UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO
EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

PAULO ROBERTO MALLMANN

UM MODELO ABSTRATO DE GERÊNCIA DE SOFTWARE PARA METODOLOGIAS ÁGEIS

SÃO LEOPOLDO
2011

Paulo Roberto Mallmann

UM MODELO ABSTRATO DE GERÊNCIA DE SOFTWARE PARA METODOLOGIAS ÁGEIS

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos.

Orientador: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto

São Leopoldo
2011

CATALOGAÇÃO NA PUBLICAÇÃO (CIP)

M254m Mallmann, Paulo Roberto
Um modelo abstrato de gerência de software para metodologias ágeis / Paulo Roberto Mallmann - 2011.
136 f. : il.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Unidade Acadêmica de Pesquisa e Pós-Graduação, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2011.

Orientação: Sérgio Crespo Coelho da Silva Pinto

1. Informática – Software 2. Metodologias ágeis. 3. Agentes de software. 4. Engenharia de software. 5. Gerência de projetos. I. Título

CDU: 004.4

Ficha catalográfica elaborada por Maristela Hilgemann Mendel – CRB-10/1459

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Reitor: Dr. Marcelo Fernandes de Aquino
Diretoria da Unidade de Pós-Graduação e Pesquisa: Prof[a]. Dr[a]. Ione Bentz
Coordenador do PIPCA: Prof. Dr. Cristiano André da Costa

Dedico este trabalho
às pessoas mais importantes da minha vida:
minha esposa Cristiane e meus pais Paulo e Sônia.
Vocês foram muito importantes para mim
nessa caminhada.

AGRADECIMENTOS

Está se cumprindo mais uma etapa da minha vida. Conciliar mestrado, família e trabalho não é uma tarefa fácil. Inúmeras horas elaborando ideias, arquiteturas, modelos e discussões que pareciam ser intermináveis. E, além de tudo isso, muita implementação. Mas, no final das contas, só posso concluir que valeu todo este esforço.

Diante disso, gostaria de deixar meu singelo agradecimento a todos que, de alguma forma, contribuíram direta ou indiretamente para a execução deste trabalho:

- A Deus pelo dom da vida e por dar-me forças para lutar sempre;
- A minha esposa Cristiane por todo o amor, carinho e compreensão que teve comigo. Sei que nunca lhe agradecerei o suficiente;
- Aos meus queridos pais pela compreensão em todas as vezes que não me pude fazer presente;
- Ao meu orientador, professor Sérgio Crespo Coelho da Silva Pinto, pela sua orientação e paciência pois sem o seu apoio certamente eu não teria iniciado e muito menos terminado este trabalho;
- Aos professores Jorge Luís Victória Barbosa e João Carlos Gluz pela participação no seminário de andamento e por suas valiosas contribuições no refinamento do trabalho;
- Aos meus grandes amigos Pablo Dall'Oglio e Cândido Fonseca da Silva pelo apoio e pelas inúmeras trocas de ideias;
- Aos professores e funcionários do PIPCA que sempre estiveram disponíveis para ensinar e ajudar;
- A Unisinos por prover meios para que o trabalho em questão fosse desenvolvido;
- A CAPES pelo apoio financeiro concedido através da bolsa de estudos.

Por fim, meu muito obrigado e agradecimento especial a todos do fundo do meu coração.

RESUMO

Nos dias atuais uma boa gerência de projetos tem se tornado um fator competitivo no mercado pois influencia diretamente na qualidade do software. Muitas empresas e organizações voltadas para o desenvolvimento de software investem na melhoria de seus processos de desenvolvimento. No entanto, o mercado atual está exigindo destas empresas uma maior rapidez e uma adaptação a ambientes de negócio bastante dinâmicos.

Para atender a esta realidade surgiram as metodologias ágeis que dão suporte a esta nova realidade. No entanto, as ferramentas atuais de gerência de projetos mais utilizadas não apresentam uma solução que permite o planejamento integrado de atividades gerenciais e de atividades de desenvolvimento. Esta deficiência de integração pode resultar em distorções no planejamento de projetos ocasionando atrasos na entrega do software.

Diante disso, o objetivo deste trabalho foi criar um ambiente de gerência de software para metodologias ágeis que permitisse uma gerência de projetos de software de forma personalizada em ambientes de desenvolvimento que utilizam conceitos e características de metodologias ágeis. Para isso ser possível, foi definido um modelo dinâmico e modular que contempla todos os aspectos relacionados à gerência de projetos e metodologias ágeis que pode ser instanciado gerando modelos concretos específicos para cada projeto. Esta dinamicidade dada ao modelo foi o grande diferencial frente aos outros trabalhos avaliados.

Baseado no modelo abstrato proposto foi criado um ambiente de gerência de projetos que explorou todo o seu potencial. Este ambiente é apoiado por um conjunto de agentes de software que mantêm o modelo concreto atualizado. Sobre o modelo concreto, podem ser extraídos relatórios gerenciais e gráficos de *gantt* dinâmicos que o deixam ainda mais flexível, de forma a dar ao gestor do projeto uma visão macro e precisa do andamento das atividades.

Como forma de validar o trabalho proposto, foi realizado um estudo de caso em uma empresa externa. A aplicação desenvolvida foi utilizada durante o desenvolvimento de um projeto real e trouxe como resultado concreto uma maior segurança para a gestão de projetos. Sua utilização permitiu detectar falhas rapidamente dando mais tempo para ajustar os impactos no cronograma, antes feitos somente através de conversas individuais ou reuniões de acompanhamento diretamente com a equipe.

Palavras-Chave: Metodologias ágeis. Gerência de projetos. Agentes de software. Engenharia de software

ABSTRACT

Nowadays a good project management has become a competitive factor in the market because it directly influences the quality of software. Many companies and organizations for the development of software invest in improving their development processes. However, the current market is requiring these companies to more quickly and adapt to highly dynamic business environments.

To meet this reality emerged agile methodologies that support this new reality. However, current tools in project management more used does not have a solution that enables integrated planning of management activities and development activities. This deficiency of integration can result in distortions in the planning of projects causing delays in delivery of the software.

Therefore, the objective was to create an environment management software that allows an agile project management software in a customized way in development environments that use concepts and characteristics of agile methodologies. To make this possible, we defined a modular dynamic model and covers all aspects related to project management and agile methodologies that can be instantiated by generating concrete models for each specific project. This dynamics model was given to the great advantage compared to other studies evaluated.

Based on the proposed model was created a project management environment that exploited its full potential. This environment is supported by a set of software agents that maintain the upgraded concrete model. On the concrete model can be extracted and management reports gantt dynamic that makes them even more flexible so as to give the project manager and have a macro view of the progress of activities.

In order to validate the proposed work, we performed a case study in a foreign company. The developed application was used during the development of a real project as a concrete result and brought greater security to project management. Its use enabled to detect failures quickly giving more time to adjust to the impacts on schedule, before being made only through individual conversations or meetings to follow up directly with the team.

Keywords: Agile methodologies. Project management. Software agents. Software Engineering

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACL	*Agent Communication Language*
ASD	*Adaptive Software Development*
AUML	*Agent Unified Modeling Language*
BDI	*Belief-Desire-Intention*
DSDM	*Dynamic System Development Method*
ERP	*Enterprise Resource Planning*
FDD	*Feature Driven Development*
IDE	*Integrated Development Environment*
JAD	*Joint Application Development*
KIF	*Knowledge Interchange Format*
KQML	*Knowledge Query and Manipulation Language*
MaSE	*Multiagent Systems Engineering Methodology*
MVC	*Model View Controller*
ORM	*Object-Relational Mapping*
PBS	*Product Breakdown Structure*
PMBOK	*Project Management Body of Knowledge*
PMI	*Project Management Institute*
RAD	*Rapid Application Development*
RUP	*Rational Unified Process*
SOAP	*Simple Object Access Protocol*
SPIM	*Software Planning Integrated Model*
SPIT	*Software Planning Integrated Tool*
SPPA	*Software Project Planning Associate*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
WBS	*Work Breakdown Structure*
XP	*Extreme Programming*

SUMÁRIO

1. INTRODUÇÃO

Neste capítulo será feita a introdução ao trabalho de dissertação com uma breve contextualização do problema dentro da perspectiva da área de engenharia de software. Além disso, será apresentada a motivação, a questão da pesquisa e os objetivos gerais e específicos para o desenvolvimento da dissertação, bem como a metodologia e a organização do presente trabalho.

1.1. Contextualização

A tecnologia da informação é apontada como uma ferramenta extremamente poderosa que impulsionou um processo de transformação nas organizações nos últimos anos. O desenvolvimento de novos softwares passou a integrar de forma cada vez mais marcante a agenda destas organizações. Sendo assim, a capacidade de operação, a habilidade de oferecer produtos e serviços, a agilidade e a flexibilidade, tão necessárias à sobrevivência e ao crescimento das organizações, estão sendo fortemente influenciadas pela tecnologia da informação e por isso começaram a ser conduzidas e gerenciadas sob a forma de projetos. (KERZNER, 2002).

A gerência de projetos de software é a aplicação de conhecimento, habilidades e técnicas no desenvolvimento de software com o objetivo de atender as necessidades do projeto. É um trabalho que visa a criação de um produto que envolve um certo grau de incerteza na sua realização. Normalmente, é caracterizado por uma sequência de atividades, sendo executadas por pessoas dentro de limitações de tempo, recursos e custos (MARTINS, 2005). Sendo assim, a gerência de projetos de software envolve, dentre outros fatores, o planejamento e o acompanhamento das pessoas envolvidas no projeto, do produto que está sendo desenvolvido e principalmente a metodologia que está sendo seguida para evoluir o software de um conceito preliminar para uma implementação concreta e operacional (PRESSMAN, 2001).

De acordo com Thomsett (2002), os projetos de desenvolvimento de software têm basicamente duas vertentes: uma técnica e outra gerencial. Por um determinado período foi dada muita atenção na ênfase técnica com o aprimoramento dos modelos de desenvolvimento de software, como por exemplo, o modelo em cascata e o modelo em espiral. Estes modelos levam em consideração os aspectos técnicos e as necessidades dos usuários e são integrantes dos chamados métodos clássicos de desenvolvimento de software. A ênfase gerencial teve o enfoque estruturado por meio de processos, que deposita importância fundamental no planejamento e no controle, assim como no rígido gerenciamento de mudanças denominado

gerenciamento clássico de projetos.

No entanto, as pressões do mercado ao desempenho insatisfatório dos projetos de desenvolvimento de software conduzidos com o uso de métodos clássicos e gerenciados com princípios de gerenciamento clássico de projetos, criaram nos últimos anos uma nova abordagem para este tema. Esta reação ocorreu, em princípio, no âmbito técnico para a estruturação de novos modelos que não só se diferenciavam dos modelos tradicionais, como também contestavam e propunham alternativas aos métodos convencionais. São os chamados métodos ágeis de desenvolvimento de software (COHEN; GRAHAM, 2002).

Os métodos ágeis têm por foco o atendimento das expectativas e das necessidades do cliente, a entrega rápida, o reconhecimento da capacidade dos indivíduos e, principalmente, a adaptação a ambientes de negócio bastante dinâmicos. Devido a isso, eles têm conseguido maior espaço principalmente porque houve uma mudança no perfil dos projetos de softwares atuais. As abordagens tradicionais são pouco propícias às mudanças para atender as necessidades do projeto. No entanto, a realidade atual é composta por alterações a todo o momento com requisitos em constante evolução e por isso surge a necessidade de metodologias que se adaptem a esse paradigma (BECK; ANDRES, 2004).

1.2. Motivação

Para acompanhar o andamento do projeto é preciso conseguir medir precisamente o progresso e comparar com o estimado. Não é possível controlar o que não se pode medir e devido a isso existe uma estreita relação entre gerência de projetos e medição. As medidas dão visibilidade ao estado do projeto, permitindo tanto saber para onde ir no início do projeto quanto verificar se o rumo está correto, tomando ações corretivas quando necessário (VAZQUEZ et al., 2005).

Por muitos anos, o gerenciamento de projetos se desenvolveu sobre uma sólida plataforma. As organizações que adotaram o gerenciamento clássico de projetos, propuseram e implementaram melhorias e adaptações, criando seus próprios modelos visto que ao se aproximar do limiar de sua abrangência, as restrições se tornam mais visíveis e o seu desempenho, menos efetivo. Continuar o desenvolvimento desta plataforma clássica em alguns momentos é mais difícil do que estruturar uma nova ideia. (CHIN, 2004)

Diante disso os métodos ágeis ganharam força, porém as ferramentas para gerência de projetos de software existentes ainda focam no trabalho do indivíduo, delegando a este não somente a tarefa de acompanhamento do projeto mas, em muitas vezes, a alimentação de indicadores que são utilizados diretamente no ajuste das fases do projeto durante a sua

execução, mediante mudanças como atrasos no desenvolvimento, mudanças no quadro de pessoal, dentre outros.

A principal motivação para o desenvolvimento do presente trabalho refere-se a tendência atual de utilização de metodologias ágeis no processo de desenvolvimento de software e a existência de poucas iniciativas em fazer com que estas metodologias sejam mais aderentes ao processo de gerência de projetos de software.

1.3. Questão da pesquisa

A questão central deste trabalho é: Como permitir uma gerência de projetos de software de forma personalizada e de acordo com as necessidades de um projeto específico a partir da combinação de conceitos e características de metodologias ágeis?

1.4. Objetivos

A proposta do trabalho é definir um modelo abstrato para gerência de processos de software usando metodologias ágeis. O foco do trabalho será o estudo de metodologias ágeis de desenvolvimento software e modelos de gerência de projetos existentes. Em consequência deste estudo, será definido um metamodelo que conterá características de gerência de processos de desenvolvimento e gerência de projetos de software que utilizem metodologias ágeis.

Este metamodelo proposto poderá ser instanciado no processo de desenvolvimento de cada projeto e uma vez definida esta configuração irá gerar um modelo de dados responsável por armazenar as informações de indicadores importantes à gerência do respectivo projeto. Uma série de agentes de software ficarão monitorando o ambiente de desenvolvimento e populando este modelo.

Sobre este modelo de dados populado será desenvolvida uma aplicação para analisar os impactos no projeto e fornecer informações gerenciais de acompanhamento para a tomada de decisões. Isso permitirá saber se o projeto está em dia, se a carga de trabalho está adequada e conforme o planejamento original, se a quantidade de recursos humanos disponíveis está sendo suficiente para desenvolver o projeto, se vai ser necessário diminuir o escopo, dentre outros, tornando assim a gerência do projeto mais dinâmica e abrangente.

Sendo assim, os objetivos específicos deste trabalho são:

- Criar um metamodelo para gerência de processos de software usando metodologias ágeis;
- Criar um ambiente com agentes de software pró-ativos que populem o metamodelo proposto;

- Criar uma aplicação que possibilite uma gerência de projetos de software mais efetiva;
- Avaliar o ambiente com base em cenários pré-estabelecidos e um estudo de caso.

1.5. Metodologia

A metodologia deste trabalho inicia pela identificação do problema a ser resolvido para que, a partir disto, possa se realizar uma revisão bibliográfica sobre métodos ágeis de desenvolvimento de software e gerência de projetos. Foi realizado um estudo sobre as principais metodologias ágeis de desenvolvimento de software e modelos de gerência de projetos existentes a fim de efetuar um levantamento de suas principais características com o objetivo de elaborar um metamodelo mais aderente aos conceitos definidos por eles.

Após a revisão bibliográfica foi feita uma análise dos principais trabalhos relacionados que serviram de base para verificar até onde o assunto já havia sido explorado, bem como verificar o grau de abrangência de algumas soluções já propostas. Em seguida foi feito um estudo dos principais metamodelos de desenvolvimento em metodologias ágeis existentes e ainda dos metamodelos de gerência de projetos mais adotados atualmente em projetos de software.

Baseado na revisão bibliográfica, nos trabalhos relacionados e nos metamodelos estudados foi elaborado um modelo abstrato de gerência de projetos de software para metodologias ágeis que leva em consideração a união das principais características das metodologias e métodos de gerência de projetos avaliados. A partir deste modelo abstrato proposto, foi criado uma aplicação que implementa esta estrutura em uma arquitetura pró-ativa com o objetivo de suportar a gerência de projetos conforme os objetivos do trabalho.

Uma vez criado o ambiente, sua avaliação será por meio de seu uso em cenários pré-estabelecidos, bem como a partir de um estudo de caso. Após a criação dos cenários, a aplicação poderá ser avaliada por meio de suas respostas à situações previamente identificadas no modelo abstrato proposto. Já o estudo de caso permitirá avaliar a resposta da aplicação sob circunstâncias reais da gerência de projetos de software.

1.6. Organização do trabalho

O presente trabalho está organizado em sete capítulos dispostos da seguinte forma:

- O primeiro capítulo já apresentado introduziu o trabalho de dissertação contextualizando o domínio do problema e apresentando a principal motivação que levou ao desenvolvimento do trabalho. Além disso, definiu a questão da pesquisa, os principais objetivos e a metodologia utilizada;

- O segundo capítulo trata do referencial teórico dos conceitos envolvidos no trabalho, como metodologias ágeis, gerência de projetos de software e agentes de software;
- O terceiro capítulo aborda os principais trabalhos relacionados ao tema da pesquisa. Para tal, serão apresentados estudos envolvendo metodologias ágeis e trabalhos referentes a sistemas baseados em métricas para a gerência de projetos;
- O quarto capítulo apresenta o trabalho proposto detalhando alguns metamodelos de metodologias ágeis estudados realizando uma comparação entre eles e um metamodelo de gerência de projetos com suas principais características. Em seguida, é definido o metamodelo integrado para gerência de projetos em metodologias ágeis e dada uma visão geral da arquitetura do ambiente pró-ativo que está se propondo a desenvolver. Além disso, são detalhados os agentes de software propostos, a arquitetura que os compõem e a aplicação desenvolvida de forma a cumprir os objetivos do presente trabalho;
- O quinto capítulo realiza uma sistematização dos trabalhos relacionados frente ao trabalho proposto conforme critérios pré-definidos;
- O sexto capítulo apresenta um estudo de caso que avalia o ambiente proposto frente a um cenário real de gerência de projetos utilizando metodologias ágeis;
- O sétimo capítulo apresenta as considerações finais bem como indicações para trabalhos futuros.

2. REFERENCIAL TEÓRICO

Neste capítulo serão abordados os principais conceitos que formam a base teórica do presente trabalho. Serão apresentadas as principais metodologias ágeis de desenvolvimento de software tais como XP, *Scrum,* FDD dentre outros. Além disso, será feito um estudo sobre o tema gerência de projetos com ênfase em metodologias ágeis. E por fim, como a arquitetura do ambiente estará fortemente dependente do trabalho de agentes de software, este assunto também será abordado neste capítulo.

2.1. Metodologias Ágeis

2.1.1. Introdução

No desenvolvimento de software com métodos tradicionais os processos eram baseados em um conjunto de atividades predefinidas onde muitas vezes o trabalho iniciava com o levantamento completo de um conjunto de requisitos, seguido por um projeto de alto nível, de uma implementação, de uma validação e, por fim, de uma manutenção (SOMMERVILLE, 2003).

A partir da década de 90, começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil onde os processos adotados tentam se adaptar às mudanças, apoiando a equipe de desenvolvimento no seu trabalho. Estes novos métodos surgiram como uma reação aos métodos tradicionais de desenvolvimento de sistemas, ganhando com o passar dos anos um número cada vez maior de adeptos (BECK; ANDRES, 2004).

A partir de fevereiro de 2001, desenvolvedores de software de várias partes do mundo, insatisfeitos com as técnicas e métodos de desenvolvimento de sistemas usados até o momento, resolveram criar uma aliança chamada de *Agile Software Development Alliance*, mais conhecida como *Agile Alliance*. Estes profissionais das diversas áreas de formação, com pontos de vista diferentes sobre os modelos e métodos de desenvolvimento de software em comum, criaram um manifesto para encorajar os melhores meios de desenvolver um software (AMBLER, 2004).

Este documento, definido como o Manifesto Ágil ou *Agile Manifesto* para as novas metodologias de desenvolvimento de software, foi criado por vários desenvolvedores, tendo como idealizadores Kent Beck, Ken Schwaber, Martin Fowler e Jim Highsmith. Este manifesto abrange um conjunto de princípios que definem critérios para o processo de desenvolvimento de software ágil. Segundo Beck e Andres (2004), os doze princípios, aos

quais os métodos ágeis devem se adequar são:

- Priorizar a satisfação do cliente através de entregas contínuas e frequentes;
- Receber bem as mudanças de requisitos, mesmo em uma fase avançada do projeto;
- Efetuar entregas com frequência, sempre na menor escala de tempo;
- Integrar diariamente o trabalho das equipes de negócio e de desenvolvimento;
- Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessários;
- Conversar face a face é a maneira mais eficiente de fazer a informação circular;
- Ter o sistema funcionando é a melhor medida de progresso;
- Processos ágeis promovem o desenvolvimento sustentável;
- Atenção contínua à excelência técnica aumentam a agilidade;
- Simplicidade é essencial;
- As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
- Em intervalos regulares a equipe deve refletir sobre como se tornar mais eficaz.

As metodologias ágeis têm conseguido maior espaço devido a mudanças no perfil dos projetos de software. Com base nos doze princípios descritos acima, cada um dos métodos ágeis apresenta um conjunto de atividades a serem adotadas durante o processo de desenvolvimento do sistema. A seguir serão brevemente descritas algumas metodologias ágeis existentes e seus principais objetivos.

