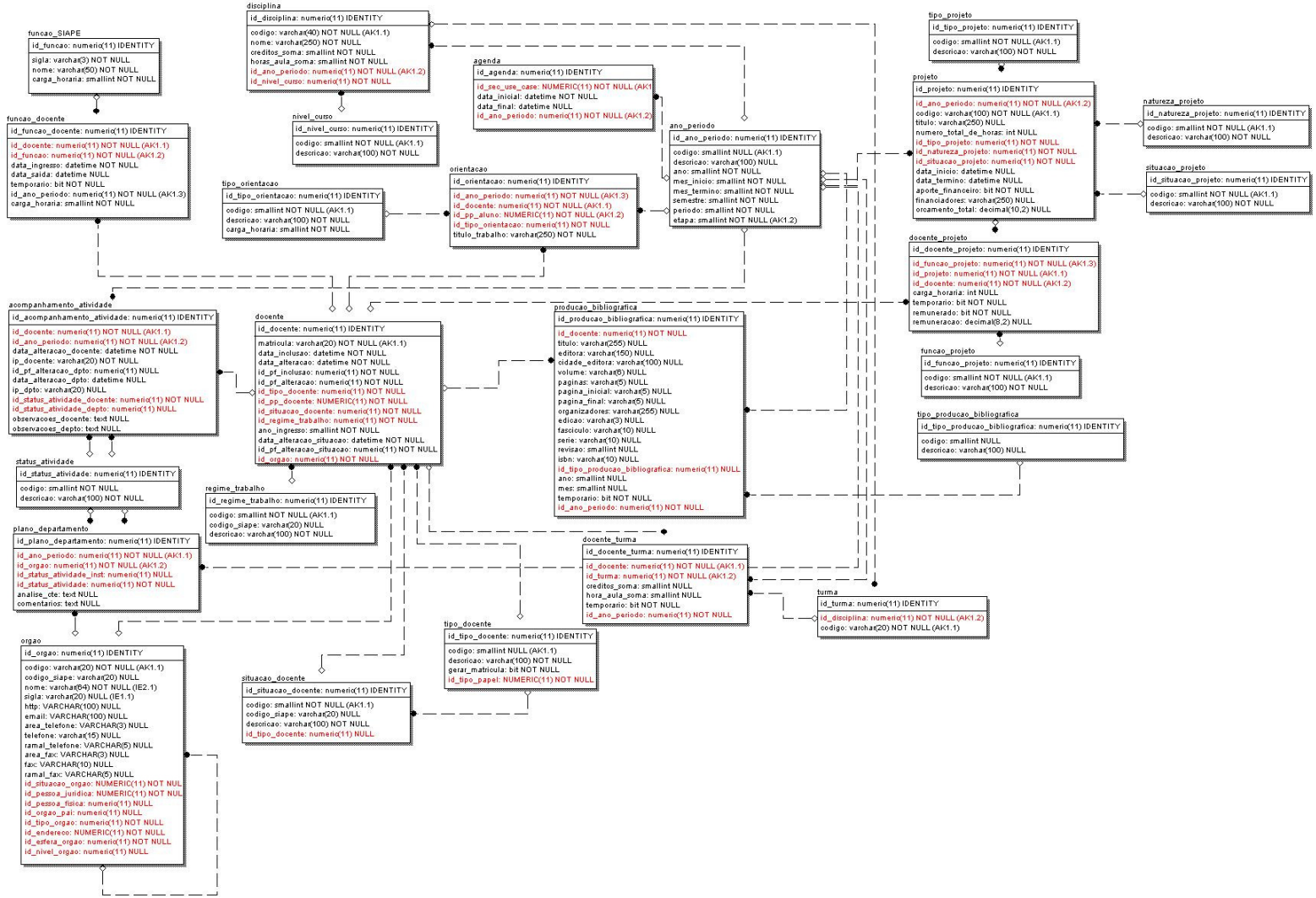
5. Protótipos de Interface

Realizados em Rascunhos .