2.1.2. Extreme Programming

A *Extreme Programming* (XP) é uma metodologia ágil para equipes que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente. XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de software para os seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback* e coragem (BECK; ANDRES, 2004).

A finalidade da comunicação é manter o melhor relacionamento possível entre clientes e desenvolvedores, preferindo conversas pessoais. A comunicação entre os desenvolvedores e o gerente do projeto também é encorajada.

A simplicidade visa permitir a criação de código simples que não deve possuir funções desnecessárias. Outra ideia importante da simplicidade é procurar implementar apenas requisitos atuais, evitando-se adicionar funcionalidades que podem ser importantes no futuro pois elas podem nem vir a acontecer.

O *feedback* significa que o programador terá informações constantes do código e do

cliente. A informação do código é dada pelos testes que indicam os erros tanto individuais quanto do software integrado. Em relação ao cliente, o *feedback* significa que ele terá frequentemente uma parte do software totalmente funcional para avaliar. Desta forma, a tendência é que o produto final esteja de acordo com as expectativas reais do cliente.

E, por fim, a coragem dá suporte à simplicidade e assim que a oportunidade de simplificar o software é percebida, a equipe pode aplicar este valor.

Além dos quatro valores, a XP baseia-se em doze práticas, conforme Beck e Andres (2004):

- **Planejamento**: Consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto. XP baseia-se em requisitos atuais para o desenvolvimento de software. Além disso, a XP procura evitar os problemas de relacionamento entre a área de negócios e a área de desenvolvimento. As duas áreas devem cooperar para o sucesso do projeto e cada uma deve focar em partes específicas. Enquanto a área de negócios decide sobre o escopo, os desenvolvedores devem focar no processo de desenvolvimento.
- **Entregas frequentes:** Visa à construção de um software simples e conforme os requisitos vão surgindo, há a atualização do mesmo. Cada versão entregue deve ter o menor tamanho possível, contendo os requisitos de maior valor para o negócio.
- **Metáfora:** São as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.
- **Projeto simples:** O programa desenvolvido usando a metodologia XP deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros. Estes devem ser adicionados assim que eles realmente existirem.
- **Testes:** A XP focaliza a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os testes.
- **Refatoração:** Focaliza o aperfeiçoamento do projeto do software e está presente em todo o desenvolvimento. A refatoração deve ser feita quando um desenvolvedor da dupla percebe que é possível simplificar o módulo atual sem perder nenhuma funcionalidade.
- **Programação em pares:** A implementação do código é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador. A grande vantagem da programação em dupla é a possibilidade dos desenvolvedores estarem

continuamente aprendendo um com o outro.

- **Propriedade coletiva:** O código do projeto pertence a todos os membros da equipe. Isto significa que qualquer pessoa que percebe que pode adicionar valor a um código, mesmo que ele próprio não o tenha desenvolvido, pode fazê-lo, desde que faça a bateria de testes necessária.
- **Integração contínua:** Interagir e construir o sistema de software várias vezes por dia, mantendo os programadores em sintonia, possibilitando processos rápidos. Uma prática que funciona bem é integrar apenas um conjunto de modificações de cada vez.
- **Semana de trabalho de 40 horas:** XP assume que não se deve fazer horas extras constantemente.
- **Cliente presente:** É fundamental a participação do cliente durante todo o desenvolvimento do projeto. O cliente deve estar sempre disponível para sanar todas as dúvidas de requisitos, evitando atrasos e até mesmo construções erradas. Uma ideia interessante é manter o cliente como parte integrante da equipe de desenvolvimento.
- **Código padrão:** Padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores.

Todas estas doze práticas citadas contribuem para a representação dos processos da metodologia XP, conforme mostrado na figura 1:

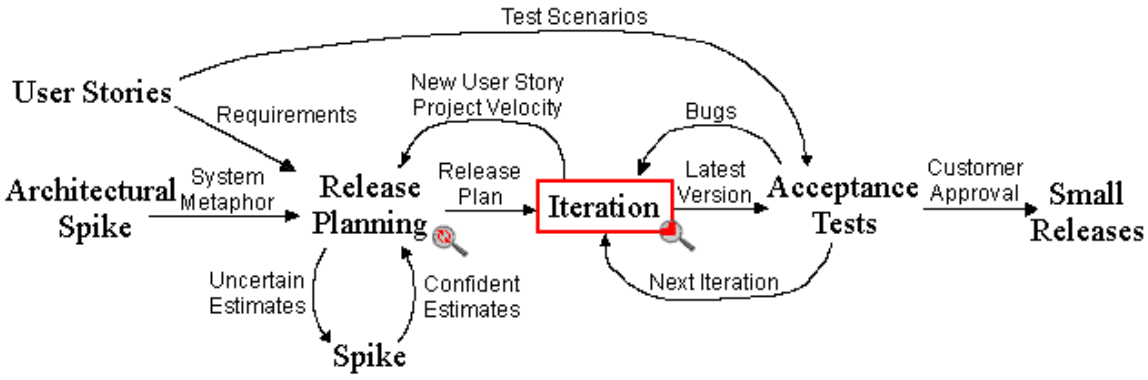


Figura 1: Processo da metodologia XP (XP, 2010)

A cada iteração, novas funcionalidades são desenvolvidas de acordo com uma priorização dada pelo cliente e definida através de critérios de maior valor para o negócio. O planejamento de uma iteração é feito detalhando-se uma *story* em suas tarefas, requisitando programadores para trabalharem nelas e solicitando a estes que estimem tempos. Para todas as

decisões tomadas no desenvolvimento de uma *story* assim como também os assuntos discutidos nas reuniões matinais são armazenados em um histórico (*history*).

De acordo com Beck e Andres (2004), enquanto o planejamento dos *releases* é sincronizado com o negócio do cliente, o planejamento das iterações é uma tarefa muito mais voltada aos programadores envolvidos.

2.1.3. Scrum

Outra metodologia ágil é a *Scrum* (SCHWABER; BEEDLE, 2002). Seu objetivo é fornecer um processo conveniente para projeto e desenvolvimento orientado a objeto. A metodologia *Scrum* apresenta uma abordagem empírica que aplica algumas ideias da teoria de controle de processos industriais para o desenvolvimento de softwares, reintroduzindo as ideias de flexibilidade, adaptabilidade e produtividade. O foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança.

A ideia principal da metodologia *Scrum* é que o desenvolvimento de software envolve muitas variáveis técnicas e do ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo. Isto torna o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças. O resultado do processo deve ser um software que é realmente útil para o cliente.

Para que a metodologia *Scrum* consiga atingir todos os seus objetivos, Schwaber e Beedle (2002) definiram alguns processos básicos, divididos em papéis, cerimônias e artefatos, que constituem a essência do processo *Scrum*, conforme detalhado abaixo:

- ***Sprint:*** Uma *Sprint* é uma iteração de entrega de incremento funcional de software. Essas iterações duram normalmente de 2 a 4 semanas e começam e terminam com uma reunião da equipe.
- ***Scrum Master:*** É o responsável por proteger os membros da equipe de desenvolvimento de qualquer pressão externa que possa afetar a produtividade. Tenta garantir que todas as práticas do *Scrum* sejam utilizadas com perfeição pela equipe e tem um papel de facilitador nas reuniões da *Sprint.*
- ***Product Owner:*** É o responsável por priorizar o *Product Backlog*. Este é um papel importante pois a equipe de desenvolvimento observará o *Product Backlog* priorizado e construirá o *Sprint Backlog*, comprometendo-se a entregar os itens postados. O *Product Owner* garante que durante a *Sprint* não haverá mudanças nos requisitos solicitados, porém, nos intervalos entre *Sprints* ele possui total liberdade

para modificá-los.

- ***Scrum Team:*** É a própria equipe de desenvolvimento. As equipes de desenvolvimento em *Scrum* são incentivadas a serem multidisciplinares, ou seja, todos trabalham cooperativamente para entregar as funcionalidades que a equipe se comprometeu a entregar no começo do *Sprint.* O tamanho típico de uma equipe varia entre 6 e 10 pessoas.
- ***BurnDown:*** Visto que o *Scrum* prega a transparência dentro de uma equipe de desenvolvimento, um quadro onde todas as estórias (requisitos) existentes para um produto estejam expostas é extremamente útil. Este quadro é conhecido como *BurnDown* e trata-se do local onde são registrados os estados das estórias em andamento.
- ***Product Backlog:*** É a lista principal de funcionalidades para o desenvolvimento de um certo produto. O *Product Backlog* é um elemento essencial na reunião de planejamento, pois é com o *Product Backlog* priorizado que o *Product Owner* irá se comunicar com a equipe de desenvolvimento com o objetivo de expressar suas maiores necessidades para aquele *Sprint.*
- ***Sprint Backlog:*** Na reunião de planejamento da *Sprint (Planning)* o *Product Owner* deverá mostrar o *Product Backlog* priorizado e a equipe irá escolher quais tarefas daquela lista serão realizadas durante aquela *Sprint*. As estórias escolhidas serão colocadas na sessão de *Sprint Backlog*, ou seja, lista de funcionalidades a serem implementadas na *Sprint* corrente.
- **Estória:** Cada elemento da lista é conhecido como Estória e deve possuir uma breve descrição das funcionalidades requeridas pelo *Product Owner* para um dado produto. A equipe pode dividir uma estória em tarefas, de preferência pequenas, para que, com a conclusão de todas as tarefas, a própria estória esteja concluída.
- ***Sprint Planning:*** É a reunião principal do *Scrum.* Nela são planejadas as atividades para a *Sprint* corrente. Nesta reunião, o *Product Owner* apresenta o *Product Backlog* priorizado e os membros da equipe decidem quais estórias do *Product Backlog* serão incluídas no *Sprint Backlog,* definindo com isso um objetivo para o *Sprint*.
- ***Sprint Review:*** No final de cada *Sprint*, a equipe de desenvolvimento, o *Scrum Master* e o *Product Owner* se reúnem para verificar o que foi feito durante a *Sprint.* A equipe apresenta as funcionalidades prontas e o *Product Owner* aprova ou desaprova o produto. Essa reunião é aberta para qualquer outra equipe presente

que queira observar a apresentação das estórias completadas durante a *Sprint*.

- ***Sprint Retrospective:*** É realizada logo após o *Sprint Review* e tem como objetivo mostrar para todos da equipe o que houve de melhor e o que pode melhorar na *Sprint* analisada. Essa reunião é fechada para apenas a equipe e o *Scrum Master*. Essa reunião é considerada muito importante para o crescimento da equipe, pois é nela onde serão mostrados os erros e as lições aprendidas durante a *Sprint*.
- ***Daily Scrum:*** É uma reunião realizada diariamente entre os membros da equipe com o objetivo de analisar o que cada membro da equipe fez no dia anterior, o que pretende fazer durante o dia presente e quais os impedimentos ou obstáculos que tem para cumprir com as tarefas. Normalmente é feita de pé e é de curta duração.

A figura 2 representa todos os itens que foram explicados acima definindo o processo da metodologia *Scrum:*

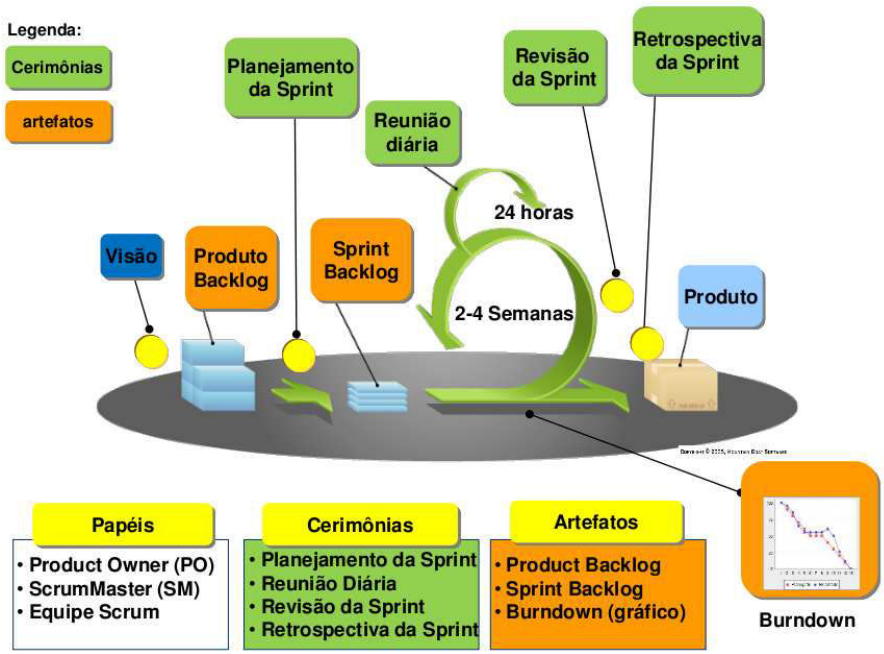


Figura 2: Processo da metodologia *Scrum* (SCHWABER; BEEDLE, 2002)

A metodologia *Scrum* divide o desenvolvimento em iterações (*Sprints*) de aproximadamente trinta dias. As equipes são formadas por projetistas, programadores, engenheiros e gerentes de qualidade que trabalham em cima de funcionalidades definidas no início de cada *Sprint* e são responsáveis pelo seu desenvolvimento.

Na metodologia *Scrum* existem reuniões de acompanhamento diárias. Nessas reuniões, que são preferencialmente de curta duração e acontecem de pé, são discutidos pontos como o que foi feito desde a última reunião e o que precisa ser feito até a próxima. As dificuldades encontradas e os fatores de impedimento são identificados e resolvidos.

O ciclo de vida da *Scrum* é baseado em três fases principais, divididas em sub-fases:

- **Pré-planejamento (*Pre-game phase*):** Os requisitos são descritos em um documento chamado *Backlog*. Posteriormente eles são priorizados e são feitas estimativas de esforço para o desenvolvimento de cada requisito. O planejamento inclui também, entre outras atividades, a definição da equipe de desenvolvimento, as ferramentas a serem usadas, os possíveis riscos do projeto e as necessidades de treinamento. Finalmente é proposta uma arquitetura de desenvolvimento. Eventuais alterações nos requisitos descritos no *Backlog* são identificadas, assim como seus possíveis riscos.
- **Desenvolvimento (*game phase*):** As muitas variáveis técnicas e de ambiente identificadas previamente são observadas e controladas durante o desenvolvimento. Ao invés de considerar essas variáveis apenas no início do projeto, como no caso das metodologias tradicionais, na metodologia *Scrum* o controle é feito continuamente, o que aumenta a flexibilidade para acompanhar as mudanças. Nesta fase, o software é desenvolvido em ciclos (*Sprints*) em que novas funcionalidades são adicionadas. Cada um desses ciclos é desenvolvido de forma tradicional, ou seja, primeiramente faz-se a análise, em seguida o projeto, implementação e testes. Cada um desses ciclos é planejado para durar de uma semana a um mês.
- **Pós-planejamento (*post-game phase*):** Após a fase de desenvolvimento são feitas reuniões para analisar o progresso do projeto e demonstrar o software atual para os clientes. Nesta fase são feitas as etapas de integração, testes finais e documentação.

2.1.4. Feature Driven Development

A metodologia *Feature Driven Development* (FDD) foi desenvolvida por Peter Coad e Jeff de Luca no final da década de 90. Oferece um conjunto coeso de princípios e práticas tanto para a gestão de projetos quanto para a Engenharia de Software, mas convive bem com abordagens mais especialistas, como a *Scrum*.

Apesar de ter algumas divergências pontuais com a XP, várias práticas propostas por ela também são utilizadas por equipes usando FDD, como os testes unitários, refatoração, programação em pares, integração contínua, entre outras. A FDD também propõe práticas como excesso de formalismo e posse individual de código, que podem contrastar com algumas das práticas fundamentais da XP. A experiência da equipe e dos gerentes deve julgar quais práticas são mais apropriadas.

A FDD é classicamente descrita por duas fases e composta por cinco processos bem definidos e integrados: a fase de concepção e planejamento, composta pelos processos de

“desenvolver um modelo abrangente”, “construir uma lista de funcionalidades” e “planejar por funcionalidade” e a fase iterativa de construção, composta pelos processos de “detalhar por funcionalidade” e “construir por funcionalidade” (PALMER; FELSING, 2002). Segue abaixo o detalhamento de cada um dos cinco processos:

- **Desenvolver um modelo abrangente:** Pode envolver desenvolvimento de requisitos, análise orientada por objetos, modelagem lógica de dados e outras técnicas para entendimento do domínio de negócio em questão. O resultado é um modelo de objetos de alto nível, que guiará a equipe durante os ciclos de construção.
- **Construir uma lista de funcionalidades:** Decomposição funcional do modelo do domínio, em três camadas típicas: áreas de negócio, atividades de negócio e passos automatizados da atividade. O resultado é uma hierarquia de funcionalidades que representa o produto a ser construído (também chamado de *product backlog* ou lista de espera do produto).
- **Planejar por funcionalidade:** Abrange a estimativa de complexidade e dependência das funcionalidades, também levando em consideração a prioridade e valor para o negócio/cliente. O resultado é um plano de desenvolvimento, com os pacotes de trabalho na sequência apropriada para a construção.
- **Detalhar por funcionalidade:** Dentro de uma iteração de construção, a equipe detalha os requisitos e outros artefatos para a codificação de cada funcionalidade, incluindo os testes. O projeto para as funcionalidades é inspecionado. O resultado é o modelo de domínio mais detalhado e os esqueletos de código prontos para serem preenchidos.
- **Construir por funcionalidade:** Cada esqueleto de código é preenchido, testado e inspecionado. O resultado é um incremento do produto integrado ao repositório principal de código, com qualidade e potencial para ser usado pelo usuário.

A figura 3 permite uma visão geral dos processos da metodologia FDD:

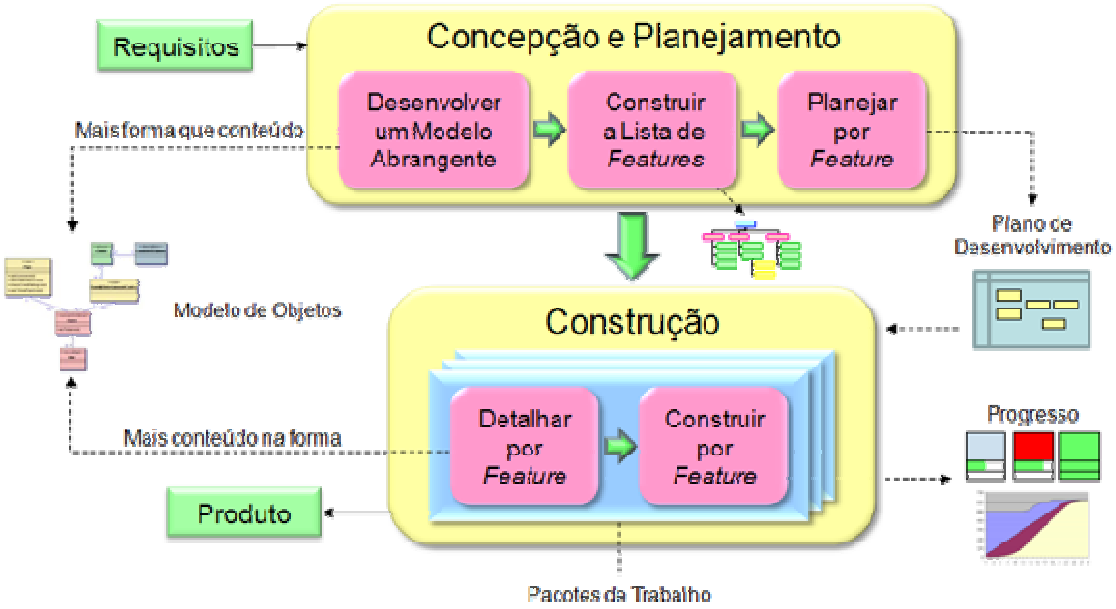


Figura 3: Processo da metodologia FDD (FDD, 2010)

2.1.5. Dynamic System Development Method

A metodologia *Dynamic System Development Method* (DSDM) foi desenvolvida inicialmente por um consórcio de empresas britânicas em 1994. Essa metodologia pode ser considerada o primeiro método ágil desenvolvido e baseou-se nas ideias do *Rapid Application Development* (RAD) proposto por Martin (1991) e no desenvolvimento iterativo e incremental proposto por Stapleton (1997).

Normalmente a DSDM é aplicado em projetos de sistemas caracterizados pelos cronogramas e custos limitados. Aponta falhas de informação mais comuns destes projetos, incluindo custos excedentes, perda de prazos, falta de envolvimento de usuários e acompanhamento da alta gerência.

A DSDM consiste de três fases sequenciais, nomeadas de pré-projeto, ciclo de vida e pós-projeto. O ciclo de vida é a fase mais elaborada pois consiste de quatro estágios que formam o passo-a-passo das iterações aplicadas ao desenvolvimento do sistema. Veja a seguir as principais atividades de cada fase:

- **O pré-projeto:** São identificados os projetos candidatos e são definidos orçamento e assinatura do contrato.
- **O ciclo de vida:** Fase mais complexa composta por cinco estágios. Os dois primeiros estágios, análise de viabilidade e negócios são fases sequenciais que se completam entre si. Após a conclusão destas fases, o sistema é desenvolvido incrementalmente segundo as iterações do modelo funcional, desenho e

construção, até à implementação:

- **Pós-projeto:** Garante a eficiência e eficácia do projeto através de manutenções, melhorias e ajustes de acordo com os princípios da DSDM a fim de refinar ainda mais o passo concluído.

A metodologia DSDM é formada por nove princípios, conforme segue abaixo:

- **Envolvimento**: O envolvimento do usuário é o ponto principal para eficiência e eficácia do projeto. Usuários e desenvolvedores dividem o mesmo espaço e com isso as decisões podem ser tomadas com maior precisão.
- **Autonomia**: A equipe deve estar empenhado em tomar decisões que sejam importantes ao progresso do projeto sem que necessitem de aprovação dos superiores.
- **Entregas**: O foco em entregas frequentes assumindo que entregar algo bom logo é melhor que entregar algo perfeito somente no fim. Iniciando a entrega do produto desde os primeiros estágios do projeto, o mesmo já pode ser testado e revisado. A evidência do teste e a revisão da documentação podem ser utilizadas na próxima iteração ou fase.
- **Eficácia:** O critério principal para considerar um sistema entregue é que ele possa auxiliar nas necessidades atuais do cliente.
- ***Feedback:*** O desenvolvimento é iterativo e controlado por *feedbacks* de usuários, a fim de tornar a solução eficaz ao negócio.
- **Reversibilidade:** Todas as alterações feitas no desenvolvimento são reversíveis.
- **Previsibilidade:** O escopo e requisitos de alto nível devem ser definidos antes que o projeto se inicie.
- **Ausência de testes no escopo:** Testes são tratados fora do ciclo de vida do projeto.
- **Comunicação:** É necessária excelente comunicação e cooperação de todos os envolvidos para obter maior eficácia e eficiência no projeto.

A metodologia DSDM tem como meta entregar softwares no tempo e custo estimados através do controle e ajuste de requisitos ao longo do desenvolvimento.

2.1.6. Família *Crystal*

Foi proposta por Alistair Cockburn em 1998. Trata-se de uma família de métodos que são necessários em diferentes abordagens para equipes de tamanhos diferentes. Apesar disso, todos os métodos dessa família compartilham propriedades como: entrega frequente, reflexão e comunicação (COCKBURN, 2004).

O ciclo de vida desta família de metodologia é baseado nas seguintes práticas:

- ***Staging:*** Planejamento do próximo incremento do sistema. A equipe seleciona os requisitos que serão implementados na iteração e o prazo para sua entrega;
- **Edição e revisão:** Construção, demonstração e revisão dos objetivos do incremento;
- **Monitoramento:** O processo é monitorado com relação ao progresso e estabilidade da equipe. É medido em marcos e em estágios de estabilidade;
- **Paralelismo e fluxo:** As diferentes equipes podem operar com máximo paralelismo. Isto é permitido através do monitoramento da estabilidade e da sincronização entre as equipes;
- **Inspeções de usuários:** São sugeridas duas a três inspeções feitas por usuários a cada incremento;
- ***Workshops* refletivos:** São reuniões que ocorrem antes e depois de cada iteração com objetivo de analisar o progresso do projeto.
- **Local *matters:*** São os procedimentos a serem aplicados, que variam de acordo com o tipo de projeto.
- ***Work Products:*** Sequência de lançamento, modelos de objetos comuns, manual do usuário, casos de teste e migração de código.
- ***Standards:*** padrões de notação, convenções de produto, formatação e qualidade usadas no projeto.
- ***Tools:*** Ferramentas mínimas utilizadas. Compiladores, gerenciadores de versão e configuração, ferramentas de versão, programação, teste, comunicação, monitoramento de projeto, desenho e medição de performance.

Existem algumas características comuns à família *Crystal*, tais como o desenvolvimento incremental com ciclos de no máximo quatro meses, ênfase maior na comunicação e cooperação das pessoas, não limitação de quaisquer práticas de desenvolvimento, ferramentas ou produtos de trabalho e incorporação de objetivos para reduzir produtos de trabalho intermediários e desenvolvê-los como projetos evoluídos (COCKBURN, 2004).

2.1.7. Adaptive Software Development

A metodologia *Adaptive Software Development* (ASD) foi proposta por Jim Highsmith, em 2000 como uma forma alternativa de se enxergar o desenvolvimento de software nas organizações (HIGHSMITH, 2002). Essa metodologia tenta explorar a natureza

adaptativa e a incerteza no desenvolvimento de software baseado nas ideias dos sistemas adaptativos complexos, relacionados à Teoria do Caos (WALDROP, 1992).

A metodologia ASD foi projetada para lidar com ambientes complexos e repletos de incertezas. Ela estimula a aprendizagem durante o processo de desenvolvimento e a adaptação constante às novas realidades do negócio e do projeto. Além disso, encoraja o desenvolvimento iterativo e incremental, com a liberação constante de novas versões. Oferece estrutura e orientação suficiente para evitar que os projetos se tornem caóticos, sem trazer uma rigidez indesejada que venha a suprimir a criatividade (HIGHSMITH, 2002).

Neste método, o papel do gerente de projetos é favorecer a colaboração entre a equipe de desenvolvimento e o cliente. Por meio de iterações curtas, a equipe cria o conhecimento cometendo pequenas *falhas,* causadas por falsas premissas e corrigindo-as aos poucos, criando uma experiência mais rica e ampla. É indicada para equipes pequenas, mas pode ser adaptado para equipes maiores (HIGHSMITH, 2002).

2.1.8. Evolução das metodologias ágeis

Nos últimos anos, o movimento de desenvolvimento de software utilizando metodologias ágeis vem ganhando espaço dentro da área de engenharia de software. As metodologias ágeis variam em termos de práticas e ênfases e compartilham algumas características como o desenvolvimento iterativo, a comunicação e a redução de produtos intermediários que fazem com que os requisitos do cliente sejam atendidos mais rapidamente.

Para visualizar de uma maneira mais clara as várias metodologias ágeis existentes atualmente, foi proposto por Abrahamsson et al. (2003) um mapa com algumas das inter-relações existentes entre as metodologias ágeis na evolução dos tempos conforme mostrado na figura 4.

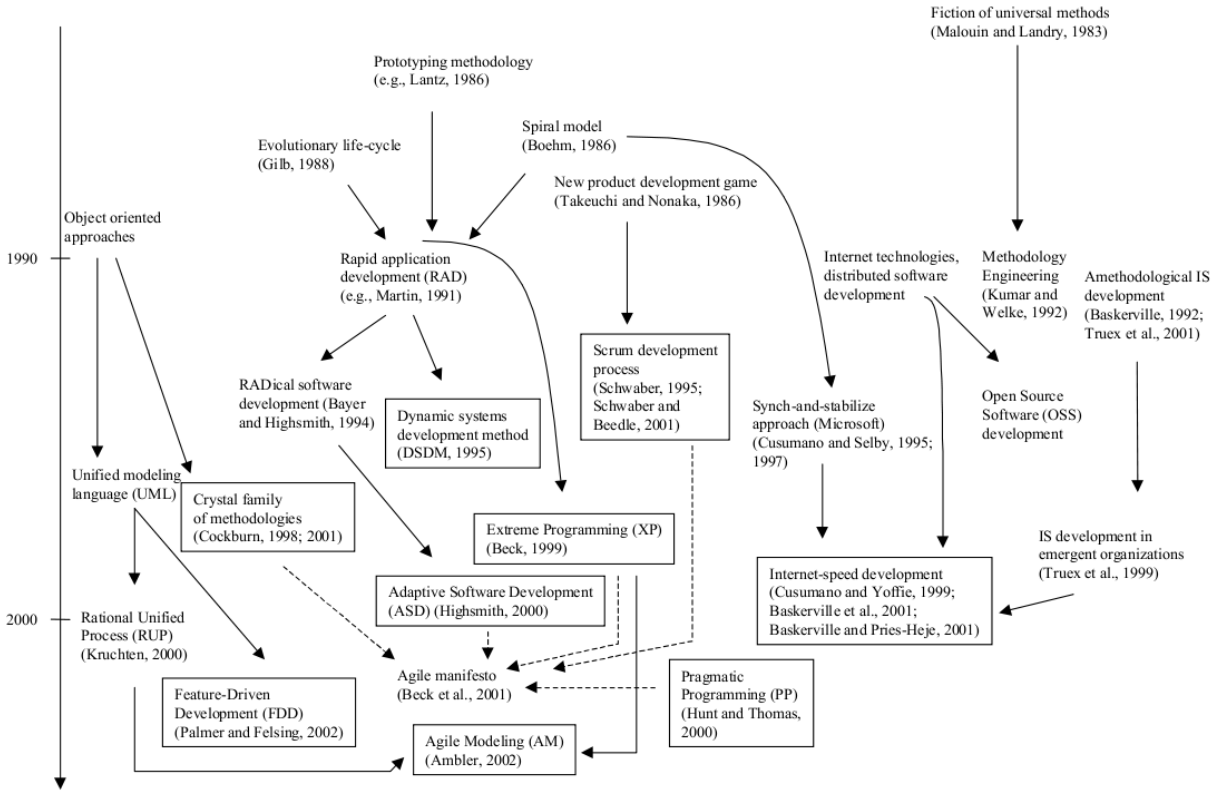


Figura 4: Mapa da evolução dos métodos ágeis (ABRAHAMSSON et al., 2003)

Neste mapa estão propositadamente relacionados métodos que se estendem além do escopo das metodologias abordadas neste trabalho pelo fato de contribuírem filosoficamente para a criação de alguns métodos ágeis. Além disso também são apresentados através da linha tracejada, quais os métodos contribuíram para a publicação do manifesto ágil[1].

2.2. Gerência de projetos

2.2.1. Introdução

Muito se fala em processo de desenvolvimento de software e com certeza seu papel é fundamental em um projeto de desenvolvimento. Porém apenas a adoção de um bom processo de software não garantirá o sucesso do projeto, acima de tudo se faz necessário planejar e gerenciar todas as atividades envolvidas durante o ciclo de vida do projeto. Grande parte das falhas nos projetos de software é resultado da não utilização de uma metodologia para planejamento e gerência do projeto (PRESSMAN, 2001).

2.2.2. Gerência de projetos de software

Projeto é um empreendimento temporário com o objetivo de criar um produto ou serviço único (VIEIRA, 2003). É um trabalho que visa à criação de um produto ou à execução de um serviço específico, temporário, não repetitivo e que envolve um certo grau de incerteza

1 http://www.agilemanifesto.org

na sua realização (MARTINS, 2005). Normalmente, é caracterizado por uma sequência de atividades, sendo executada por pessoas dentro de limitações de tempo, recursos e custos.

Assim sendo, a gerência de projetos de software envolve, dentre outros, o planejamento e o acompanhamento das pessoas envolvidas no projeto, do produto sendo desenvolvido e do processo seguido para evoluir o software de um conceito preliminar para uma implementação concreta e operacional (PRESSMAN, 2001).

Cada vez mais os projetos de software estão ganhando importância e fazendo parte das atividades estratégicas das organizações (PRADO, 1998). No planejamento do projeto são estabelecidas as metas ou objetivos a serem atingidos, as atividades a serem realizadas e a sua sequência com base nos recursos necessários e disponíveis. Por se tratar de um processo que tem um determinado tempo para ser concluído, todo projeto passa por algumas etapas que se costuma chamar de ciclo de vida do projeto. Existem algumas fases que podemos considerar como sendo parte do ciclo de vida genérico de um projeto (VALERIANO, 1998):

- **Fase conceitual:** Inclui todas as atividades que vão desde a ideia inicial do assunto a se pesquisar que engloba a elaboração da proposta até a aprovação da mesma. As decisões tomadas nesta fase, em meio a dúvidas e incertezas, são responsáveis pelos maiores efeitos e consequências do restante da vida do projeto.
- **Fase de planejamento e organização:** Fase em que o projeto é traçado com os mínimos detalhes necessários para sua execução e controle. Nesta etapa são determinadas as responsabilidades e as condições de controle, quanto a prazos, custos e desempenho.
- **Fase de implementação:** Esta fase consiste na realização e controle das tarefas, ou seja, executar o que foi planejado e ajustar a execução quando necessário para atingir o objetivo, dentro das condições pré-determinadas.
- **Fase de encerramento:** Os resultados do projeto são passados ao cliente que efetua a sua aprovação. Nesta fase, além da aprovação do cliente que é o objetivo principal, a equipe realiza uma avaliação interna de todo o projeto, efetua as prestações de contas e entrega sua documentação final.

A gerência de projetos de software tem algumas particularidades em relação aos projetos genéricos. Para realizar um projeto de software com sucesso, deve-se ter bem claro o seu escopo, os riscos que poderão ocorrer, as tarefas a serem executadas, os pontos de controle a serem acompanhados, os recursos humanos, financeiros e materiais (hardware, software) necessários, o esforço empregado e o cronograma a ser seguido. De posse dessas informações é possível ter uma estimativa de custo confiável, uma correta divisão das tarefas

e programação do projeto (SCHNEIDER, 2010).

No entanto, é muito importante observar a estreita relação entre gerência de projetos e medição. Para acompanhar o andamento do projeto, é preciso medir o progresso e comparar com o estimado. Mesmo no planejamento, sobretudo quando se pretende utilizar dados de projetos anteriores, dados de métricas são muito importantes. Não é possível controlar o que não se pode medir e, principalmente, só é possível chegar a boas estimativas com base em dados históricos se estes dados forem coletados criteriosamente. Assim, as medidas dão visibilidade ao estado do projeto, permitindo tanto saber para onde ir no início do projeto quanto verificar se o rumo está correto, tomando ações corretivas quando necessário (VAZQUEZ et al., 2005).

A falta de métricas de projeto, prejudica de forma geral o seu acompanhamento, uma vez que pode haver um problema, mas ele não está sendo percebido por aqueles que podem direcionar esforços para a sua solução. Assim, métricas têm um importante papel na rápida identificação e correção de problemas ao longo do desenvolvimento do projeto. Com a sua utilização, fica muito mais fácil justificar e defender as decisões tomadas, afinal, o gerente de projeto não decidiu apenas com base em seu sentimento e experiência, mas também fundamentado na avaliação de indicadores que refletem uma tendência de comportamento futuro (VASQUEZ, 2005).

No que tange à gerência de projetos, estabelecer classes de projetos e coletar algumas métricas pode ser bastante importante para apoiar a realização de estimativas. Por exemplo, se uma organização tem indicadores para produtividade (tamanho/esforço) e custo (valor/tamanho) para diversas classes de projetos diferentes, é possível, a partir de uma estimativa de tamanho, chegar a estimativas de esforço e custo (VASQUEZ, 2005).

Estimativas também são importantes num projeto de software. Antes mesmo de serem iniciadas as atividades técnicas de um projeto, o gerente e a equipe de desenvolvimento devem estimar o trabalho a ser realizado, os recursos necessários, a duração e, por fim, o custo do projeto. Apesar das estimativas serem um pouco de arte e um pouco de ciência, essa importante atividade não deve ser conduzida desordenadamente. As estimativas podem ser consideradas a fundação para todas as outras atividades de planejamento de projeto (PRESSMAN, 2001).

Para alcançar boas estimativas de prazo, esforço e custo, existem algumas opções:

- Postergar as estimativas até o mais tarde possível no projeto;
- Usar técnicas de decomposição;
- Usar um ou mais modelos empíricos para estimativas de custo e esforço;

- Basear as estimativas em projetos similares que já tenham sido concluídos.

O objetivo destas abordagens é atingir várias formas para realizar estimativas e usar seus resultados para torná-las mais precisas. Quando se fala em estimativas, está se tratando na realidade de diversos tipos: tamanho, esforço, recursos, tempo e custos. Geralmente, a realização de estimativas começa pelas de tamanho. A partir delas, estima-se o esforço necessário e na sequência alocam-se recursos, elabora-se o cronograma do projeto e finalmente o custo é estimado.

O uso de algumas técnicas auxiliam os gerentes de projetos na divisão do trabalho, e consequentemente na distribuição dos recursos necessários para cada etapa do projeto, de maneira correta e eficaz. Uma destas técnicas utilizada atualmente é o *Project Management Body of Knowledge* – PMBOK, reconhecido internacionalmente pelo seu esforço em definir normas e dar suporte aos profissionais de gerência de projetos. O PMBOK reúne as melhores práticas aplicáveis à maioria dos projetos e sobre as quais há um amplo consenso sobre o seu valor e utilidade.

O *Project Management Institute* – PMI publicou um guia geral de gerência de projetos, o PMBOK Guide. O principal objetivo do PMBOK Guide é identificar um subconjunto dos conhecimentos sobre gerência de projetos que seja reconhecido, genericamente, como sendo uma coleção de boas práticas (PMBOK, 2008).

Visando organizar o conhecimento e as práticas relativas à gerência de projeto, o PMI organizou os processos de gerência de projeto em nove áreas de conhecimento conforme descritas a seguir (PMI, 2010):

- **Gerência de Integração:** Nesta área de conhecimento estão os processos responsáveis por garantir um acompanhamento apropriado dos vários elementos que compõem um projeto. Podemos construir um plano de iteração realizando as atividades de detalhamento dos objetivos e desenvolvendo uma estrutura de produto detalhada (PBS – *Product Breakdown Structure*), definindo critérios de avaliação e marcos de iteração, organizando o esforço entre as disciplinas, desenvolvendo a estrutura de trabalho detalhada (WBS – *Work Breakdown Structure*), aquisição de pessoal, atribuição das atividades e plano consolidado.
- **Gerência de Escopo:** Identifica e controla os processos necessários para garantir que o projeto contenha todo o trabalho requerido para atingir o sucesso. É importante distinguir os conceitos de escopo do produto e escopo do projeto. Este último trata das atividades que devem ser realizadas durante o projeto para se atingir o resultado final desejado, enquanto que o primeiro trata das características

do produto de software final.

- **Gerência do Tempo:** Nesta área de conhecimento se encontram os processos responsáveis por garantir que os prazos determinados para o projeto sejam cumpridos.
- **Gerência de Custo:** Inclui os processos responsáveis por garantir que o projeto termine dentro do orçamento aprovado para o mesmo. O gerenciamento do custo está relacionado ao custo dos recursos utilizados para realizar todas as atividades do projeto. É importante perceber que algumas decisões, principalmente decisões gerenciais, causam alterações no custo do produto final do projeto.
- **Gerência da Qualidade:** Nesta área de conhecimento se encontram os processos necessários para garantir que o projeto atinja de forma correta os seus objetivos finais. A gerência de qualidade é uma disciplina intimamente relacionada com a gerência do projeto. Ambas possuem aspectos em comum tais como satisfação do cliente, liberando um produto em conformidade com o escopo e que atenda as reais necessidades do cliente, prevenção ao invés de inspeção, gerência responsável provendo o projeto dos recursos necessários, fases do projeto compostas por processos e registro de evidências dos pontos de controle.
- **Gerência de Recursos Humanos:** Processos necessários para garantir uma utilização efetiva dos recursos humanos envolvidos no projeto, incluindo todos *stakeholders,* tais como patrocinadores, clientes, parceiros, consultores, colaboradores individuais entre outros. O gerente de projeto deve estar atento para alguns aspectos fundamentais relacionados a gerência de pessoas como liderança, comunicação, negociação, delegação, motivação, treinamento, formação de equipe, avaliação de performance, recrutamento, retenção, relações de trabalho, regulamentações de saúde e segurança no trabalho e outros assuntos relacionados com a administração de recursos humanos.
- **Gerência da Comunicação:** Processos necessários para garantir uma efetiva geração, coleta, disseminação e armazenamento das informações do projeto, de forma apropriada e oportuna. Nesta área de conhecimento são criados os vínculos entre pessoas, ideias e informações, fundamentais para o sucesso do projeto.
- **Gerência de Risco:** Nesta área de conhecimento se concentram as atividades de identificação, análise e resposta aos riscos do projeto. Como regra geral devemos maximizar a probabilidade de ocorrência dos eventos que atuem positivamente, bem como minimizar a probabilidade dos eventos que atuam de forma negativa.

Os riscos podem ser ameaçadores aos objetivos do projeto, como também podem trazer oportunidades de melhoria caso sejam adotados, por exemplo um cronograma agressivo.

- **Gerência de Aquisição:** Trata dos processos necessários para aquisição de bens e serviços externos para a realização do projeto e que são importantes para atingir os seus objetivos.

Gerenciar projetos de software com eficiência constitui-se não apenas um grande desafio dos dias atuais, mas é o fator crítico para o sucesso e para a sobrevivência das empresas. Gerenciar projetos de software com eficiência requer um esforço de conscientização das empresas em adotar metodologias de gerência de projetos e, se possível, manter e suportar uma única metodologia para gerenciamento de projetos.

Neste cenário, o gerente do projeto capacitado é aquele que tem melhores condições de ver as necessidades do mesmo. Ele deve ser um profissional treinado para usar uma metodologia de gerenciamento de projetos e aplicá-la de forma eficiente. Ao gerente devem ser dados autorização formal e apoio visível da alta administração para que ele possa desempenhar bem o seu papel de gestor buscando o sucesso do projeto.

2.2.3. Gerência de projetos em metodologias ágeis

As metodologias tradicionais de gerência de projeto têm foco no planejamento e muita disciplina no processo, partindo do princípio de que a aplicação de controle possibilita alcançar o objetivo planejado dentro de limites pré-estabelecidos de tempo, custo e escopo. Para a maioria dos projetos, porém, o gerenciamento tradicional acaba acrescentando somente custo e complexidade, enquanto fornece um falso senso de segurança, uma vez que o gerente está sempre ocupado exaustivamente planejando, medindo e controlando.