6. Projeto Arquitetural



7. Modelo Lógico de dados



8. Plano de Release

PLANOS DE RELEASE - SINGU

Release 1: 10/09 - 8/10	**Gerente: Elton**	
Iteração	**User Stories**	**Período**
Iteração 01	1,2,3	10/09 - 24/09
Iteração 02	2,4,5,6,M1,M2	24/09 - 8/10

8.1. Plano da iteração 1

Iteração 01 - (10/09 - 24/9)					
US01 - Alteração da interface para o layout da UFSC					
Testes de aceitação				Resp.	Status
TA1.01	Verificar se o novo layout satisfaz os clientes e usuários			Marcio	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A1.01	Estudar o manual de identidade visual da UFSC	Luis	2h	1h	OK
A1.02	Estudar os arquivos de template do SINGU	Luis	4h	6h	OK
A1.03	Alteração dos arquivos do template	Luis	8h	6h	OK
A1.04	Realinhamento do Menu	Luis	4h	4h	OK
A1.05	Verificação do padrão das páginas	Luis	4h	4h	OK
A1.06	Testes nas interfaces	Luis	2h	2h	OK
A1.07	Cores das interfaces	Luis	2h	16h	OK
US02 -Implementar funcionalidade Criar nova Etapa					
Testes de aceitação				Resp.	Status
TA2.01	Criar uma nova etapa com todos os dados informados (sucesso)			Luis	
TA2.02	Criar uma nova etapa com nenhum dado informado (erro)			Luis	
TA2.03	Criar uma nova etapa sem agendas copiadas (deve alertar o usuário)			Luis	
TA2.04	Criar uma nova etapa com todos os dados e agendas copiadas de um periodo anterior (sucesso)			Luis	
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A2.01	Estudo do problema	Elton	2h	3h	OK
A2.02	Criação da interface a partir do prototipo	Leonardo	4h	5h30	OK
A2.03	Lógica dos componentes da interface	Leonardo	3h	12h	OK
A2.04	Estudo do modelo	Elton	4h	4h	OK
A2.05	Lógica de Modelo (BD)	Elton	12h		
A2.06	Testes com interfaces dependentes	Elton	4h		

US03 - Relatórios					
Testes de aceitação				Resp.	Status
TA3.01	Verificar se o novo layout satisfaz os clientes e usuários			Marcio	OK
TA3.02	Verificação dos dados na interface			Elton	FALHOU
TA3.03	Verificação dos dados de um docente no relatório gerado			Elton	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A3.01	Verificação das funcionalidades atuais	Filipe	1h30	1h	OK
A3.02	Verificação dos dados impressos	Filipe	3h	1h	OK
A3.03	Mudança na organização e layout dos dados	Filipe	6h	3h	OK
A3.04	Testes com diversos docentes e deptos.	Filipe	1h	0,5h	OK

8.2. Plano da Iteração 2

Iteração 02 - (24/09 - 8/10)					
US02 -Implementar funcionalidade Criar nova Etapa					
Testes de aceitação				Resp.	Status
TA2.01	Criar uma nova etapa com todos os dados informados (sucesso)			Luis	OK
TA2.02	Criar uma nova etapa com nenhum dado informado (erro)			Luis	OK
TA2.03	Criar uma nova etapa sem agendas copiadas (deve alertar o usuário)			Luis	OK
TA2.04	Criar uma nova etapa com todos os dados e agendas copiadas de um periodo anterior (sucesso)			Luis	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A2.01	Estudo do problema	Elton	2h		
A2.02	Criação da interface a partir do prototipo	Leonardo	4h		
A2.03	Lógica dos componentes da interface	Leonardo	3h		
A2.04	Estudo do modelo	Elton	4h		
A2.05	Lógica de Modelo (BD)	Elton	10h	8h	OK
A2.06	Lógica de Interface	Elton	4h	3h30	OK
A2.07	Testes com interfaces dependentes	Elton	4h	3h	OK
US04 - Captura dos dados de produção bibliográfica do Lattes para o relatório final					