As metodologias ágeis estão se consolidando e atraindo novos adeptos, pois apresentam como uma coleção de boas práticas. Na sua essência, o ciclo de vida ágil é semelhante a qualquer outra abordagem de desenvolvimento: descobrir o que o cliente quer, certificar-se exatamente sobre o que ele quer, calcular o esforço, construir a solução e entregá-la. O que muda na abordagem ágil, porém, é que o ciclo fica reduzido em algumas semanas para cada componente funcional desenvolvido, tornando-se muito mais rápido e sendo executado diversas vezes.

O gerenciamento tradicional de projetos parte do princípio de que o planejamento direciona os resultados e que entregar o planejado é sinônimo de sucesso em um projeto. Já a abordagem ágil considera que os resultados devem direcionar o planejamento e o sucesso

advém de entregar o resultado desejado, não necessariamente o resultado planejado. A grande diferença entre estas duas abordagens está na atitude em relação às mudanças no planejamento: a abordagem ágil considera necessário incentivar a mudança, e não desencorajá-la por meio da aplicação de procedimentos que restringem e diminuem a criatividade. A abordagem ágil esforça-se para reduzir o custo de mudança ao longo do processo de desenvolvimento de software (NESMA, 2005).

No planejamento de um projeto tradicional, o principal processo é o desenvolvimento do plano do projeto que definirá como ele será executado e controlado contendo os planos para cada uma das áreas. Nas metodologias ágeis o plano de projeto é feito e revisado continuamente com um nível de detalhe suficiente para a realidade daquele instante, conforme os *releases* e iterações. Da mesma forma o controle integrado de mudança é natural no dia a dia da equipe, e ocorre através das priorizações feitas pelo cliente durante as reuniões de planejamento e revisão de iteração e *releases.*

Nas metodologias ágeis a gerência do escopo procura seguir uma filosofia totalmente diferente das tradicionais. Ao invés de blindar o escopo contra mudanças, a abordagem ágil aceita a mudança. Uma abordagem colocada por Slinger (2006) é a forma de criar um WBS. Ao invés de conter todo o trabalho para o projeto, nos métodos ágeis um WBS pode ser feito no início de cada *release* pois a iteração das funcionalidades são quebradas em tarefas.

Diante disso, Koch (2004) fez um estudo comparativo das nove áreas de conhecimento do PMBOK em relação a gerência tradicional e a gerência ágil, conforme descrito na tabela 1:

Tabela 1: Comparativo entre gerência tradicional e ágil (KOCH, 2004)

Área do PMBOK	**Gerência tradicional**	**Gerência ágil**
Escopo	Bem definido nas fases iniciais do projeto e formalizado através do WBS.	Escopo é definido em alto nível e os requisitos são priorizados e definidos de forma iterativa.
Tempo	Cronograma detalhado para a realização de todo o projeto.	Cronograma orientado ao produto com entregas incrementais de 2-4 semanas.
Custo	Monitoração das alterações para que não afete o custo planejado.	Maior controle em função da rapidez na incorporação de alterações.
Qualidade	Processos de verificação e validação do plano de testes.	Programação em pares, testes incrementais e refatoração.
Riscos	Análise de riscos durante todo o ciclo de vida do projeto.	Aplica-se o mesmo conceito de gerenciamento tradicional.
Comunicação	Documentado e formal.	Implícita, interpessoal e colaborativa.
Recursos Humanos	Papéis claros e bem definidos.	Confiança nos membros da equipe e

Área do PMBOK	Gerência tradicional	Gerência ágil
		ambiente colaborativo.
Aquisição	Controle por contrato e escopo bem definido e documentado.	Presença do cliente, volatilidade de requisitos e pouca documentação torna o processo um desafio.
Integração	Plano de projeto detalhado e controle total do projeto pelo gerente.	Plano de projeto evolutivo e gerente do projeto atua como facilitador.

Boehm (2006) explica que tanto a abordagem ágil quanto a tradicional possuem aspectos positivos e negativos e também compara as duas em relação a uma série de fatores. A tabela 2 apresenta um comparativo de diferenças e similaridades existentes entre as metodologias tradicionais e as metodologias ágeis:

Tabela 2: Comparativo entre metodologias tradicionais e ágeis (BOEHM, 2006)

Metodologias tradicionais	Metodologias ágeis
São adaptativas: O desenvolvimento também ocorre de forma iterativa e evolutiva, com ondas sucessivas. A principal diferença é que cada iteração corresponde usualmente a uma fase do projeto, o que normalmente não é de curta duração.	São adaptativas: Partem do princípio de que o conhecimento sobre o problema aumenta à medida que o sistema vai sendo desenvolvido buscando constantemente melhores soluções. O desenvolvimento ocorre de forma iterativa e evolutiva, com iterações de 2 a 4 semanas.
São orientadas a processos e pessoas: Partem do princípio de que processos bem definidos devem ser impostos e executados, para garantir a qualidade do produto resultante.	São orientadas a pessoas: Para melhor gerenciar projetos de software, pessoas devem ser tratadas como indivíduos e não como recursos, pois cada pessoa apresenta um ritmo de trabalho único e completamente diferente, que é fruto da sua vivência pessoal.
São rígidas: Pressupõe que é possível especificar de antemão todos os requisitos do software a ser desenvolvido dificultando e muitas vezes impedindo realizar alterações e comprometendo a evolução natural da solução.	São flexíveis e iterativas: Adaptam-se constantemente ao conjunto de requisitos mais atual. A cada iteração usuários, clientes e desenvolvedores decidem sobre quais características devem ser adicionadas, modificadas e até retiradas do sistema.
São burocráticas: Geram muita sobrecarga no desenvolvimento a ser realizado, comprometendo a velocidade de desenvolvimento e, muitas vezes, o sucesso do projeto.	Simplicidade: Partem do princípio de que é preferível fazer algo simples de imediato e pagar um pouco mais para alterá-lo depois, se necessário, do que fazer algo complicado de imediato e acabar não utilizando depois.

Boehm (2006) conclui dizendo que o objetivo principal da abordagem ágil é simplificar o processo de desenvolvimento, minimizando e dinamizando tarefas e artefatos

que são importantes no processo, além do funcionamento do software. E na realidade, independentemente das metodologias ágeis, outras tendências em gerenciamento de projetos já indicavam uma certa convergência entre a comunidade de gerência e a comunidade técnica em relação à agilidade nos projetos de software.

2.2.4. Comparativos entre as metodologias ágeis na gerência de projetos

As metodologias ágeis estão chamando muito a atenção de engenheiros de software e pesquisadores de todo o mundo devido a sua simplicidade e agilidade. Abrahamsson (2003) propôs uma análise comparativa para verificar se as metodologias ágeis de desenvolvimento de software aderem a conceitos básicos de gerência de projetos. Esta análise comparativa foi focada em três perspectivas básicas: o gerenciamento de projetos, o ciclo de vida de desenvolvimento de software e em princípios abstratos *versus* uma orientação concreta.

Métodos devem ser eficientes em relação ao tempo e ao consumo de recursos (KUMAR; WELKE, 1992). Eficiência exige a existência de atividades de gerência de projeto para permitir a correta execução das tarefas de desenvolvimento de software. Gerência de projeto é uma função de apoio que fornece a espinha dorsal para o desenvolvimento eficiente de software. Devido a isso a perspectiva de gerência de projetos deve ser vista como uma dimensão relevante na avaliação de metodologias ágeis de desenvolvimento de software.

O ciclo de vida do desenvolvimento de um software é uma sequência de processos que uma organização utiliza para conceber e comercializar um produto de software (NANDHAKUMAR; AVISON, 1999). O ciclo de vida do desenvolvimento de um software é composto por nove fases: início do projeto, especificação de requisitos, design, codificação, teste unitário, teste de integração, teste de sistema, teste de aceitação e sistema em uso. Na perspectiva de ciclo de vida de um software é necessário observar quais as fases de desenvolvimento de software são cobertas pelas metodologias ágeis.

Métodos de desenvolvimento de software são frequentemente utilizados de outras maneiras do que as inicialmente pretendidas pelos seus autores (NANDHAKUMAR; AVISON, 1999). Assim, a fim de avaliar se as metodologias ágeis baseiam-se principalmente em princípios abstratos ou fornecem uma orientação concreta, a perspectiva de princípios abstratos *versus* orientação concreta também é necessária.

A seguir os métodos ágeis existentes são comparados usando três perspectivas: apoio à gestão, cobertura do ciclo de vida e tipo de princípio de orientação. Cada perspectiva será analisada separadamente.

- **Gerência de projetos:** É uma das grandes deficiências das metodologias de

desenvolvimento ágil. A XP foi suplementada com algumas orientações sobre a gestão de projetos mas ainda não oferece uma perspectiva global a respeito disso. Na *Scrum,* o gerenciamento de projetos é previsto para fins extremos de desenvolvimento ágil de software. Assim, Schwaber e Beedle (2002) sugerem a utilização de outros métodos para complementar o desenvolvimento de um software baseado na abordagem *Scrum*, nomeando a XP como uma alternativa. A abordagem promovida pela ASD é o modelo de gestão adaptativo (HIGHSMITH, 2002). O foco nas ASD é na alteração da cultura de desenvolvimento de software, afirmando que a gestão também terá que melhorar em resposta às mudanças nos projetos. FDD oferece meios para o planejamento por produto, recursos e acompanhamento do progresso dos projetos. FDD também acredita na capacitação dos gerentes e dentro de um projeto, este tem a última palavra a dizer sobre o escopo e o cronograma. DSDM sugere uma estrutura de controles para completar a abordagem de desenvolvimento rápido de aplicações. Todos esses controles são designados para aumentar a capacidade de organização para reagir às mudanças do negócio. Portanto, a abordagem DSDM para gerenciamento de projetos é em grande parte para facilitar o trabalho das equipes de desenvolvimento, com acompanhamento diário do andamento do projeto. No *Crystal* a solução para gerência de projeto centra-se no aumento da capacidade de escolher o método correto para a finalidade (COCKBURN, 2004).

- **O ciclo de vida do desenvolvimento de um software:** Os diferentes métodos ágeis são focalizados sobre os diferentes aspectos do ciclo de desenvolvimento de software. DSDM é um método independente no sentido de que existem tentativas de dar apoio total a todas as fases do ciclo de vida de um projeto. Outros são mais focados. ASD cobre todas as outras fases menos a fase de início do projeto, as fases de teste de aceitação e o sistema em uso. A família *Crystal* abrange as fases desde a concepção até teste de integração. XP, FDD e *Scrum* estão focados em especificação de requisitos, projeto, implementação e testes até o teste do sistema.
- **Princípios abstratos *versus* orientação concreta:** Um método que é útil e eficaz deve suportar orientações concretas e não deve referir-se apenas a princípios abstratos. (NANDHAKUMAR; AVISON, 1999). Orientações concretas, neste contexto, referem-se a práticas, atividades e produtos de trabalho que caracterizam e fornecem orientações sobre como uma tarefa específica deve ser executada. Por exemplo, FDD estabelece oito práticas que devem ser usadas em conformidade

com suas normas de desenvolvimento. Segundo Abrahamsson (2003), verificou-se que somente dois dos seis métodos ágeis de desenvolvimento de software incluídos na comparação realizada possuem princípios abstratos sobre uma orientação concreta. ASD é a metodologia mais completa. *Crystal*, depende da criticidade do sistema e do tamanho do projeto, mas não fornece nenhuma orientação sobre como executá-las. Na DSDM, as organizações devem desenvolver as práticas próprias. XP foi diretamente derivada de cenários da prática. Sendo assim, o forte deste método é sua orientação concreta. *Scrum* define as práticas para as fases de especificação de requisitos e de testes de integração. Fases de implementação não fazem parte do método.

A figura 5 apresenta uma comparação das três perspectivas para cada um dos métodos estudados. A barra superior indica se um método fornece apoio à gestão de projetos. A barra do meio compara o método com o ciclo de vida de desenvolvimento do software. Finalmente, a barra inferior mostra se o método baseia-se principalmente em princípios abstratos (cor branca) ou fornece orientações concretas (cor cinza). Em geral, a cor cinza em um bloco indica que o método compreende a perspectiva analisada enquanto que a cor branca indica falta de apoio.

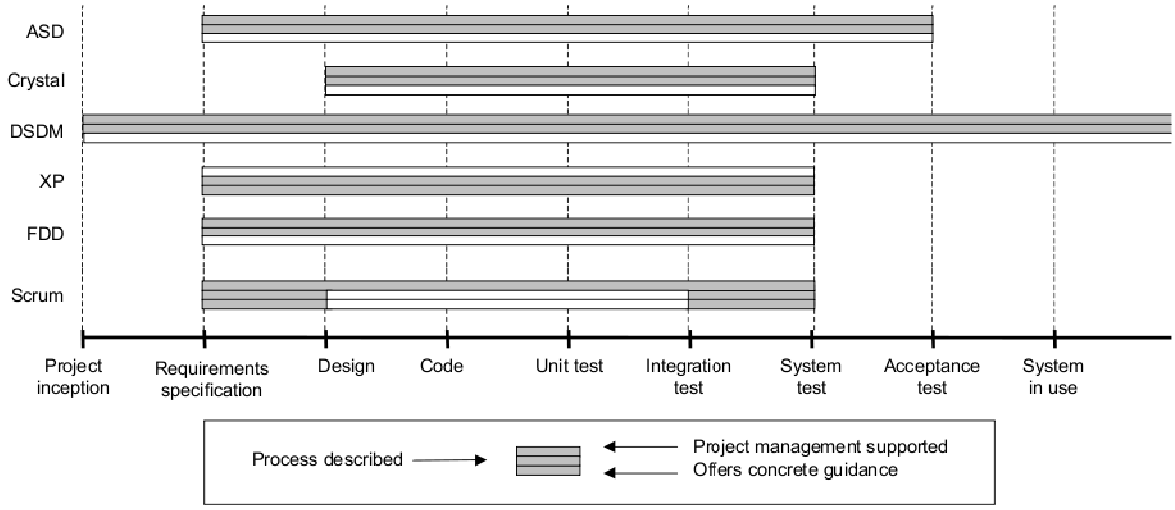


Figura 5: Perspectivas para metodologias ágeis (ABRAHAMSSON et al., 2003)

Após esta análise, Abrahamsson (2003) conclui que a maioria dos métodos ágeis incorporam suporte para a gestão de projeto mas o apoio verdadeiro ainda é escasso. Diante desta situação, a gerência do projeto é eficiente e de extrema importância quando os princípios ágeis como compilações diárias e *releases* curtos são seguidos. Com isso gestores do projeto estão em uma posição difícil quando uma decisão de abordagem mais apropriada deve ser feita. O sucesso operacional de qualquer método situa-se na sua capacidade de ser

incorporada ao ritmo do projeto de software. Desta maneira é necessário manter considerações que permitam assegurar o alinhamento entre o desenvolvedor e o gestor do projeto.

Além disso, segundo Abrahamsson (2003) os diferentes métodos ágeis cobrem o ciclo de vida do desenvolvimento de software nas diferentes fases. A questão que deve ser levantada é se é mais rentável cobrir mais fases e ser mais extenso ou cobrir menos fases e ser mais preciso. Além disso, os métodos que cobrem muitas fases acabam ficando demasiadamente gerais ou superficiais para serem usados. Por outro lado, métodos que cobrem muito pouco podem ser muito limitados ou podem ter falta de ligações a outros métodos.

E por fim Abrahamsson (2003) conclui que a minoria dos métodos ágeis oferecem uma orientação concreta. Princípios abstratos parecem dominar a literatura dos métodos ágeis e as mentes dos desenvolvedores. A comunidade ágil é mais preocupada em conseguir a aceitação de propostas e valores do que em oferecer orientação sobre como usar e operar versões destes valores. Atualmente, as orientações concretas existem principalmente nos métodos que são muito limitados em seu âmbito de aplicação ou profundidade.

2.3. Agentes de software

2.3.1. Introdução

A tecnologia de agentes de software vem fornecer soluções para abordar a crescente complexidade dos sistemas de computação que geralmente devem operar em ambientes não predizíeis, abertos e que mudam rapidamente. Estes sistemas devem ser capazes de decidir por si mesmos o que fazer em qualquer situação para alcançar seus objetivos. Esses tipos de sistemas são tipicamente difíceis de tratar usando concepções tradicionais de sistemas computacionais.

2.3.2. Conceito de agentes

Segundo Russel e Norvig (1995) um agente é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo utilizando-se de executores. Por exemplo, nos agentes humanos, os olhos e ouvidos são sensores e as mãos e a boca são executores.

A autonomia é a característica principal dos agentes. Os agentes são capazes de atuar sem intervenção humana ou de outros sistemas através do controle que eles possuem do seu estado interno e do seu comportamento.

Na construção de programas baseados em agentes, deve-se decidir como construir o

mapeamento de percepções a ações. A estrutura básica de um agente possui uma memória interna que irá atualizar-se com a chegada de novas percepções. Essa memória é utilizada nos procedimentos de tomada de decisão, os quais irão gerar ações para serem executadas. De forma a manter um histórico do comportamento do agente, a memória do agente é também atualizada a partir da ação selecionada.

Para Wooldridge (1999) um agente possui comportamento autônomo e flexível para alcançar seus objetivos, onde flexibilidade significa três coisas:

- **Reatividade:** os agentes inteligentes percebem seu ambiente e respondem em tempo às mudanças que nele ocorrem para alcançar seus objetivos;
- **Proatividade:** os agentes inteligentes são capazes de exibir comportamento orientado a objetivos tomando a iniciativa de forma a alcançar seus objetivos;
- **Habilidade social:** os agentes inteligentes são capazes de interagir com outros agentes e, possivelmente, com humanos para alcançar seus objetivos.

As principais contribuições do paradigma de desenvolvimento baseado em agentes provêm da área da inteligência artificial. Esta disciplina propõe quarto abordagens principais para a construção de sistemas computacionais inteligentes (RUSSEL; NORVIG, 1995):

- A abordagem do teste de *Turing (*sistemas que atuam como humanos);
- A abordagem cognitiva (sistemas que pensam como humanos);
- A abordagem das leis do raciocínio (sistemas que pensam racionalmente);
- A abordagem dos agentes racionais (sistemas que atuam racionalmente)

Os sistemas que exibem comportamento racional caracterizam a abordagem de agentes para a construção de sistemas inteligentes. Atuar racionalmente significa atuar para alcançar metas considerando as próprias crenças.

As ações tomadas por um agente racional são supostamente as corretas, aquelas que causam que o agente seja o mais bem-sucedido. Uma medida de desempenho estabelece o critério que determina quanto bem sucedido é um agente. Segundo Russel e Norvig (1995), a racionalidade de um agente em um dado momento é caracterizada através dos seguintes aspectos:

- A medida de desempenho que define o grau de sucesso;
- O histórico completo das percepções do agente;
- O conhecimento que o agente possui do ambiente;
- As ações que o agente pode realizar.

Um agente racional ideal deveria selecionar e executar aquela ação que é de esperar que maximize sua medida de desempenho, considerando o histórico das percepções e o

conhecimento que o agente possui.

A abordagem de agentes da inteligência artificial é mais geral que aquela baseada nas leis do raciocínio, onde a ênfase está na realização de inferências corretas. A realização de inferências às vezes caracteriza o comportamento de um agente porque uma das formas de se comportar racionalmente é raciocinar logicamente até a conclusão de que uma ação em particular permitirá atingir as metas do agente. Porém, realizar apenas inferências não é suficiente para exibir comportamento inteligente. Existem situações onde uma ação reflexiva simples pode ser mais efetiva que uma ação cuja decisão de execução é tomada mais lentamente através de algum mecanismo de inferência.

2.3.3. Arquiteturas de agentes

A arquitetura de um agente mostra como ele está implementado em relação as suas propriedades, a sua estrutura e como os módulos que a compõem podem interagir, garantindo sua funcionalidade. Em suma, a arquitetura de um agente especifica sua estrutura e funcionamento.

Uma outra definição vê a arquitetura de um agente como um modelo para a construção de sistemas computacionais que satisfazem as propriedades abordadas pela teoria de agentes, onde um agente é decomposto em componentes modulares. O conjunto destes módulos e suas interações devem prover respostas de como os sensores e o estado interno do agente determinam suas ações e seu próprio estado futuro.

Assim como existem inúmeros estilos de arquiteturas de software, o mesmo ocorre com relação às arquiteturas de agentes, as quais possuem certas características que permitem a avaliação de sua qualidade e eficácia. De acordo com Russel e Norvig (1995), alguns conceitos podem ser úteis para o desenvolvimento de uma arquitetura promissora:

- **Simplicidade:** Idealizar a arquitetura e seus componentes de forma a facilitar seu entendimento, implementação e manutenção;
- **Funcionalidade:** Selecionar uma arquitetura e ferramentas de desenvolvimento que focalizem aspectos específicos do problema a ser abordado;
- **Expansividade:** A arquitetura deve poder ser ampliada, uma vez que nem todas as necessidades futuras podem ser previstas em um primeiro momento;
- **Isolamento ou Portabilidade:** Uma arquitetura, para poder ser expansiva, deve possuir uma implementação portável, evitando-se soluções não padronizadas.

2.3.4. Tipos de agentes

Existem diferentes propostas de classificação dos tipos de agentes e arquiteturas. A

seguir são apresentadas as propostas de Russel e Norvig (1995) e de Wooldridge (1999).

Russel e Norvig (1995), definiram quatro tipos diferentes de agentes, possuindo cada um aspectos diferenciados na busca da solução de problemas:

- **Agentes reflexivos:** Cada percepção dispara alguma ação pré-estabelecida no programa, sendo esta relação uma simples regra do tipo <condição, ação> onde a condição é dada pela percepção do agente.
- **Agentes reflexivos que mantém registro do ambiente:** O estado interno do agente é atualizado, ou seja, ocorre um registro da sequência perceptual e das ações executadas pelo agente. A estrutura deste tipo de agente mostra como a percepção corrente combinada com o estado interno antigo gera a atualização do estado corrente.
- **Agentes baseados em metas:** Além de manterem um registro do estado do ambiente, estes agentes possuem uma meta que descreve o estado desejável a ser atingido. Então, o agente pode fazer uma combinação do desejável com os resultados de possíveis ações, para tomar decisões na busca da meta.
- **Agentes baseados na utilidade:** Existe uma preocupação entre os estados do agente. Uma medida de utilidade mais geral deve permitir uma comparação entre vários diferentes estados de acordo com o que se está procurando.

Wooldridge (1999) definiu quatro tipos de arquiteturas de agentes, cada uma com suas características específicas:

- **Arquiteturas baseadas em lógica:** Nas arquiteturas baseadas em lógica os agentes contêm um modelo simbólico do ambiente do agente, explicitamente representado em uma base de conhecimento e a decisão da ação a executar é tomada a através de raciocínio lógico. Esta arquitetura é utilizada nos agentes baseados em metas e agentes baseados na utilidade da classificação de Russel e Norvig (1995).
- **Arquiteturas reativas:** As arquiteturas reativas são aquelas que não possuem conhecimento do ambiente do agente e não utilizam raciocínio lógico. Elas se baseiam na suposição de que um agente pode desenvolver inteligência a partir de percepções e interações com seu ambiente, não necessitando de um conhecimento do mesmo. A decisão da ação a executar é tomada através de um mapeamento direto da situação para a ação. Esta arquitetura é utilizada nos agentes reflexivos na classificação de Russel e Norvig (1995)
- **Arquiteturas em camadas:** Nas arquiteturas em camadas a decisão da ação a

executar é sobre o ambiente em diferentes níveis de abstração. Esta abordagem, também conhecida como arquitetura híbrida, estrutura um agente em dois subsistemas: um deliberativo, contendo um modelo simbólico do ambiente que desenvolve planos e toma decisões via raciocínio lógico e um reativo, capaz de reagir a eventos que ocorrem no ambiente sem nenhum tipo de raciocínio.

- **Arquiteturas de crença-desejo-intenção (BDI)**: As arquiteturas BDI (*Belief-Desire-Intention*) (RAO; GEORGE, 1995) baseiam-se na teoria de que os seres humanos são regidos pôr três estados mentais fundamentais: crenças, desejos e intenções. O estado do agente é representado pôr três estruturas: suas crenças (*beliefs),* seus desejos (*desires)* e suas intenções (*intentions).* As crenças de um agente são o conhecimento que o agente possui sobre o ambiente em que ele se encontra. Os desejos de um agente representam o estado motivacional do sistema. As intenções de um agente são as ações que têm decidido realizar.

2.3.5. Coordenação entre agentes

A coordenação entre agentes refere-se à maneira em que os agentes estão organizados a fim de cooperar para alcançar um objetivo comum do sistema. Existem dois mecanismos básicos para coordenar os agentes: o mecanismo mestre-escravo e o mecanismo de mercado (PARAISO, 1997).

Nas arquiteturas do tipo “mestre-escravo” existem duas classes de agentes: os gerentes (mestres) e os trabalhadores (escravos). Os agentes trabalhadores são coordenados por um gerente que distribui as tarefas entre estes e espera o resultado. O agente facilitador pode existir se os agentes estiverem divididos em grupos.

Nas arquiteturas baseadas no mecanismo de mercado, todos os agentes estão em um mesmo nível e sabem as tarefas que cada agente é capaz de desempenhar. Esta arquitetura visa diminuir a quantidade de mensagens trocadas, tendo em vista que cada agente já conhece todos os outros. Baseado nesse mecanismo surgiu a Programação Orientada ao Mercado, onde os agentes podem ser classificados como produtores e consumidores e são utilizados para maximizar lucros através de um processo de negociação (PARAISO, 1997)

2.3.6. Comunicação entre agentes

A comunicação é uma importante característica que os agentes inteligentes possuem para atingirem seus objetivos. A linguagem ACL (*Agent Communication Language*) foi definida para a comunicação entre agentes e pode ser dividida em (FININ et al., 1993):

- **Vocabulário:** Dicionário de conceitos do domínio de aplicação do sistema de agentes e

seus relacionamentos. Também conhecido como ontologia.

- **Linguagem interna:** KIF (*Knowledge Interchange Format*), linguagem de programação em lógica de primeira ordem com capacidade de codificar dados simples, restrições, regras e expressões.
- **Linguagem externa:** KQML (*Knowledge Query and Manipulation Language*) que é uma camada linguística capaz de encapsular estruturas KIF, fornecendo uma comunicação mais eficiente.

Uma mensagem ACL nada mais é do que uma expressão KQML onde os argumentos são termos ou sentenças KIF formadas por termos da ontologia.

2.3.7. Metodologias de engenharia de software orientada a agentes

Algumas metodologias voltadas ao projeto de sistemas orientados a agentes surgiram ao longo dos tempos, sendo que as mais discutidas na literatura atualmente são: MaSE, Prometheus, Tropos e Gaia. Essas metodologias foram selecionadas baseadas em vários fatores, como a abrangência na engenharia de software orientada a agentes, a difusão na comunidade de pesquisadores e os recursos disponíveis pelas metodologias como documentação detalhada e de fácil acesso e suporte à ferramenta. A seguir, serão expostas algumas das características de cada uma destas metodologias:

- **MaSE:** É uma metodologia orientada a agentes e o significado da sigla é *Multiagent Systems Engineering Methodology*. Nela os agentes não são considerados como autônomos e pró-ativos. Eles são processos de software simples que interagem uns com os outros para atingir a finalidade geral do sistema (DELOACH; WOOD, 2001). O objetivo principal da MaSE é guiar o projetista através de todo o ciclo de vida do software, independentemente de qualquer arquitetura de agente, linguagem de programação ou sistema de troca de mensagens. A metodologia MaSE consiste da fase de análise e da fase de projeto e possui uma ferramenta denominada *AgentTool*[2] que auxilia no processo de modelagem.
- **Prometheus:** É uma metodologia na engenharia de software orientada a agentes para especificar, projetar e implementar sistemas de agentes (PADGHAM; WIKIKOFF, 2005). O principal objetivo no desenvolvimento desta metodologia é ter um processo definido, que possa ser usado no desenvolvimento de sistemas de agentes e que possa ser ensinado aos profissionais da área e aos estudantes que não

2 http://agenttool.cis.ksu.edu

possuam conhecimento em agentes. Prometheus suporta o desenvolvimento de agentes inteligentes que possuem objetivos, crenças, planos e eventos. A metodologia utiliza o conceito de capacidade, a qual pode ser composta por planos, eventos, crenças e outras capacidades que dão habilidades específicas ao agente.

- **Tropos:** É uma metodologia de desenvolvimento de software orientado a agentes, criada por um grupo de pesquisadores de várias universidades do Brasil, do Canadá e da Itália (BRESCIANI et al., 2004). A metodologia usa a noção de agente em seus conceitos, tais como objetivos, planos e tarefas durante as fases do desenvolvimento do software e adota uma abordagem de refinamento de passos, utilizando um conjunto específico de operadores de transformação. Diferentemente das demais metodologias apresentadas, Tropos tem um foco maior na fase de requisitos iniciais, onde os conhecimentos e as necessidades dos clientes são identificados e analisados. A metodologia consiste das fases de requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado e implementação.
- **Gaia:** É uma metodologia de uso geral, aplicável à uma gama de domínios de aplicação. Para Gaia, um sistema multi-agente é visto como uma organização composta de vários papéis. A metodologia provê um *framework* conceitual que permite ao analista sistematicamente refinar requisitos de maneira tal que estes possam ser diretamente implementados, detalhando papéis, interações, responsabilidades, permissões e propriedades na fase de análise (WOOLDRIDGE, 1999). Apesar de ser uma metodologia de uso geral, Gaia é voltada aos aspectos organizacionais e fornece um formalismo que dá uma ênfase muito maior à etapa de análise, sendo que a etapa de projeto provê apenas um detalhamento mais refinado, produto do trabalho já realizado.

Além das metodologias de desenvolvimento de software orientado a agentes citadas acima, existe a linguagem de modelagem AUML (*Agent Unified Modeling Language*) que é de uma extensão da linguagem de modelagem UML (*Unified Modeling Language*), proposta por Odell et al. (2000), concebida para dar suporte ao desenvolvimento de sistemas orientados a agentes. Essa extensão visa exclusivamente estender a linguagem UML de modo a dar suporte à especificação de protocolos de interação entre agentes, não se preocupando em fornecer artefatos para todo o ciclo de vida do desenvolvimento de um sistema orientado a agentes. Os diagramas AUML são utilizados em diversas metodologias orientadas a agentes com o objetivo de fornecer uma semântica semi-formal e intuitiva através de uma notação

gráfica amigável para o desenvolvimento de protocolos de interação entre agentes.

3. TRABALHOS RELACIONADOS

Neste capítulo serão apresentados alguns dos principais trabalhos relacionados ao tema proposto. Neste sentido, serão abordados trabalhos que realizam estudos comparativos entre metodologias ágeis, tradicionais e o desenvolvimento iterativo e incremental, comparação entre *frameworks* de avaliação para metodologias ágeis assim como também trabalhos referentes a sistemas multi-agentes baseados em métricas para a gerência de projetos e integração com o desenvolvimento de software. Todos estes trabalhos contribuíram, de certa forma, como referência para a criação do modelo abstrato, objetivo deste trabalho.

3.1. Uma proposta de integração entre métodos ágeis e métodos tradicionais

Kee (2006) sugeriu uma integração global de métodos ágeis com algumas boas características de métodos tradicionais para melhorar as técnicas de desenvolvimento de software atual. Foi foco de seu estudo o método cascata e os pontos mais relevantes dos principais métodos ágeis existentes. Foram identificados conflitos específicos que as empresas enfrentam e sugeridas estratégias para resolvê-los com o intuito de tornar-se mais competitivo integrando as metodologias tradicionais e modernas.

Segundo Kee (2006), o método em cascata é definido como sendo um método rígido composto por cinco fases que devem ser executadas sequencialmente: levantamento de requisitos, projeto, implementação, testes e manutenção. Assume-se que todos os requisitos tenham sido levantados na primeira fase do projeto. Obviamente poderão ficar de fora alguns detalhes importantes porque na maioria das vezes os interessados não conseguem visualizar todo o sistema no início do projeto. Em consequência disso, mudanças de requisitos representam um desafio e um custo muito grande para projetos desenvolvidos na metodologia em cascata porque quando isso ocorrer, o desenvolvedor é forçado a refazer todas as fases do projeto desde o seu início.

De acordo com Kee (2006) as metodologias ágeis de desenvolvimento de software estão sendo bem aceitas como alternativas para os métodos tradicionais. Elas possibilitam uma maior qualidade no software em um curto período de tempo. Os métodos ágeis promovem um mecanismo mais dinâmico para a produção de software aumentando as iterações a pelo menos uma vez por dia em oposição a uma vez por fase como ocorre na maioria dos métodos tradicionais.

Em geral, os métodos ágeis empregam curtos ciclos iterativos, envolvendo ativamente os usuários, priorizando e verificando necessidades e confiando no conhecimento tácito da equipe como oposição à documentação. Os conceitos chave das metodologias ágeis são:

- Indivíduos e interação sobre processos e ferramentas;
- Software produzido sobre documentação abrangente;
- Colaboração com o cliente sobre contrato de negociação;
- Responder a mudanças mais que seguir um plano.

A figura 6 apresenta uma relação entre as curvas de custos decorrentes de solicitações de alterações em projetos utilizando metodologias tradicionais, evidenciado por Barry Boehm e o mesmo impacto financeiro gerado pelas solicitações em projetos que utilizam metodologias ágeis, defendido pelo visionário Kent Beck:

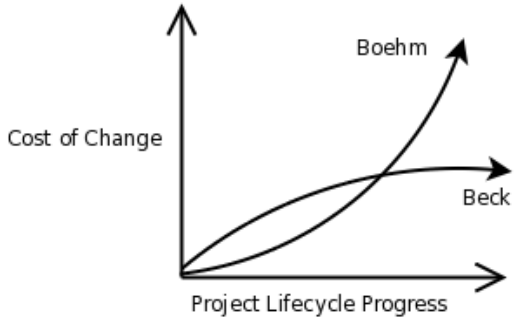


Figura 6: Curvas de custo (KEE, 2006)

Segundo Kee (2006), embora os métodos ágeis provaram ser eficientes, existem algumas complexidades inerentes que combinadas com os métodos tradicionais podem vir a compor uma solução mais robusta. A figura 7 apresenta algumas destas soluções:

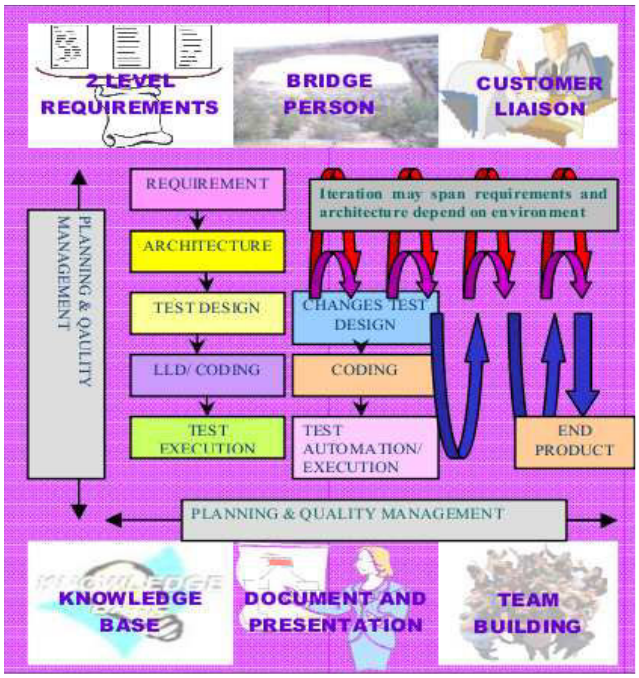


Figura 7: Fluxo de integração recomendado (KEE, 2006)

Segue abaixo uma explicação dos pontos analisados no fluxo de integração proposto

por Kee (2006):

- **Requisitos ambíguos:** Em métodos ágeis, a incidência de requisitos ambíguos é maior devido as solicitações serem escritas como *user stories* que dependem fortemente da comunicação face a face (MOHAN, 2005). Para resolver este problema Kee (2006) sugere dois níveis de requisitos. O primeiro é tratado a nível de *user stories* e inclui funcionalidades que ainda não estão plenamente analisadas. O segundo nível é feito com a colaboração do cliente ajudando a esclarecer e identificar o trabalho da próxima iteração. Além disso, para evitar requisitos ambíguos, o autor sugere eleger uma pessoa, denominada por ele de *Bridge Person*, que vai participar de reuniões de equipe proporcionando assim uma documentação útil e sustentável que ajude o usuário e o desenvolvedor.
- **Projetos grandes e distribuídos:** Em grandes projetos distribuídos normalmente o cliente não pode estar presente em tempo integral. Diante disso, a falta de comunicação com o cliente pode repercutir no progresso de grandes projetos distribuídos (MOHAN, 2005). Para resolver isso, Kee (2006) recomenda uma pessoa de ligação com o cliente (*Customer Liaison*). O papel desta pessoa é possibilitar uma resposta rápida aos pedidos de clientes que não estão sempre presentes. Além disso ela é responsável pela construção de um canal de comunicação e de um relacionamento contínuo com estes clientes, mantendo a velocidade dos métodos ágeis.
- **Conflitos de processos:** Os conflitos de processos podem ocorrer de duas maneiras. A primeira é em relação a como evoluir os processos ágeis e a segunda é sobre a forma de como mesclar métodos de desenvolvimento ágeis com tradicionais (BOEHM; TURNER, 2005). Para isso, Kee (2006) sugere documentar e comunicar com clareza os processos para todas as partes envolvidas. O objetivo é apresentar este documento para clientes e membros da equipe de desenvolvimento aumentando assim o seu comprometimento.
- **Conflitos entre pessoas:** A adoção e integração de métodos ágeis pode causar conflitos entre pessoas. Mudanças de paradigma no processo podem não ser aceitos pelos membros da equipe e clientes (BOEHM; TURNER, 2005). A fim de integrar métodos ágeis com tradicionais, a construção de uma boa base de conhecimento tem um papel muito importante no aumento da produtividade da equipe e permite que os clientes tenham um acompanhamento mais claro do que realmente está acontecendo.

Kee (2006) conclui que a integração entre métodos ágeis e tradicionais é o próximo passo a ser seguido por empresas que possuem equipes geograficamente distribuídas ou diversas culturas e experiências. Ter uma melhor colaboração, comunicação e uma documentação concisa permitirá uma integração mais rápida, eliminando a repetição de métodos ineficazes. A *Bridge Person* compartilha os conhecimentos entre a equipe e os clientes contribuindo para a eficácia de um projeto de desenvolvimento de software.

3.2. Uma comparação entre métodos ágeis e o desenvolvimento iterativo incremental

Fagundes et al. (2008) visando auxiliar os profissionais interessados na utilização da abordagem ágil a escolherem uma metodologia que melhor se adapte às necessidades dos seus projetos, equipes ou organizações, propuseram uma comparação entre os processos dos métodos ágeis XP, *Scrum,* FDD e ASD em relação ao desenvolvimento iterativo e incremental.

Os critérios adotados para servirem de base para a identificação das atividades propostas pelas metodologias ágeis são fundamentados no desenvolvimento incremental. E com isso as atividades sugeridas durante o seu processo de desenvolvimento são bastante semelhantes aos princípios ágeis (SOMMERVILLE, 2003).

O desenvolvimento incremental inicialmente identifica, em um esboço, os requisitos do sistema e seleciona quais são os mais e os menos importantes. Em seguida é definida uma série de iterações de entrega, onde em cada uma é fornecido um subconjunto de funcionalidades, dependendo das suas prioridades. Após a identificação dos incrementos, as funcionalidades a serem entregues na primeira iteração são detalhadas e desenvolvidas. Paralelamente a este desenvolvimento, outras funcionalidades podem ser analisadas para fazerem parte dos outros incrementos. Uma vez que cada incremento é concluído e entregue, ele pode ser colocado em operação. A medida que novos incrementos são concluídos, eles são integrados às iterações existentes, de modo que o sistema melhora a cada novo incremento entregue (SOMMERVILLE, 2003). A figura 8 apresenta as etapas do desenvolvimento incremental:

Fagundes et

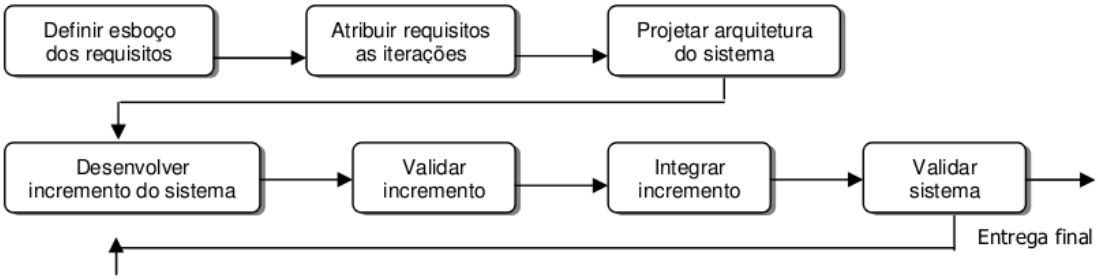


Figura 8: Desenvolvimento incremental (SOMERVILLE, 2003)

al. (2008) possibilitaram uma melhor compreensão da comparação realizada para cada uma das atividades do desenvolvimento incremental, apresentando uma pequena explicação da atividade correspondente em cada um dos métodos analisados, conforme detalhado na tabela 3:

Tabela 3: Desenvolvimento incremental em métodos ágeis (FAGUNDES et al., 2008)

Atividade	**Especificação**
Definição do Esboço dos Requisitos	**XP:** Clientes escrevem as *user stories*. ***Scrum:*** Definição do *Product Backlog*. **FDD:** Geração de artefatos para a documentação dos requisitos. **ASD:** Requisitos definidos nas sessões *Joint Application Development* (JAD).
Atribuição dos Requisitos as Iterações	**XP:** Equipe técnica e clientes definem as *user stories* que serão desenvolvidas nas iterações. As iterações duram de 1 a 4 semanas. ***Scrum:*** Definição do *Sprint Backlog* e as *Sprints (*iterações) duram no máximo 30 dias. **FDD:** As características são agrupadas, priorizadas e distribuídas aos responsáveis pelo seu desenvolvimento e as iterações duram no máximo 2 semanas. **ASD:** Definição do número de iterações que duram de 4 a 8 semanas.
Projeto da Arquitetura do Sistema	**XP:** Propõe que paralelamente à escrita das *user stories*, seja realizado o projeto da arquitetura do sistema. ***Scrum:*** Sugere que seja feito um projeto geral do sistema baseado nos itens do *Product Backlog*. **FDD:** Sugere que seja construído um diagrama de classes da UML para representar a arquitetura do sistema. **ASD:** Não especifica nada para esta atividade.
Desenvolver Incremento do Sistema	**XP:** Implementação das *user stories* que fazem parte da iteração corrente por duplas de programadores. ***Scrum*:** Implementação dos requisitos contemplados no *Sprint Backlog* para a *Sprint* corrente. **FDD:** Análise da documentação existente, refinamento do modelo gerado nas atividades anteriores e implementação das características que serão desenvolvidas durante a iteração corrente. **ASD:** Implementação dos requisitos que fazem parte da iteração corrente.
Validar Incremento	**XP:** Os programadores executam os testes de unidade e os clientes executam os testes de aceitação. ***Scrum:*** Não adota nenhum processo de validação pré-definindo. **FDD:** Os testes e inspeções são executados pelos próprios programadores após a implementação. **ASD:** São verificadas a qualidade técnica e funcional do sistema.
Integrar Incremento	**XP:** A integração acontece paralelamente ao desenvolvimento das

Atividade	Especificação
	user stories. ***Scrum:*** Atividade realizada ao final de cada *Sprint.* **FDD:** Atividade realizada após os testes no incremento. **ASD:** Não especifica nada para esta atividade.
Validar Sistema	**XP:** O sistema é disponibilizado ao cliente para que o mesmo realize validações. ***Scrum:*** O cliente valida o sistema integrado em uma reunião no último dia da *Sprint.* **FDD:** Esta atividade ocorre através das inspeções e dos testes de integração. **ASD:** Não especifica nada para esta atividade.
Entrega Final	**XP:** Cliente satisfeito com o sistema. ***Scrum:*** Todos os itens no *Product Backlog* desenvolvidos. **FDD:** O sistema é entregue após todos os conjuntos de características implementados. **ASD:** Todos os requisitos desenvolvidos.