Testes de aceitação				Resp.	Status
TA4.01	Verificar Funcionamento da tecnologia para captura dos currículos			Elton	OK
TA4.02	Verificação da correta captura dos dados de um docente			Elton	Ok
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A4.01	Estudo do problema e do subsistema do Lattes	Filipe	20h	12h	OK
A4.02	Integração Singu/subsistema ServletLattes	Filipe	6h	6h40	OK
A4.03	Alteração das classes e interfaces necessárias	Filipe	12h	11h	OK
US05 - Criação da interface principal (HOME)					
Testes de aceitação				Resp.	Status
TA5.01	Verificar se o novo layout satisfaz os clientes e usuários			Marcio	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A5.01	Estudar quais tarefas são relevantes para os usuários	Luis	2h	2h	OK
A5.02	Estudar como diferenciar o acesso ao sistema para Home pages diferentes	Luis/Leon.	4h	7h	OK
A5.03	Prototipagem das novas páginas	Luis	2h	6h30	OK
A5.04	Implementação das novas páginas	Luis/Leon.	8h	8h	OK
A5.05	Testes nas interfaces	Luis	2h	2h30	OK
US06 - Autenticação SSL					
Testes de aceitação				Resp.	Status
TA6.01	Realizar login e verificar se o SSL está funcionando de acordo			Marcio	OK
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
A6.01	Estudar a tecnologia SSL	Elton	4h	2h	OK
A6.02	Estudar a integração SSL com Jboss	Elton	2h	1h	OK
A6.03	Criação do Certificado	Elton	4h	1h	OK
A6.04	Instalação no Jboss	Elton	4h	5h	OK

USM01 - Manutenção interface Relatorio Plano Institucional / Departamental (US03)

Testes de aceitação		Resp.	Status
TAM1.01	Verificar novamente teste de aceitacao TA03.02	Elton	OK

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM1.01	Adicionar o status do plano do departamento na interface do relatorio departamental (visao da instituicao)	Leonardo	1h	1h	OK
AM1.02	Adicionar o status do plano do departamento na interface do relatorio departamental (visao do depto.)	Leonardo	1h	1h	OK

USM02 - Adição de link para visualização direta do relatorio individual de um docente apartir do PIA

Testes de aceitação		Resp.	Status
TAM2.01	Verificação da existencia dos links nos lugares corretos	Elton	OK

Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM2.01	Adicionar link para acesso direto ao relatorio individual de atividades de um docente a partir de seu PIA (visao docente)	Leonardo	1h	1h	OK
AM2.02	Adicionar link para acesso direto ao relatorio individual de atividades de um docente a partir de seu PIA (visao depto.)	Leonardo	2h	4h	OK

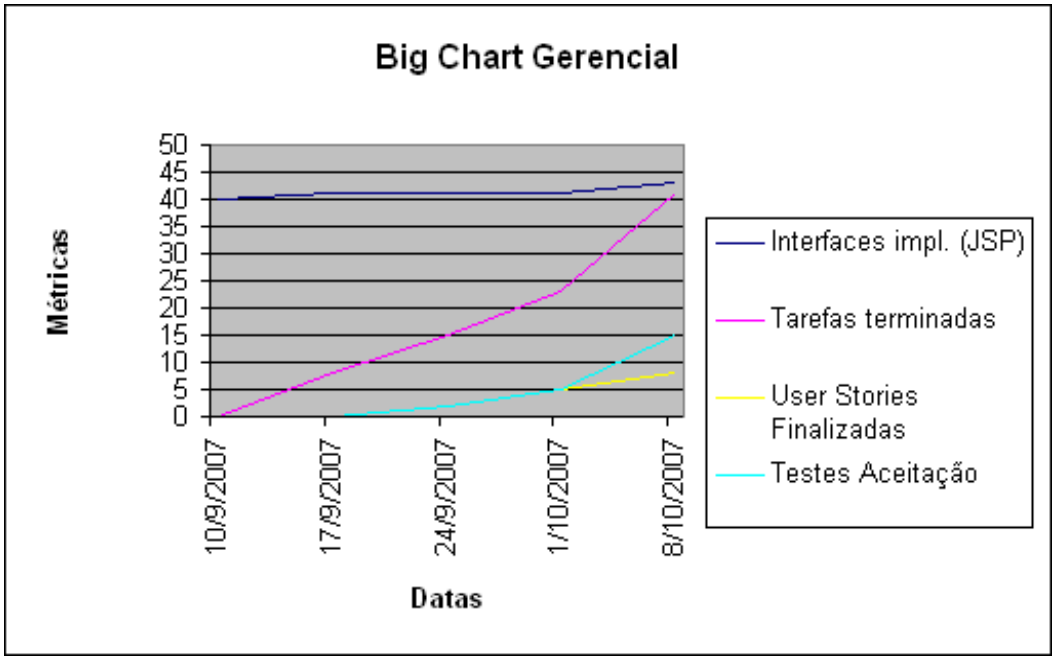
USM03 - Alteração da interface principal para adequar ao padrao UFSC