Fagundes et al. (2008) concluíram que os métodos ágeis selecionados para fazerem parte do estudo apresentam atividades bastante semelhantes em relação aos seus processos de desenvolvimento. Porém algumas atividades apresentam particularidades, como por exemplo, a programação em dupla, as *user stories* escritas pelos clientes e a escrita dos testes antes da implementação da XP, as reuniões de planejamento, diárias e de revisão adotadas pela *Scrum,* as inspeções de código de FDD e as sessões JAD sugeridas pelo ASD. No entanto, o estudo possibilita aos interessados em utilizar a abordagem ágil no desenvolvimento de seus sistemas, obterem uma visão geral das atividades propostas pelos métodos ágeis de maneira a auxiliá-los na escolha de qual método adotar ou mesmo, de reduzir o tempo gasto com estudos sobre métodos que não se adaptarão as suas necessidades.

3.3. Uma avaliação comparativa de *frameworks* para metodologias ágeis

A emergência por metodologias ágeis de desenvolvimento levou Taromirad e Ramsin (2008) a publicarem um artigo fazendo uma análise comparativa de *frameworks* de avaliação para metodologias ágeis, identificando suas fragilidades, potencialidades, semelhanças e diferenças. Para ser útil a gerentes de projeto, a avaliação levou em conta os diversos parâmetros de projetos de desenvolvimento de software e acentuou as características e aplicações das diferentes metodologias ágeis avaliadas.

O objetivo do trabalho foi analisar avaliações existentes sobre *frameworks* de metodologias ágeis para identificar as suas deficiências e, em seguida, fornecer orientações para melhorar esta avaliação. O primeiro passo foi coletar e definir todos os requisitos e

características que um *framework* de avaliação deve abordar. Estes requisitos e características são expressos como critérios de avaliação e utilizados como meta-critérios para avaliar os outros *frameworks* existentes.

Abrahamsson et al. (2002) introduziram uma estrutura na qual os processos, papéis, responsabilidades, práticas, experiências e utilização em relação a cada metodologia são identificados. Eles têm utilizado esta estrutura como base analítica para verificar as diferenças e semelhanças entre as diferentes metodologias de desenvolvimento ágil de software. Esta análise foi realizada comparando metodologias ágeis em duas partes:

- Comparação de adoção, referindo-se ao âmbito de utilização;
- Comparação de características gerais tendo como referência a cobertura do ciclo de vida do desenvolvimento, apoio a gerência do projeto e definição de processos claros e práticas aplicáveis.

Em outro trabalho, Abrahamsson et al. (2003) propuseram um *framework* analítico para a análise de metodologias ágeis. Baseado em seus trabalhos anteriores, o foco desta proposta foi uma análise da cobertura do ciclo de vida do desenvolvimento de software, o suporte ao gerenciamento de projetos e a incidência de princípios abstratos *versus* orientação concreta, determinando se as metodologias ágeis fornecem orientação concreta ou somente invocam regras abstratas.

Koskela (2003) introduziu uma série de análises para verificar a gerência da configuração do software (*Software Configuration Management*) em metodologias ágeis. São definidas análises para o planejamento, o gerenciamento de mudanças e para as ferramentas.

Williams et al. (2004) apresentam um valor de referência para a avaliação das práticas da XP que uma organização possa ter selecionado permitindo customizar os seus resultados. Este *framework* é chamado XP-EF e grava informações de contexto de um projeto em categorias pré-definidas.

Germain e Ribillard (2005) fizeram uma comparação entre um processo de engenharia baseado em RUP (*Rational Unified Process*) e um processo ágil construído em torno dos princípios da metodologia XP analisando o tempo gasto em cada uma das atividades.

Um dos últimos trabalhos analisados foi o 4-DAT, uma ferramenta de avaliação baseada em *frameworks* de análise e comparação dos métodos ágeis, proposto por Qumer e Hendersson-Sellers (2006). O objetivo da 4 DAT é fornecer um mecanismo para medir o grau de agilidade de qualquer método utilizando determinadas práticas.

A tabela 4 apresenta um panorama geral dos *frameworks* analisados por Taromirad e Ramsin (2008).

Tabela 4: Panorama dos *frameworks* avaliados (TAROMIRAD; RAMSIN, 2008)

Framework	**Principais objetivos**	**Principais critérios de avaliação**
Abrahamsson et al. (2002)	Identificação de diferenças e similaridades entre metodologias ágeis.	- Define uma estrutura comum para descrever metodologias. - Cobre os ciclos de vida de desenvolvimento de software. - Suporta a gerência de projetos. - Define processos claros e práticas aplicáveis.
Abrahamsson et al. (2003)	Identificação de novos rumos das metodologias ágeis através de uma análise comparativa.	- Cobre os ciclos de vida de desenvolvimento do software. - Apóia a gestão de projetos. - Define princípios abstratos e orientações concretas.
Koskela (2003)	Analisando a gerência de Configuração de Software em metodologias ágeis.	- Suporta abordagem, planejamento, configuração, gestão da mudança e ferramentas de gerência de configuração.
Williams et al. (2004)	Criação de uma referência para avaliar as práticas utilizadas na XP e os seus resultados.	- Define medidas objetivas e subjetivas como a análise qualitativa de métricas de desenvolvimento tradicional de software.
Germain et al. (2005)	Comparação entre um processo de engenharia e um processo ágil	- Define um conjunto de atividades e mede o tempo e o esforço despendido em cada uma delas.
Qumer et al. (2006)	Introduzindo um *framework* para análise e comparação dos métodos ágeis (4-DAT)	- Fornece um mecanismo para medir o grau de agilidade em qualquer método definido.

Taromirad e Ramsin (2008) concluem que apesar do elevado número de métodos ágeis existentes e da utilização generalizada destas metodologias, ainda são difíceis de encontrar respostas relativas a questões de adequações de processos ágeis em projetos específicos. Para responder a estes desafios, a metodologia precisa ter uma estrutura adequada de avaliação. Através desta avaliação, os autores também concluíram que a maioria dos *frameworks* de avaliação existentes não abordam questões relacionadas com a agilidade mesmo sendo esta a característica marcante das metodologias ágeis.

3.4. Um sistema multi-agente baseado em métricas para gerência de projetos

Chang et al. (2009) desenvolveram um trabalho que apresenta um sistema multi-agente denominado *Software Project Planning Associate* (SPPA) que auxilia os gestores na visualização e acompanhamento do processo de gerência de projetos de um software durante o desenvolvimento do mesmo. Os recursos, tarefas, cronogramas e metas do projeto de software são descritos em métricas genéricas estipuladas no plano de projeto. No processo de

desenvolvimento de software as métricas são discretamente recolhidas pelo agentes inteligentes. Baseado nisso são geradas dinamicamente ao gerente do projeto recomendações e previsões eficazes que indicam o que deve mudar na execução do desenvolvimento do software para melhor cumprir com o plano de projeto estipulado.

Sem um plano realista e objetivo, um projeto de desenvolvimento de software não pode ser gerido de forma eficaz (WU, 2004). O plano de projeto prevê um cronograma de atividades e marcos que, juntos, visam alcançar o objetivo final. No entanto, um plano de projeto de software é baseado em estimativas de tamanho e duração das atividades. Como cada resultado pode ter um impacto sobre as estimativas existentes, o gerente pode necessitar alterar este plano. Os erros de planejamento do projeto de software são assim como erros de desenvolvimento de software: quanto mais cedo forem resolvidos, menos impacto terão no sucesso do projeto.

Diante desta situação existem três problemas gerais no ambiente de gerenciamento de projetos (CHANG et al., 2009):

- **Planejamento a força bruta:** Gerentes utilizam ferramentas inadequadas para gerenciar projetos de software;
- **É fácil se perder:** Eles não sabem quando os problemas vão acontecer e não sabem como aperfeiçoar o plano para resolvê-los.
- **Os projetos são muitas vezes geograficamente dispersos:** Em ambientes modernos de desenvolvimento de software onde as organizações são separadas e dispersas entre os países e continentes torna-se muito difícil gerenciar um projeto.

De acordo com Chang et al. (2009), com todas as ferramentas de planejamento de projeto existentes atualmente, o gerente precisa se lembrar de uma grande variedade de atributos de projeto e pode ser difícil para os gestores transmitir isso de um plano para outro ou para quem deve executar atribuições definidas por ele. Diante disso, o SPPA foi desenvolvido para ajudar os gestores a consolidarem informações a respeito do projeto a partir de qualquer ambiente de desenvolvimento. A figura 9 mostra como SPPA coleta dados para apresentar o estado exato de um projeto de software. Objetos juntamente com seus atributos, relacionamentos e propriedades são armazenados na base de conhecimento do SPPA. Esta ferramenta pode ser usada para monitorar, medir e calcular atributos de um projeto de software além de gerar relatórios precisos de gestão.

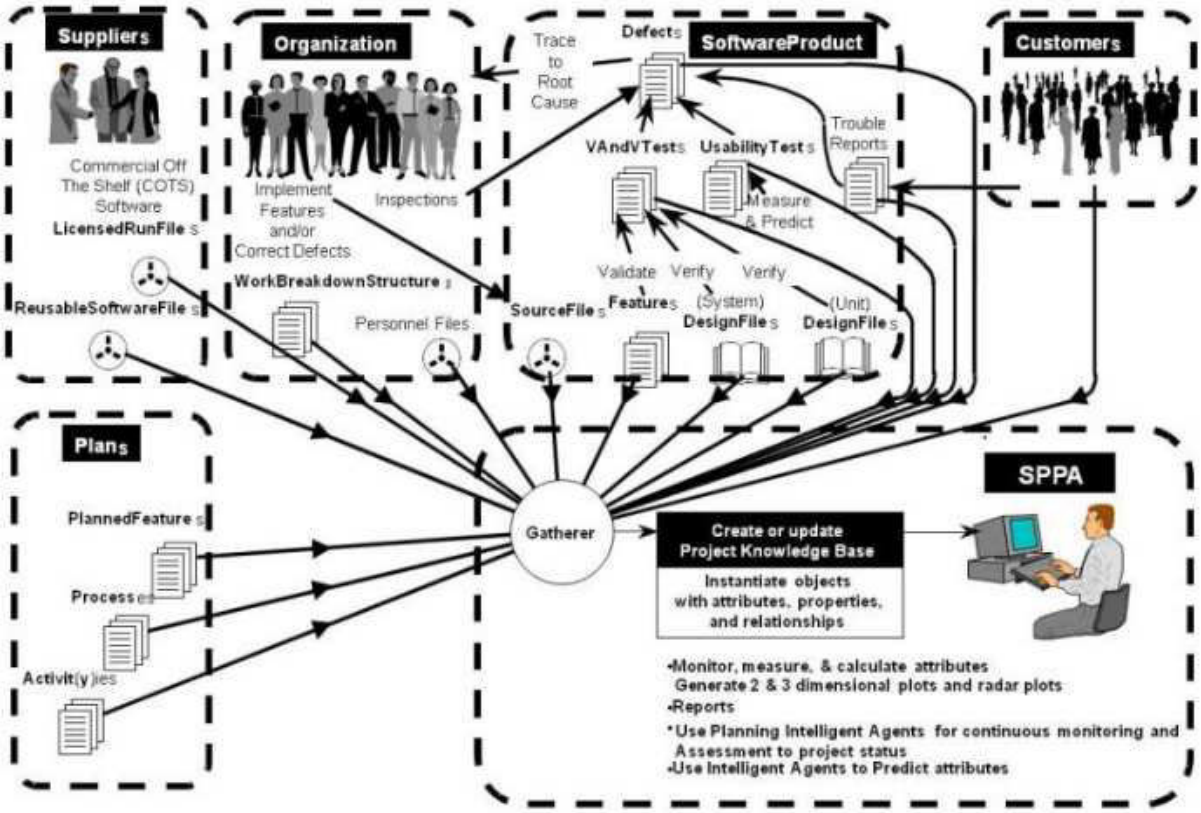


Figura 9: Estrutura do SPPA (CHANG et al., 2009)

O SPPA usa arquitetura de três camadas e um sistema especialista como ferramenta para a criação de agentes inteligentes. Além disso ele é dividido em subsistemas conforme apresentado na tabela 5:

Tabela 5: Subsistemas do SPPA (CHANG et al., 2009)

Subsistema	**Características**
Planejamento	O gestor escolhe a metodologia de desenvolvimento (cascata, incremental, espiral, etc) para iniciar o plano de projeto. É fornecido um *gantt* de forma que o gerente acompanhe o andamento das tarefa e dos caminhos críticos do projeto.
Agentes inteligentes de planejamento	Mantêm o controle da situação do projeto prevendo alertas e problemas.
Agentes inteligentes de métricas e recomendações	Traduzem e calculam as métricas dos atributos do projeto e fornecem recomendações ao gestor.
Grupo de pessoal	Mantém informações relacionadas com os recursos humanos (salário, experiência, tarefas, etc).
Base de conhecimento	Contém todas as regras e fatos do projeto para que os agentes inteligentes possam utilizar.

Chang et al. (2009) definiram um conjunto de métricas genéricas para avaliar atributos de gerência de projetos de software. Neste conjunto são armazenados em base de

conhecimento os dados dos atributos para que os agentes inteligentes possam auxiliar os gerentes. A figura 10 apresenta as métricas genéricas para planejamento de projetos de software definidas pelo autor.

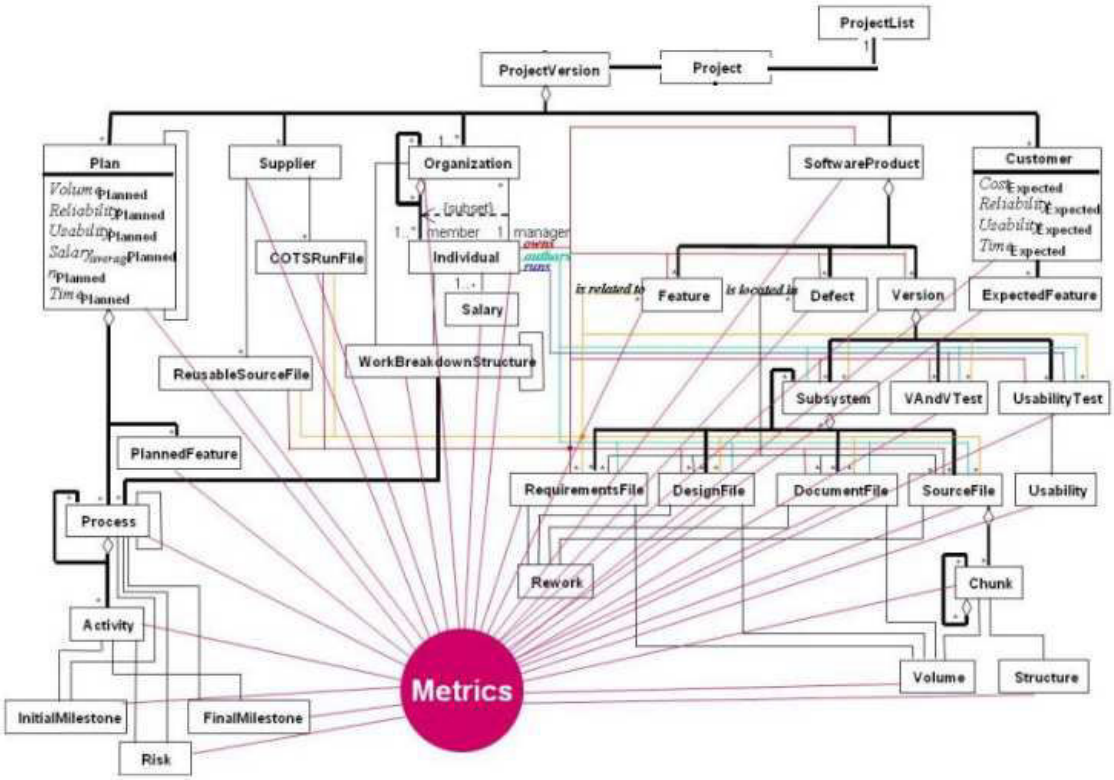


Figura 10: Métricas para planejamento de projetos de software (CHANG et al., 2009)

Os agentes inteligentes utilizam os atributos definidos no conjunto de métricas genéricas para calcular métricas específicas. Por exemplo, se um gerente quiser saber o estado atual de produtividade de um determinado sistema, os agentes deverão utilizar outras duas métricas tais como volume e esforço para calcular o valor da métrica produtividade.

Segundo Chang et al. (2009), o SPPA possui vários agentes inteligentes que ajudam o gerente acompanhar o andamento do projeto. Os agentes relatam problemas e fazem recomendações para prevenir potenciais riscos que possam vir a ocorrer durante todo o ciclo de vida do projeto. A arquitetura dos agentes inteligentes é mostrada na figura 11:

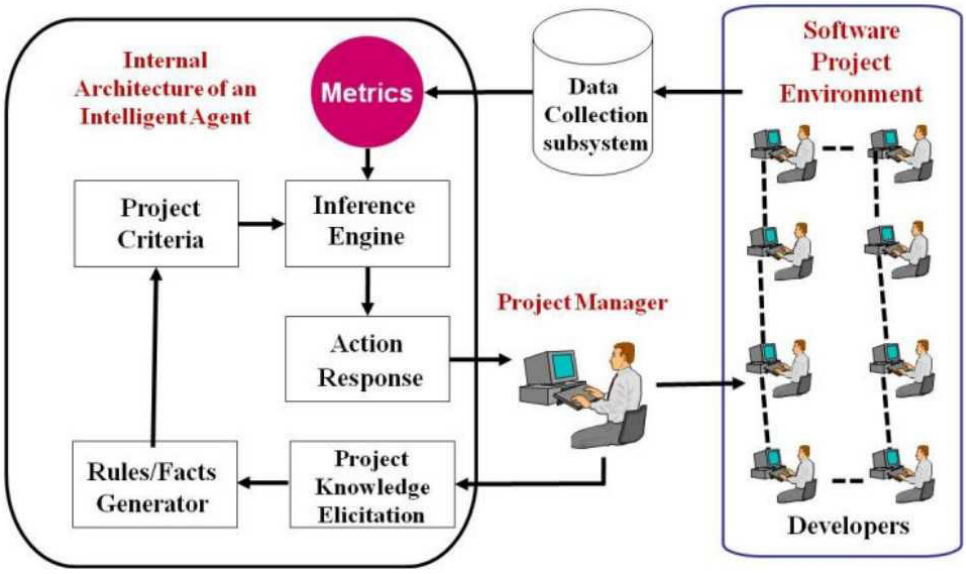


Figura 11: Arquitetura dos agentes inteligentes (CHANG et al., 2009)

Um subsistema coleta os dados em tempo real das estações de trabalho de desenvolvimento e envia estes dados para o agente inteligente. Em seguida, o agente recupera as métricas de projeto necessárias e calcula as relações existentes entre elas. Os marcos das atividades e o tempo de duração prevista do projeto são definidos pelo gestor do projeto. Em seguida o módulo *Project Knowledge Elicitation* transmite ao projeto conhecimentos necessários e define regras de inferência. Estas regras geram critérios de projeto. O motor de inferência carrega as métricas de software e os critérios de projeto como insumos para fazer as medidas adequadas.

Chang et al. (2009) concluem o trabalho afirmando que softwares de controle de gerência de projetos são possíveis mesmo que as equipes de desenvolvimento se encontram dispersas em todo o mundo. O SPPA pode ser executado em qualquer ambiente de computador e pode ser usado para qualquer ambiente de projeto de software para ajudar a definir um plano, coletar dados de atributos de projeto, fazer previsões, calcular métricas de software, estimar os atributos e controlar o andamento do projeto. Com o apoio de vários agentes inteligentes, o SPPA pode fazer recomendações aos gestores para evitarem riscos, melhorando o plano do projeto. Ao utilizar o SPPA, os gestores conseguem detectar os problemas à medida que acontece o desenvolvimento e podem tomar ações corretivas imediatamente.