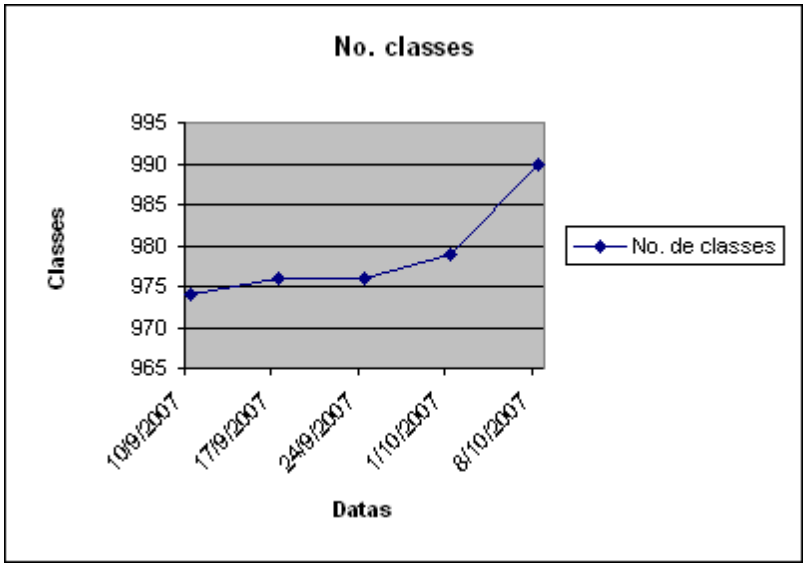
Testes de aceitação		Resp.	Status
TAM3.01	Verificar se o novo layout satisfaz os clientes e usuários	Marcio	OK

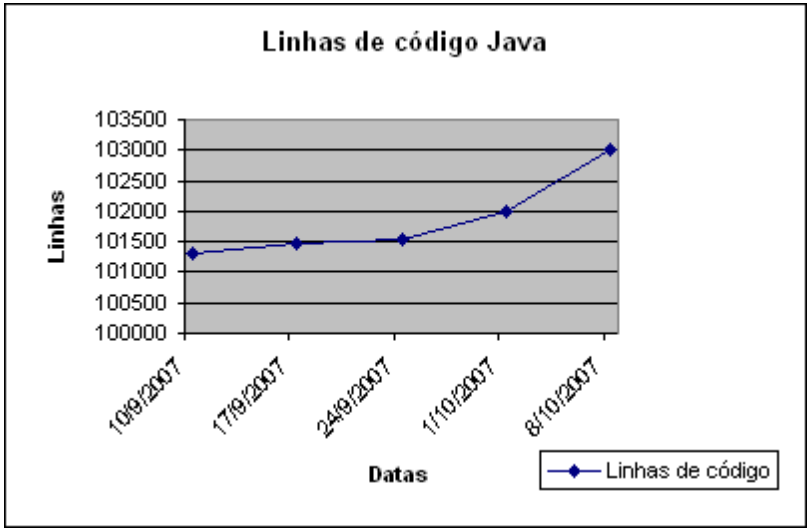
Atividade	Descrição	Resp.	Estimativa	Tempo real	Status
AM3.01	Modificar barra superior do menu	Luis	8h	12h	OK

9. Big Chart

DATAS	No. de classes	Linhas de código	Interfaces impl. (JSP)	Tarefas terminadas	User Stories Finalizadas	Testes Aceitação
10/set	974	101313	40	0	0	0
17/set	976	101481	41	8	0	0
24/set	976	101532	41	15	2	2
1/out	979	101992	41	23	5	5
8/out	990	102997	43	41	8	15







10. Análise de Riscos

Data	Risco	Prior.	Resp.	Status	Providência/Solução
10/09	Desconhecimento da metodologia YP	Alta	Todos	Superado	Realização de uma apresentação da metodologia aos envolvidos e *workshops* de reflexão para discussão da mesma
17/09	Desconhecimento do diagrama de seqüência	Alta	Todos	Superado	Inicialmente todos os diagramas devem ser desenvolvidos em conjunto com o Gerente. Após aprendido seu uso, então cada desenvolvedor deve criá-los separadamente.
24/09	Desconhecimento do framework para tratamento de Currículos Lattes	Média	Filipe Leon.	Superado	Estudar o assunto, gerar testes e utilizar exemplos.
24/09	Desconhecimento das classes JSFBean	Alta	Luis	ABERTO	Inicialmente o desenvolvedor deve programar em par com um membro mais antigo para aprender a tecnologia.
1/10	Desconhecimento da tecnologia SSL	Média	Elton	Superado	Estudar o assunto e a integração com o JBoss

APÊNDICE F

ARTIGO

Escolha e implantação de uma metodologia de desenvolvimento de software: um estudo de caso para o Laboratório de Aplicação em Tecnologia da Informação

Elton A. dos Santos

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil
freak@inf.ufsc.br

Abstract. *This work proposed the implantation of a software development methodology in the Laboratory for Application of Technology Information of the Federal University of Santa Catarina, as a way to solve problems that were inherent to the development process of the laboratory. For this it was researched the best-known methodologies, and among them one was chosen, adapted and implanted: the academic methodology easYProcess (YP). This work describes the whole process, pointing to the necessary adaptations for the implantation of this methodology in a real production environment.*

Resumo. *Este trabalho propôs a implantação de uma metodologia de desenvolvimento de software no Laboratório de Aplicação em Tecnologia da Informação, da Universidade Federal de Santa Catarina, como modo de resolver problemas inerentes ao processo de desenvolvimento do laboratório. Para isto pesquisou-se as metodologias mais conhecidas, e entre as mesmas uma foi escolhida, adaptada e implantada: a metodologia acadêmica easYProcess (YP). Este trabalho relata todo o processo, indicando quais as adaptações necessárias para a implantação desta metodologia em um ambiente de produção real.*

1. O LATIN

O Laboratório de Aplicação em Tecnologia da Informação (LATIN), laboratório que desenvolve aplicações de médio/grande porte para a Universidade Federal de Santa Catarina (UFSC), passava por problemas sérios de produtividade em seu processo de desenvolvimento de *software*. Isto era causado por um conjunto de problemas, alguns inerentes ao processo administrativo da instituição, e outros de ordem técnica. Por ter uma equipe formada em grande parte por estagiários, notava-se uma alta-rotatividade na equipe, bem como uma grande falta de experiência em programação. Aliado a isto, o processo não tinha organização, pois não havia metodologia para o desenvolvimento.

A equipe do LATIN analisou os problemas encontrados e, através de discussões entre seus membros, decidiu que, para resolver os principais problemas no processo, uma metodologia de desenvolvimento de *software* apropriada deveria ser implantada nos projetos do laboratório. Esta foi a motivação para este trabalho.

2. Estudo de metodologias e conceitos relacionados

A primeira etapa do trabalho se focou na pesquisa bibliográfica necessária para a criação de uma sólida base teórica sobre os assuntos de processos e metodologias.

Formada esta base, foram estudadas também as principais metodologias comerciais e acadêmicas disponíveis: *Extreme Programming* (XP), *Scrum*, *Rational Unified Process* (RUP), *Crystal Clear*, *Crystal Orange* e *easYProcess* (YP). Estas metodologias se enquadram em diversas categorias diferentes, sendo elas:

- Quanto ao "peso", termo cunhado por Cockburn (2002): Ágeis: XP, *Scrum*, Família *Crystal;* Tradicional: RUP; e Híbrida: YP;
- Comerciais: RUP, XP, *Scrum*, *Família Crystal;* Acadêmica: YP.

Cada metodologia foi estudada individualmente, com base em bibliografia especializada. Foram levantadas as principais características do processo, papéis e responsabilidades, práticas e princípios utilizados e artefatos. Em seguida foram traçadas as primeiras impressões sobre cada método.

3. Comparação entre metodologias

Após o estudo individual de cada um dos métodos supracitados, se fez necessário uma comparação entre os mesmos para que se pudesse escolher qual o mais apropriado para o problema de desenvolvimento do laboratório.