3.5. Proposta de integração entre gerência de projetos e desenvolvimento de software

Rosito et al. (2008) apresentaram uma proposta de modelo de integração entre a gerência de projetos projetos e o processo de desenvolvimento de software denominado *Software Planning Integrated Model* (SPIM). Este modelo inclui um conjunto de regras para o

planejamento integrado de atividades gerenciais e produtivas para projetos de desenvolvimento de software.

Foram desenvolvidos os conceitos advindos da integração do PMBOK com o RUP tornando-se possível o desenvolvimento de ferramentas para o apoio ao processo de planejamento e acompanhamento das atividades gerenciais e produtivas para projetos de desenvolvimento de software. Neste sentido, foi desenvolvida a ferramenta denominada *Software Planning Integrated Tool* (SPIT) que atua como um *Add-in* para uma ferramenta de gerenciamento de projetos comercial muito utilizada atualmente.

Rosito et al. (2008) fizeram um estudo detalhado dos modelos do PMBOK e do RUP o que permitiu identificar como são organizados e quais as relações válidas entre os elementos de cada modelo. Através da integração entre a gerência de projetos e os processos de desenvolvimento de software foi possível identificar as principais características e discrepâncias entre os elementos de tais modelos.

O critério de integração entre os modelos proposto por Rosito et al. (2008) respeitou três regras básicas:

- **Sobreposição de conceitos:** Unificou-se o conceito dentro de um pacote comum;
- **Relação entre conceitos:** Criou-se uma associação entre elas;
- **Conceitos independentes:** Manteve-se cada classe em seu próprio pacote.

Segundo Rosito et al. (2008), a proposta de integração entre gerência de projetos e processos de desenvolvimento de software apresentado neste trabalho foi constituída de três pacotes: um para os conceitos de gerência de projetos, outro para os relacionados aos processos de desenvolvimento de software e, finalmente, um pacote comum que une os conceitos que ocorrem em ambos os modelos. Além disso, foi ressaltado no trabalho que alguns conceitos relacionados à gerência de projetos e que estão contidos nos processos de desenvolvimento de software do RUP foram propositadamente deslocados para o pacote de classes gerenciais do PMBOK com o objetivo de deixar mais explícita a classificação dos conceitos de gerência de projetos e do processo de desenvolvimento de software.

O modelo de integração desenvolvido denominado *Software Planning Integrated Model* (SPIM), representa os conceitos referentes à integração do PMBOK com o RUP. A figura 12 representa as etapas que constituíram o desenvolvimento deste modelo de integração.

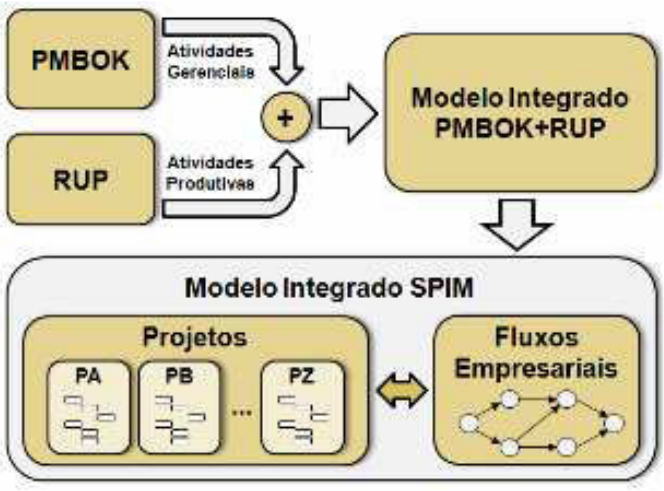


Figura 12: Etapas do desenvolvimento do modelo SPIM (ROSITO et al., 2008)

De acordo com Rosito et al. (2008), como o gerente de projetos pode não possuir todas as informações relevantes para o projeto, poderão ocorrer interações com outros departamentos da organização durante o planejamento de atividades de um projeto de software. Assim, o fluxo de atividades de um projeto de software pode interagir com os demais fluxos de atividades da organização denominados fluxos empresariais. Ambos os fluxos de trabalho são executados em paralelo, possuem recursos próprios e podem influenciar nos prazos das atividades e custos do projeto de software. As ferramentas atuais mais utilizadas não apresentam uma solução que permita o planejamento integrado de atividades gerenciais e de produção e esta carência pode resultar em distorções no planejamento de projetos pela desconsideração das dependências entre as atividades dos diferentes fluxos de trabalho.

A solução proposta no trabalho de Rosito et al. (2008) permite que cada instância de um fluxo empresarial seja registrada como correspondente a uma atividade gerencial de apoio ao projeto de desenvolvimento de software. Desta forma, cada atividade gerencial de apoio pode ter claramente definida as suas relações de dependência com outras atividades do projeto. Isto permite a integração com uma ferramenta de *workflow*, de maneira que este último seja responsável por atualizar os prazos das atividades de apoio ao projeto.

Rosito et al. (2008) desenvolveram um protótipo denominado *Software Planning Integrated Tool* (SPIT) para demonstrar os conceitos propostos pelo modelo integrado SPIM, fornecendo, desta forma, o suporte necessário para trabalhar com diferentes fluxos de trabalho, variáveis de projeto, responsabilidades e tarefas. O SPIT apresentado na figura 13, foi desenvolvido na linguagem C#.Net e atua como um *Add-in* para a ferramenta de gerenciamento de projetos *Microsoft Office Project*. As informações necessárias para as

validações dos conceitos propostos pelo modelo integrado SPIM estão contidas em campos customizados desta ferramenta comercial.

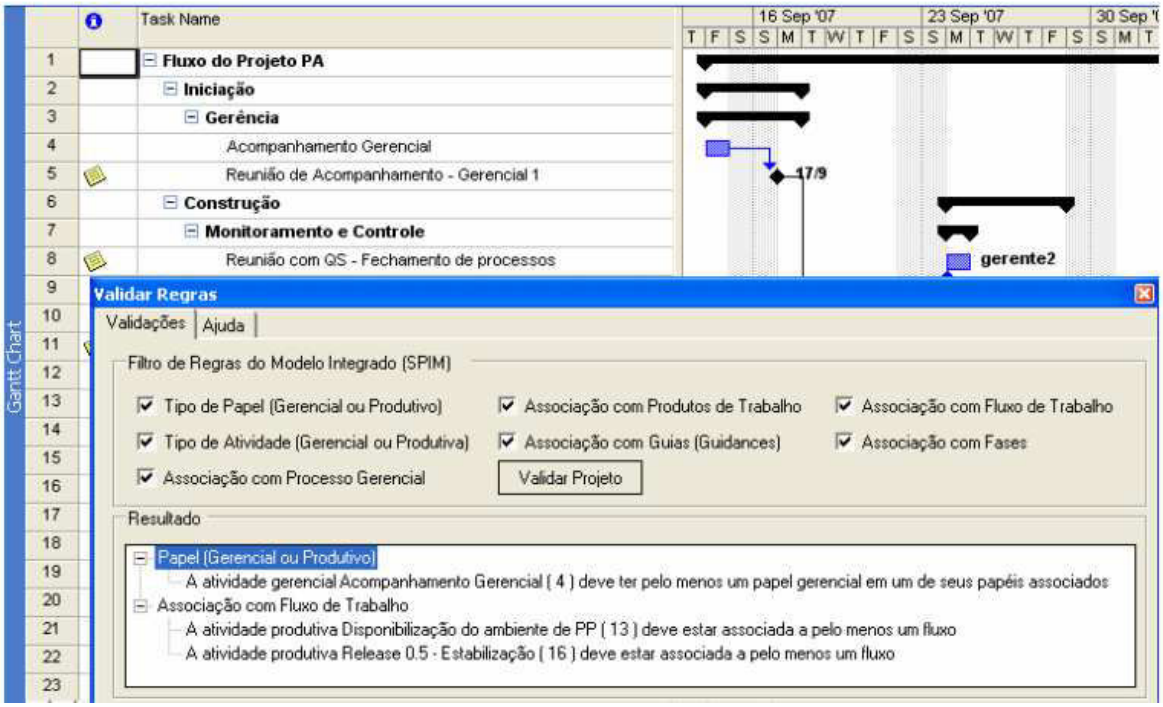


Figura 13: Protótipo do *Software Planning Integrated Tool* (ROSITO et al., 2008)

Este trabalho apresentou uma proposta para a integração dos principais conceitos sobre gerência de projetos e processos de desenvolvimento de software. Inicialmente, identificou-se a importância das atividades de gerência de projetos durante um projeto de desenvolvimento de software. A falta de uma metodologia de gerenciamento de projetos, bem como o crescimento da complexidade e do volume dos projetos em uma organização, contribui para o aumento das dificuldades relacionadas ao desenvolvimento de projetos (ROSITO et al., 2008). Em seguida, observou-se a carência existente no quesito de gerência de projetos nos processos de desenvolvimento de software atuais. Após uma análise individual de cada metamodelo base, foram apresentadas as principais colaborações ao metamodelo.

Finalmente, Rosito et al. (2008) apresentaram o protótipo que implementa o modelo de integração SPIM objetivando fornecer suporte ao gerente de projetos para lidar com diferentes fluxos de trabalho, variáveis de projeto, responsabilidades e tarefas. Mais uma vez, o objetivo é facilitar o trabalho do gerente de projetos, que tem de levar em conta cada vez mais elementos em suas decisões e com cada vez menos tempo.

4. TRABALHO PROPOSTO

Neste capítulo será apresentado o trabalho proposto pela presente dissertação que consiste na definição de um modelo abstrato de gerência do processo de software para metodologias ágeis. Inicialmente serão apresentados os principais metamodelos de metodologias ágeis e gerência de projetos encontrados na literatura, seguidos da apresentação do metamodelo definido. Em seguida, será dada uma visão geral da arquitetura do ambiente pró-ativo para gerência de software em metodologias ágeis assim como também um detalhamento dos agentes e da arquitetura que o compõem.

4.1. Introdução

Gerentes de projeto de software estão cada vez mais conscientes da importância de utilização de uma metodologia de desenvolvimento de software para garantir a qualidade do produto final. A crescente preocupação relativa ao desenvolvimento de software pode ser observada devido à adoção de práticas de engenharia de software pelas empresas. Um segmento crescente da engenharia de software vem defendendo a adoção de processos mais simplificados conhecidos como métodos ágeis, que visam a desburocratização das atividades associadas ao desenvolvimento. Os métodos ágeis têm despertado um grande interesse entre a comunidade de desenvolvimento de sistemas e acredita-se que devido a esta demanda, uma considerável quantidade de métodos apresentando características ágeis têm surgido nos últimos anos.

4.2. Metamodelos de metodologias ágeis estudados

Baseado em todas as características e princípios das metodologias ágeis expostos no referencial teórico deste trabalho, foi possível propor alguns metamodelos simples para as principais metodologias ágeis estudadas. Estes metamodelos serão apresentados a seguir e servirão de base para a criação do metamodelo integrado que será apresentado mais adiante neste trabalho. Foram selecionadas três metodologias ágeis como alvo deste estudo: XP, *Scrum* e FDD. A seleção destas metodologias se deu em função de existirem vários estudos indicando que elas se complementam. Enquanto que XP foca no desenvolvimento de software, *Scrum* é mais voltada para o gerenciamento do projeto e FDD orientada a modelagem.

4.2.1. Metamodelo da metodologia XP

A metodologia XP é uma abordagem voltada para o desenvolvimento de software. Ela foi criada com o objetivo de entregar o software que o cliente precisa quando ele precisa. XP

encoraja os desenvolvedores a se adaptarem a mudanças no escopo do projeto, mesmo que tardiamente no ciclo de produção do software (BECK; ANDRES, 2004).

Na figura 14 será apresentado um metamodelo de classes com os principais conceitos da metodologia XP, bem como o relacionamento entre eles:

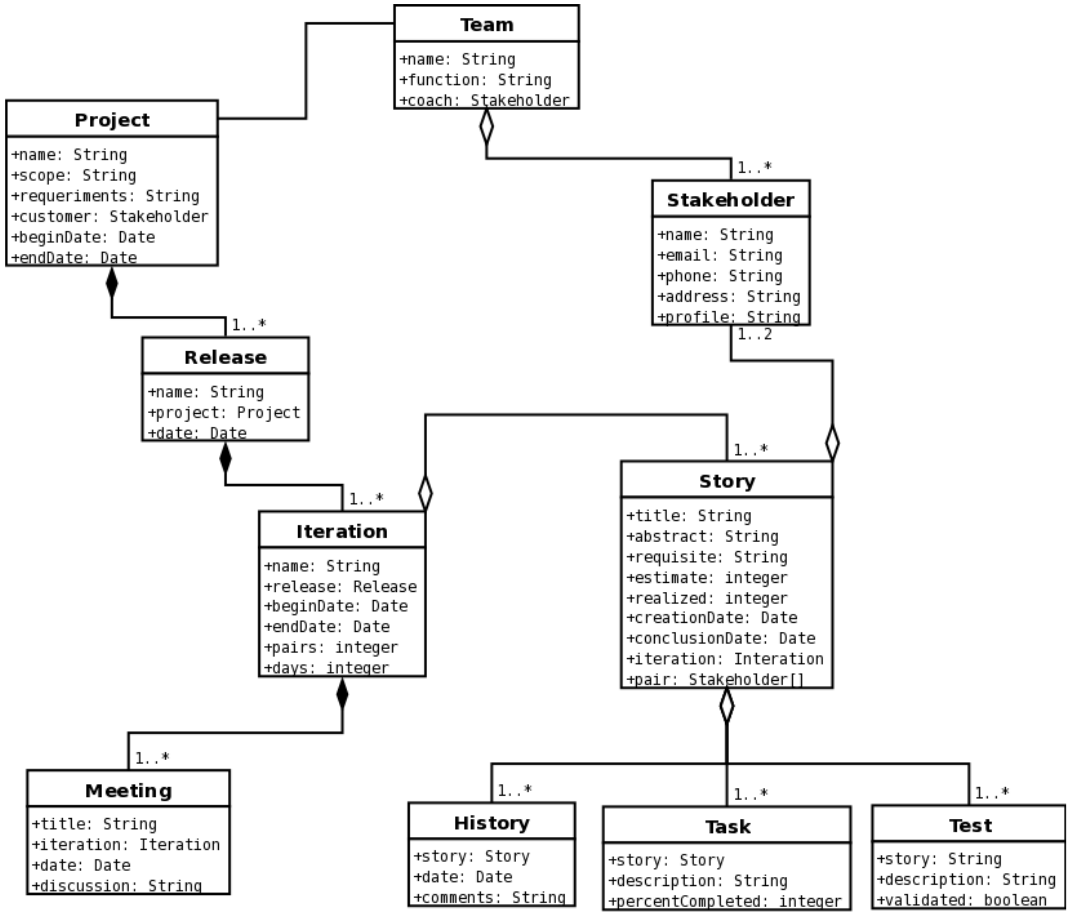


Figura 14: Metamodelo da metodologia XP (Desenvolvido pelo autor)

Segue abaixo uma descrição de cada uma das classes apresentadas no metamodelo:

- ***Project:*** O metamodelo suporta múltiplos projetos. Cada projeto tem seu cliente, além de informações gerenciais como nome, escopo e requisitos sucintos.
- ***Stakeholder*****:** São os membros do projeto. Podem ser cliente, gerente ou desenvolvedor. Possui alguns dados básicos de contato como telefone e email, dentre outros atributos.
- ***Team:*** Trata-se da equipe de um determinado projeto formada por *stakeholders.* A equipe possui um líder e tem determinada função dentro do projeto.
- ***Release:*** São os diferentes *releases* de um projeto. Possuem informações essenciais como nome e data de lançamento.
- ***Iteration:*** São as diferentes iterações de um *release.* Cada iteração possui um nome, data de início e fim, quantidade de pares alocados e dias de trabalho.

- ***Story:*** São os requisitos do sistema. Cada *story* possui informações como seu título, resumo, estimativa, tempo realizado, data de criação, data de conclusão e par de programadores. Além disso, possui alguns objetos agregados como *History, Tasks* e *Tests,* vistos abaixo.
- ***History:*** Armazena o histórico de decisões de uma *story*, através de informações como a data e um comentário.
- ***Tasks:*** São as diferentes tarefas nas quais uma *story* é dividida. Cada tarefa possui uma descrição e o percentual de conclusão.
- ***Tests:*** São os testes de aceitação da *story.* Provê uma descrição dos testes e se ele já está validado ou não.
- ***Meeting:*** Armazena os *Stand Up Meetings*, que são as rápidas reuniões matinais realizadas antes de se iniciar um dia de trabalho.

Na metodologia XP o desenvolvimento iterativo adiciona agilidade ao processo. A duração das iterações permite avaliar o progresso de um projeto e realizar o seu planejamento de forma simples e viável. Por isso, cada projeto inclui as iterações necessárias para que o trabalho selecionado maximize o valor do negócio.

4.2.2. Metamodelo da metodologia *Scrum*

A metodologia *Scrum,* conforme já abordado no referencial teórico deste trabalho, é uma metodologia ágil voltada ao gerenciamento do processo de desenvolvimento de software baseada em princípios como flexibilidade, adaptabilidade e produtividade. Seu foco é o desenvolvimento de um trabalho de qualidade em um ambiente suscetível à constantes mudanças (PAETSCH et al., 2003).

A metodologia *Scrum* foi modelada para ser adaptável a mudanças durante todas as fases do projeto. Ela provê mecanismos de controle para planejar uma entrega de produto e variáveis de gerenciamento para monitorá-lo, conforme pode ser visto na figura 15:

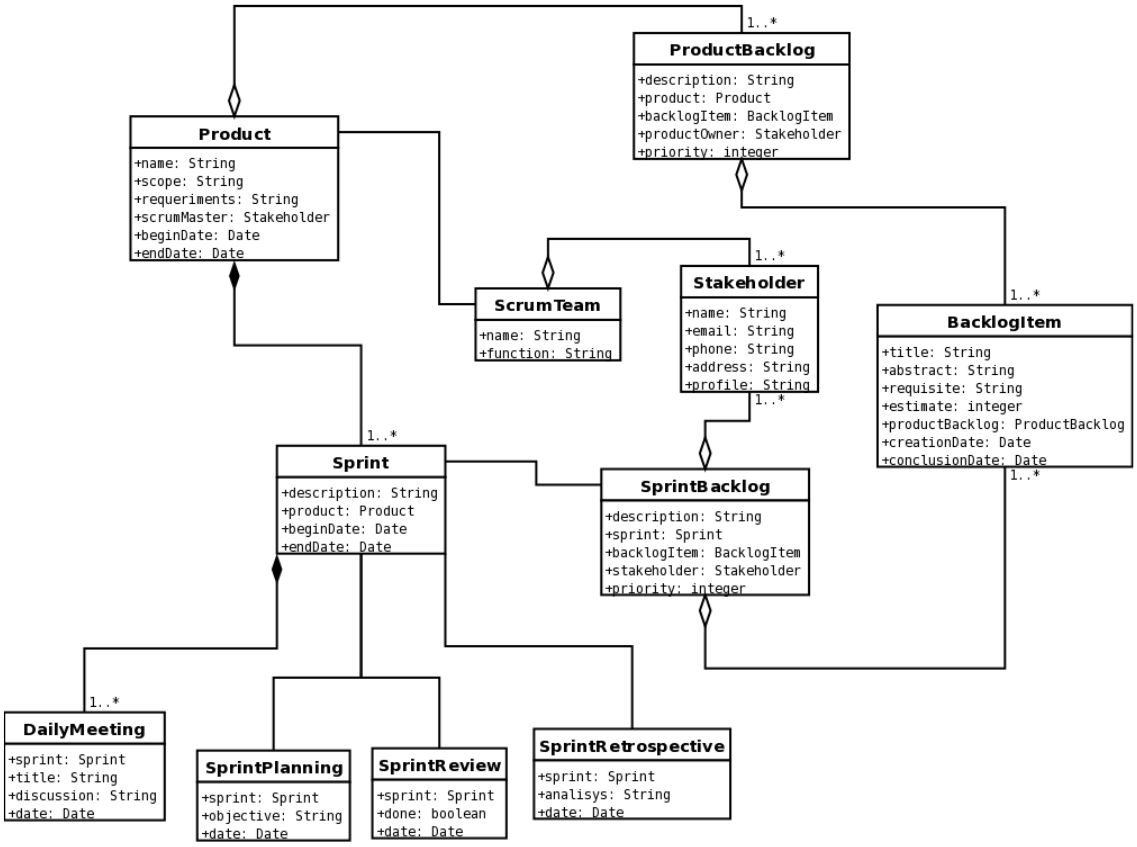


Figura 15: Metamodelo da metodologia *Scrum* (Desenvolvido pelo autor)

A seguir serão detalhadas as classes apresentadas no metamodelo:

- ***Product:*** É a classe principal do *Scrum*. Armazena o nome, escopo, requisitos gerais, data de início e fim e o *ScrumMaster* do produto final.
- ***Stakeholder:*** São as pessoas do processo *Scrum* tais como o *ScrumMaster*, *ProductOwner* e os próprios desenvolvedores. Possui alguns dados básicos como nome e email, dentre outros atributos.
- ***ScrumTeam:*** Trata-se da equipe de desenvolvimento do *Scrum* formada por *stakeholders* que são incentivadas a trabalharem cooperativamente para serem multidisciplinares.
- ***ProductBacklog:*** É a lista de todas funcionalidades requeridas para o produto com a devida ordem de prioridade estipulada pelo *ProductOwner*.
- ***BacklogItem:*** Armazena os detalhes de cada item da *ProductBacklog* tais como título, requisitos, estimativa de tempo, data de criação e conclusão.
- ***SprintBacklog:*** Define quais *BacklogItems* serão resolvidos no respectivo *Sprint* com uma ordem de prioridade. Além disso, o *SprintBacklog* indica quais os *stakeholders* estarão envolvidos neste trabalho.
- ***Sprint:*** Corresponde a uma iteração de entrega de um *SprintBacklog*. Tem uma

breve descrição assim como uma data de início e de fim.