O método escolhido foi o método quasi-formal definido por Hank G. Sol (1983) e baseado nos estudos de Abrahamson et. al (2002) para comparação de metodologias. Neste estudo já eram comparadas diversas metodologias, porém como a YP não era abordada (por ser brasileira, acadêmica e muito recente), o estudo foi estendido para que a aborde também.

As metodologias foram comparadas em relação aos seguintes aspectos: pesquisas relacionadas, diferenciais e vantagens de cada abordagem, abrangência (do processo, das técnicas, práticas e artefatos e da gerência), tamanho da equipe necessária para a implantação e facilidade no aprendizado.

4. A escolha de uma metodologia adequada

Depois de realizadas as comparações, a equipe tem agora o conjunto de informações necessário para a escolha de uma metodologia apropriada. Não se esperava que fosse encontrada uma metodologia 100% apropriada e perfeita para o problema do LATIN, mas sim a mais adequada entre as estudadas, para que esta seja então adaptada para o problema, pois de acordo com Cockburn (2002) não é o projeto que deve se adequar a metodologia, e sim a metodologia que deve ser ajustada ao projeto.

Com base nestas pesquisas e comparações, e com o problema do laboratório, optou-se por escolher uma metodologia de fácil aprendizado, com elementos ágeis de desenvolvimento, com boas práticas de gerência, e com processo bem definido e robusto. Durante os estudos, notou-se um beco sem saída quando da escolha entre metodologias ágeis e tradicionais, pois uma metodologia ágil não seria a mais indicada para o problema do laboratório, dado a baixa experiência da equipe. Por outro lado, uma metodologia tradicional exigiria uma definição apropriada do processo, e o laboratório não possuía um núcleo estável para defini-lo. Optou-se pelo uso então da metodologia híbrida YP, visto que a mesma utiliza aspectos de metodologias ágeis e tradicionais, além de possuir processo robusto e o suporte desejado à gerência dos projetos.

5. Implantação da metodologia

Escolhida a metodologia para adaptar e implantar, fez-se necessário o planejamento da implantação, para que se pudesse analisar de antemão os impactos que esse processo poderia causar no laboratório. De acordo com as sugestões para implantação do XP, feitas por seu criador Kent Beck (1999), decidiu-se por uma implantação gradual, e cada intervalo gradual seria delimitado por um *Workshop* de reflexão, que é uma reunião para análise de metodologia sugerida por Cockburn (2002).

Todas as etapas planejadas para a implantação podem ser vistas na figura 1. As duas primeiras etapas são para análise e definição das primeiras modificações na metodologia, a terceira é a implantação da metodologia em um projeto pequeno, para que se analisem os impactos com um exemplo real da utilização do YP. As duas finais implementam as teorias de Beck e Cockburn para implantação gradual e adaptatividade contínua de uma metodologia.

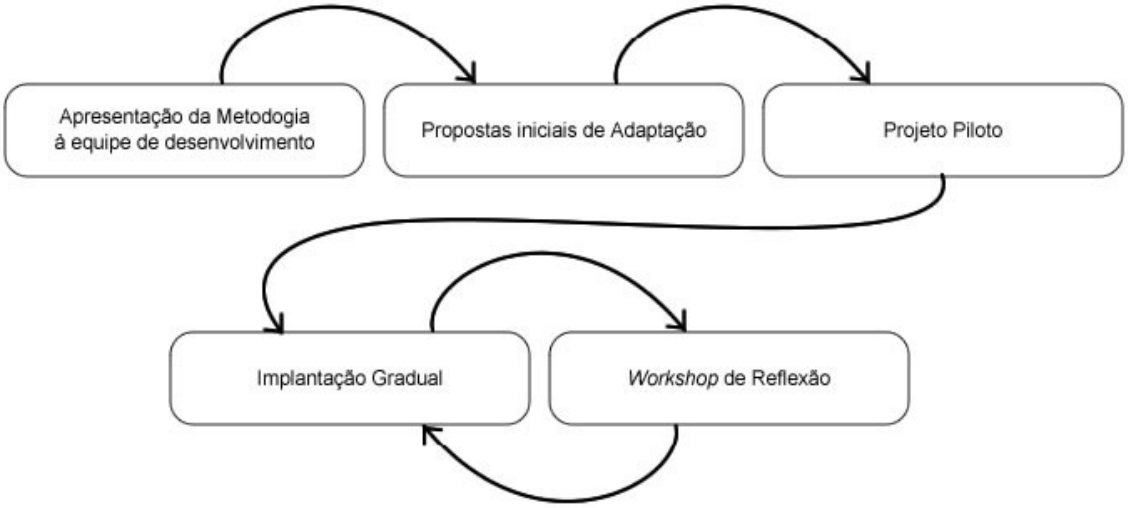


Figure 1. Planejamento da implantação

Nesta etapa também foram propostas métricas, ainda que empíricas, para medir o sucesso ou não da implantação. Porém, como é muito difícil se obter métricas para implantação de metodologias de desenvolvimento, optou-se por utilizar os *workshops* de reflexão como um termômetro da adaptação dos envolvidos com a metodologia, e a partir desta análise se concluir sobre o sucesso ou não desta implantação.

6. O processo de implantação

Após o planejamento da implantação iniciou-se o processo de implantação propriamente dito, seguindo as etapas planejadas. A seguir são relatados brevemente cada uma destas etapas.

6.1. Apresentação da metodologia à equipe de desenvolvimento

A apresentação da metodologia se deu de uma maneira sucinta, sem muitos termos técnicos, e com exemplos e ilustrações para facilitar o entendimento. Ao final da etapa a equipe estava apta a propor as primeiras alterações na metodologia.

6.2. Propostas iniciais de adaptação à metodologia

O maior problema levantado foi o fato de a metodologia ser voltada para trabalhos acadêmicos, sendo o primeiro passo adaptá-la para projetos comerciais e/ou institucionais. Estas adaptações alteraram aspectos do escopo dos projetos previstos pela metodologia para incluir propostas de manutenção de aplicativos e modificar as limitações de prazos para *releases*. Também foram definidos quais artefatos seriam utilizados, quais seriam adaptados se necessário, e quais os primeiros aspectos da metodologia seriam inseridos na primeira iteração da implantação gradual.

6.3. Projeto piloto

A idéia de implantação do YP primeiramente em um projeto piloto foi muito bem aceita pela equipe, e provou ser realmente uma boa opção. O projeto *ServletLattes*, que serviu como projeto piloto, era uma aplicação relativamente simples, que utiliza um *servlet* como ponte de comunicação entre dois sistemas distantes. Utilizaram-se praticamente todas as práticas propostas pela metodologia, mas já com as adaptações propostas pela equipe do LATIN.

O objetivo desta etapa, como dito, foi obter um *feedback* inicial do uso da metodologia, e como esperado, dificuldades foram encontradas, sendo as maiores no desenvolvimento utilizando testes unitários e funcionais, devido à baixa experiência da equipe com a tecnologia necessária. Optou-se por deixar o TDD (*Test Driven Development*) para uma etapa posterior da implantação gradual.

Poucas modificações na metodologia foram propostas nesta etapa. Isto é devido basicamente pela simplicidade do sistema desenvolvido, que apesar de proporcionar uma experiência inicial com o YP, foi muito pequeno para identificar problemas de longo prazo.

6.4. Implantação gradual

Em uma implantação gradual as mudanças no processo são implantadas aos poucos, para que o impacto inicial não seja grande demais e, neste caso, assuste os desenvolvedores.

A primeira iteração da implantação gradual implantou todo o processo do YP, a idéia de papéis e responsabilidades e as práticas de gerência sugeridas pelos autores da metodologia em PET/UFCG (2007). Os únicos aspectos que ficaram para iterações futuras foram as técnicas de modelagem de tarefas utilizando o formalismo *Task and Action Oriented System* (TAOS), o TDD e os testes de usabilidade.

6.5. *Workshops* de reflexão

Os *workshops* de reflexão do LATIN são reuniões de 15 a 30 minutos, onde é discutido o processo de desenvolvimento do laboratório, a metodologia YP e os projetos em desenvolvimento. Nestas reuniões os envolvidos têm a oportunidade de contribuir para a melhora do processo, propondo modificações na metodologia, na arquitetura dos projetos, entre outras questões técnicas. Estas reuniões serviram também, para este trabalho, como um termômetro da aceitação do YP no ambiente de trabalho. Através das opiniões e relatos de dificuldades no desenvolvimento, acarretadas pelas novas técnicas e princípios propostos pelo YP, foi possível decidir quando era a hora certa

para inserir novos princípios e técnicas no processo de desenvolvimento, realizando mais uma iteração da implantação gradual.

Em resumo, esta etapa complementa a anterior, que só pode ser declarada completada quando em um *workshop* puder se concluir que a equipe assimilou bem as mudanças implantadas.

7. Revisando o YP: adaptação durante o uso.

Durante o uso do YP em projetos reais do LATIN foi possível criar uma opinião mais concreta sobre a metodologia. Os *workshops* de reflexão serviram para identificar problemas na abordagem do YP, ou possíveis adaptações para melhorar o processo.

As principais modificações sugeridas, e conseqüentemente assimiladas ao processo, foram:

- Modificações em artefatos: a principal mudança foi a troca do TAOS pelo Diagrama de Atividades da UML, que já era conhecido pela equipe, diminuindo o impacto da utilização de um formalismo para modelar tarefas, aspecto que foi destacado pela equipe como muito importante para os projetos. O *Big Chart*, artefato de gerência do YP, também sofreu modificações leves, para facilitar sua visualização e entendimento;
- Adaptações para o uso em ambiente de produção: aumento do número de *releases* por projeto e aumento do escopo dos projetos, visto que o sugerido de 3 meses por projeto com 3 *releases* nem sempre serve para projetos comerciais.
- Uso de *softwares* para automatização de etapas do processo: Isto não foi bem uma modificação, apenas a pesquisa e implantação de programas para automatização de tarefas do processo, como o levantamento de métricas de código e o gerenciamento das tarefas.

8. Conclusões

Apesar de todos os problemas encontrados durante a implantação, o resultado do processo é muito positivo. Com a implantação do YP pôde-se notar uma grande melhoria na gerência dos projetos e planejamento de prazos. As melhorias também foram sentidas no entendimento das tarefas, visto que agora tudo deveria ser modelado e entendido antes da implementação em código da tarefa. O uso de testes unitários fez com que o código gerado tivesse uma qualidade superior ao processo caótico anterior, e os testes funcionais (ainda que executados manualmente) conseguem obrigar a implementação das funcionalidades do modo esperado pelo usuário.

Esta implantação, porém, não resolve todos os problemas do laboratório. A alta-rotatividade e baixa experiência em programação por parte dos envolvidos, acarretada pela falta de verba para alocação de funcionários para o laboratório e obrigando a contratação de estagiários de graduação para o desenvolvimento dos projetos, é um problema inerente ao modelo de administração da UFSC, sendo, portanto, impossível de ser tratado no momento.

Entretanto, as experiências e conhecimentos adquiridos no desenvolvimento deste trabalho ficarão para sempre com os envolvidos, que agora estão aptos a utilizar, adaptar e implantar metodologias de desenvolvimento, um conhecimento que não

receberiam na academia e que, com certeza, irá diferenciá-los no mercado. Só por este aspecto já é possível concluir que este trabalho foi um sucesso.

Referências Bibliográficas

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J (2002). “Agile software development methods. Review and analysis”. Edited by VTT Publications, Espoo.

Beck, K. (1999) “Extreme Programming Explained: Embrace Change”. Editado por Addison-Wesley.

Cockburn, A. (2002) “Agile Software Development: The Cooperative Game”. Editado por Addison-Wesley.

Cockburn, A. (1999) “Characterizing People as Non-linear, First-order components in Software Development”, http://alistair.cockburn.us/index.php/Characterizing_people_as_non-linear,_first-order_components_in_software_development. Junho.

PET/UFCG. (2007) “easYProcess: Um processo de desenvolvimento de software. Apostila. 2007”. http://www.dsc.ufcg.edu.br/~yp/Download/YP%20Completo%20-%20%202007.pdf. Junho.

Sol, H. G. (1983) “A feature analysis of information systems design methodologies: Methodological considerations”. In: Olle, T. W., Sol, H. G. and Tully, C. J. (eds.). “Information systems design methodologies: A feature analysis”. Editado por Elsevier, Amsterdam.