- ***DailyMeeting:*** Reunião diária para troca de experiências dentro da equipe de desenvolvimento *Scrum.*
- ***SprintPlanning:*** Reunião de planejamento do *Sprint* onde são registrados os principais objetivos do respectivo *Sprint*.
- ***SprintReview:*** Reunião de fechamento de um *Sprint* onde o *ProductOwner* revisa cada uma das funcionalidades ditas como concluídas no respectivo *Sprint*. São armazenadas as situações de cada uma destas funcionalidades (aprovada/reprovada).
- ***SprintRetrospective:*** Tem a função de armazenar uma análise crítica feita sobre o último *Sprint* entregue.

Por fim, a *Scrum* é um método que possibilita uma gestão constante e muito próxima. Isso significa que o gestor consegue estar sempre monitorando as equipes de desenvolvimento e se certificando de que as práticas da metodologia estão sendo postas em prática como deveriam.

4.2.3. Metamodelo da metodologia FDD

A metodologia FDD é voltada para o cliente e orientada à modelagem. Ela oferece um conjunto coeso de princípios e práticas tanto para a gestão de projetos quanto para a engenharia de software. Consiste de cinco atividades básicas e possui métricas para marcar o progresso dessas atividades (PALMER; FELSING, 2002).

Nessa metodologia as entregas são incrementais. É baseada em funcionalidades que devem ser pequenas o suficiente para serem implementadas em no máximo uma iteração. A figura 16 apresenta um metamodelo de classes com os principais conceitos da metodologia FDD bem como o relacionamento entre eles:

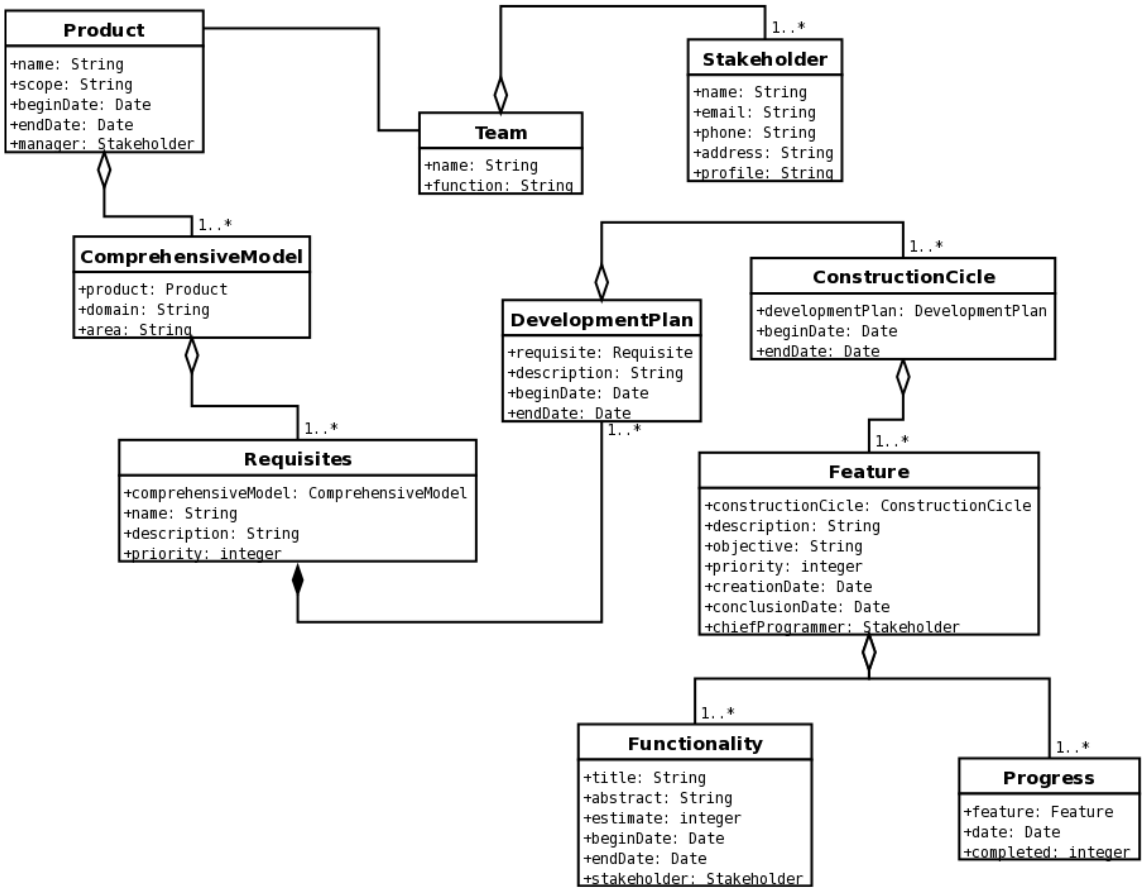


Figura 16: Metamodelo da metodologia FDD (Desenvolvido pelo autor)

A seguir serão detalhadas as classes apresentadas no metamodelo FDD:

- ***Product:*** É a classe principal que armazena o nome, escopo, data de início, data de fim e o nome do gerente do produto.
- ***Team:*** Trata-se da equipe envolvida no processo FDD.
- ***Stakeholder:*** São as pessoas do processo FDD tais como gerentes de projeto, especialistas de domínio, gerentes de desenvolvimento, programadores-chefes e desenvolvedores.
- ***ComprehensiveModel:*** É o modelo gerado que é suficientemente abrangente, mas não detalhado. O objetivo é ter uma definição a priori do que será feito, para guiar a equipe durante a fase de construção.
- ***Requisites:*** De posse do modelo básico criado, gera-se uma lista de requisitos. Trata-se de uma decomposição funcional do domínio do negócio.
- ***DevelopmentPlan:*** Com a lista de requisitos e o modelo, deve-se planejar a ordem na qual as funcionalidades serão implementadas.
- ***ConstructionCicle:*** Ciclo de construção. Seguindo o que foi definido no *DevelopmentPlan*, são eleitas as *Features* a serem desenvolvidas.
- ***Feature:*** Corresponde a um pacote de trabalho, sob a responsabilidade de um

programador-chefe. Tem uma prioridade estabelecida assim como também uma data de criação e uma data de conclusão.

- ***Functionality:*** Trata-se de uma *Feature* decomposta em funcionalidades mais simples que possam ser desenvolvidas o mais rápido possível.
- ***Progress:*** Armazena as informações de evolução daquela *Feature.*

A metodologia FDD é um método ágil altamente adaptativo que produz resultados frequentes, tangíveis e funcionais. Além disso, oferece vantagens dos métodos prescritivos pois implementa o conceito importante de planejamento, mas sem os exageros de documentação e controle. Também oferece vantagens dos métodos extremamente ágeis, onde a preocupação maior é com a produção de código, mas toma o cuidado de planejar o suficiente e controlar o andamento do projeto.

4.2.4. Comparação entre os metamodelos estudados

As três metodologias analisadas possuem um objetivo comum: conferir agilidade ao processo de desenvolvimento de software ganhando eficiência. A seguir será feita uma comparação entre os metamodelos estudados. Na tabela 6 serão listadas algumas características e como as metodologias se adaptam a elas, de acordo com a seguinte legenda:

- 0: Ausência desta característica;
- I: Acontece com frequência razoável;
- II: Acontece com alta frequência.

Tabela 6: Análise comparativa entre XP, *Scrum* e FDD (Desenvolvido pelo autor)

Características/Metodologia	**XP**	***Scrum***	**FDD**
Equipes pequenas	I	II	0
Ênfase no modelo	0	0	II
Ênfase gerencial	0	II	I
Ênfase no desenvolvimento	II	I	I
Interação cliente/desenvolvedor	II	II	I
Gerenciamento de risco	0	I	0
Projetos grandes/complexos	0	0	I
Stand-up meetings	II	II	0
Especialistas em documentação	I	0	II
Código coletivo	II	I	0

A seguir, serão detalhadas cada uma destas características:

- **Equipes pequenas:** Em XP a comunicação pessoal é um dos princípios da metodologia e devido a isso é melhor colocado em prática em equipes pequenas. Em *Scrum* existem poucos papéis (*ScrumMaster*, *ProductOwner* e membro da equipe) que não podem ser acumulados por um mesmo membro. Em função do FDD definir muitos papéis entre os seus membros, eles não podem ser exercidos pela mesma pessoa
- **Ênfase no modelo:** Em XP, a ênfase é na codificação/manutenção do código. Em *Scrum* a ênfase é nas interações entre os indivíduos. FDD é a única destas três metodologias que possui um foco na arquitetura ou modelagem do sistema. Nesta metodologia, a arquitetura é pensada desde a primeira fase do ciclo iterativo e passa por inspeções durante todas as outras fases.
- **Ênfase gerencial:** XP é uma metodologia orientada a processos de desenvolvimento. A *Scrum* é uma metodologia focada em processos gerenciais que incentivam agilidade em equipe quando comparado a outras metodologias ágeis. FDD possui certa ênfase gerencial mas não é o seu foco.
- **Ênfase no desenvolvimento:** XP é a metodologia que mais foca nos processos de desenvolvimento com práticas específicas, como por exemplo, a programação em pares. *Scrum* e FDD melhoram o processo de desenvolvimento mas isso não é a ênfase de ambas as metodologias.
- **Interação cliente/desenvolvedor:** Na XP o cliente deve estar presente a todo momento não apenas para ajudar a equipe de desenvolvimento mas também para ser parte dela. *Scrum* aplica intensivamente esta característica, visto que o *ProductOwner* faz o papel do cliente durante a *Sprint* e suas cerimônias. Em FDD o cliente deve estar sempre presente nas reuniões de planejamento e principalmente durante a primeira fase de modelagem do ciclo. Entretanto, como os programadores-chefe não são propriamente desenvolvedores, o contato entre o cliente e os desenvolvedores é realizado indiretamente.
- **Projetos grandes e complexos:** XP foi idealizado para equipes pequenas com poder de decisão própria. *Scrum* foi primeiramente pensada para controlar grupos pequenos. No entanto, pode ser escalável para trabalhar com grupos de 50 ou 60 pessoas dividindo a equipe em grupos menores executando *Scrum* de *Scrums.* FDD pode ser utilizada em projetos grandes sem necessitar repartir os membros em pequenos grupos de FDD devido sua origem ter sido em um projeto de mais de 50 pessoas.

- **Gerenciamento de riscos:** Como a ênfase da XP é no processo de desenvolvimento e não no processo de gerência, não há uma base para tratar riscos. Na *Scrum*, a cada *DailyScrum*, a equipe se reúne e conversa sobre o que foi feito desde a última reunião podendo discutir nestes momentos possíveis riscos que possam atrapalhar o trabalho de desenvolvimento. Em FDD não existe um documento de gerenciamento de riscos assim como também não há impedimentos levantados pela equipe.
- ***Stand-up meetings*:** Esse estilo de reunião proposto inicialmente na metodologia XP é utilizado também por *Scrum.* Em FDD, apesar da comunicação interna ser estimulada, não existem traços específicos deste estilo de reunião.
- **Especialista em documentação:** Em XP, existe um especialista em documentação responsabilizando-se sobre documentos de requisitos, casos de uso, mudanças de requisitos, entre outros quando estes promovem algum valor para o cliente. No *Scrum* não há um redator técnico. Como FDD dá muita atenção à documentação produzida durante a modelagem, uma pessoa é responsável por estes documentos.
- **Código coletivo:** XP possui a coletividade do código como um princípio. *Scrum* quando utilizada em projetos de software, não prega abertamente a coletividade do código. FDD designa alguns membros específicos como proprietários de classe.

4.3. Metamodelo de gerência de projetos estudado

No referencial teórico sobre gerência de projetos, o PMBOK foi citado como o principal método utilizado para o processo de gerência. Da mesma forma como foi feito para as metodologias ágeis e com o intuito de propor um metamodelo para gerência de projetos de software em metodologias ágeis, foi proposto um metamodelo simples do PMBOK conforme será detalhado a seguir.

4.3.1. Metamodelo de gerência do PMBOK

O PMBOK formaliza diversos conceitos em gerenciamento de projetos, como a própria definição de projeto e do seu ciclo de vida. Define que a iniciação, o planejamento, a execução, o monitoramento e o encerramento são os cinco grupos de processos de gerência de projetos e descreve estes processos em nove áreas de conhecimento. Normalmente os conceitos do PMBOK são representados por textos descritivos. No entanto, a figura 17 apresenta um metamodelo de classes do PMBOK com seus principais conceitos bem como o relacionamento entre eles:

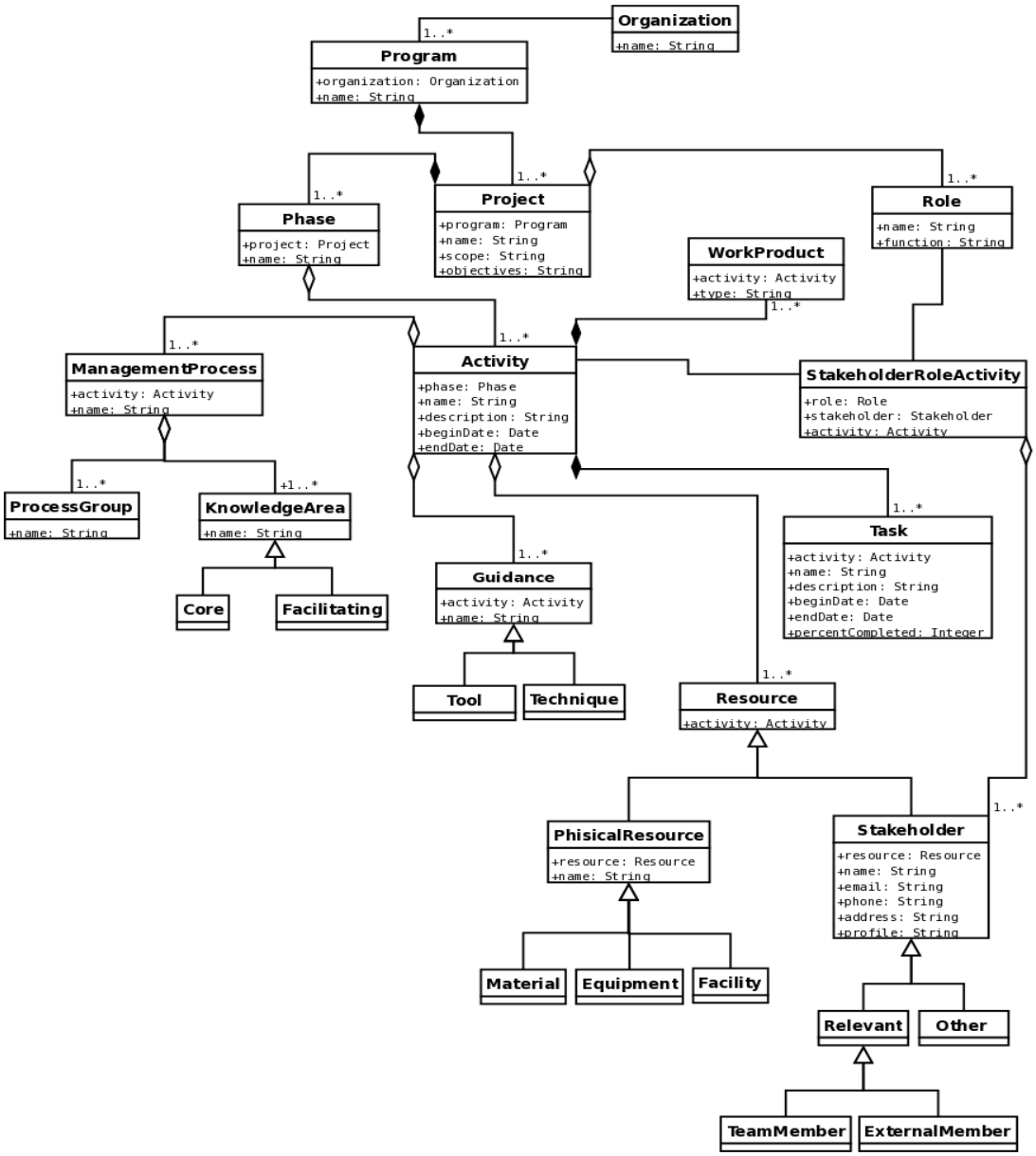


Figura 17: Metamodelo de gerência do PMBOK (Desenvolvido pelo autor)

Na sequência serão detalhadas as principais classes apresentadas no metamodelo:

- ***Organization:*** Classe que representa a empresa que se organiza por programas.
- ***Program:*** Os programas são um grupo de projetos designados a alcançar um objetivo estratégico.
- ***Project:*** Trata-se do projeto onde estão sendo aplicados os processos e as áreas de conhecimento do PMBOK.
- ***Phase:*** Geralmente as organizações dividem os projetos em várias fases visando um melhor controle gerencial.
- ***Activity:*** Armazena as atividades gerenciais do projeto em determinada fase. Cada *activity* possui informações tais como descrição, data de início e data de término.

Além disto, possui alguns objetos agregados como *ManagementProcess*, *Guidance, Task, Resource* e *WorkProduct* que serão detalhados abaixo.

- ***ManagementProcess:*** Descreve os processos gerenciais da atividade. Estes processos podem pertencer a grupos de processos (*ProcessGroup)* ou áreas de conhecimento (*KnowledgeArea*) que se subdividem em áreas centrais e de apoio.
- ***Guidance:*** Oferece guias que podem ser utilizados como ferramentas ou apoio técnico.
- ***Task:*** Objetiva dividir atividades em tarefas menores.
- ***Resource:*** Armazena os recursos necessários para um projeto. Estes recursos são divididos em recursos ativos (*Stakeholder)* e recursos inativos (*PhysicalResource*). A classe *Stakeholder* se divide em uma subclasse de pessoas relevantes (membros da equipe ou membros externos) e outra com as demais. A classe *PhysicalResource* se desmembra em materiais, equipamentos e facilitadores necessários para o projeto.
- ***Role:*** Representa os papéis importantes para um determinado projeto.
- ***StakeholderRoleActivity:*** Classe que designa uma atividade para ser resolvida por um ou mais *stakeholders* assumindo determinado papel no projeto.

O metamodelo do PMBOK contempla em sua estrutura as melhores práticas que podem ser aplicadas na maioria dos projetos. A gerência de projetos em PMBOK consiste da aplicação de conhecimentos, habilidades e técnicas para projetar atividades que visem atingir requisitos definidos. O principal objetivo disso é visualizar a gerência de projetos como um conjunto de processos encadeados e integrados.

4.4. Metamodelo para gerência de projetos em metodologias ágeis

Após estudo dos metamodelos das principais metodologias ágeis e de um metamodelo de gerência de projetos, definiu-se conforme a figura 18 a proposta do metamodelo para gerência de projetos em metodologias ágeis. Logo após a figura, serão explicadas as principais classes bem como os conceitos novos criados neste metamodelo.

Figura 18: Metamodelo para gerência de projetos em metodologias ágeis

No metamodelo proposto, a classe ***Organization*** representa a empresa que pode possuir vários programas (classe ***Program***). Ambos são do PMBOK. Os programas são grupos de projetos (classe ***Project)*** que têm o propósito de alcançar um objetivo estratégico. A classe *Project* representa a mesma classe *Project* do XP e do PMBOK e a classe *Product* do *Scrum* e do FDD.

A classe ***Phase*** relacionada ao projeto é integrante do PMBOK. Cada fase do projeto é

composta por atividades (classe ***Activity***) que possuem alguns objetos agregados como *ManagementProcess*, *Guidance*, *Resource, StakeholderRoleActivity* e *WorkProduct* todos integrantes do PMBOK. Cabe ressaltar que não está mais agregado a atividade a classe *Task,* originalmente presente no metamodelo do PMBOK pois ela passou a compor um *BacklogItem* que será detalhado mais a frente.

A classe *Activity* está relacionada com a classe ***Requisites*** que por sua vez se liga com a classe ***ComprehensiveModel***. Estas duas classes pertencem a metodologia FDD que procura atribuir uma maior concepção e planejamento a uma atividade que está sendo desenvolvida.

Na sequência aparece a classe ***ProjectBacklog*** que tem a mesma finalidade do *ProductBacklog* da metodologia *Scrum.* Neste caso somente o seu nome foi adaptado no metamodelo. Agregada ao *ProjectBacklog* está a classe ***BacklogItem*** que na XP se chama *Story* e no PMBOK se chama *Task.* Esta classe possui outras quatro agregadas. São as classes *History, Task e Test* pertencentes a XP e a classe *Functionality* pertencente a FDD.

A classe ***CycleBacklog*** tem o mesma finalidade da *SprintBacklog* do *Scrum*. Somente seu nome foi adequado ao metamodelo mas continua representando quais *BacklogItems* serão resolvidos na respectiva iteração com uma ordem de prioridade e *stakeholders* envolvidos neste trabalho.

Em seguida aparece a classe ***Cycle*** que é uma abstração das classes *Iteration* da XP e *Sprint* da *Scrum* e *ConstructionCycle* da FDD. Agregadas a ela existem as classes *Planning, Review* e *Retrospective* oriundas do *Scrum* e a classe *Meeting* que vem do XP e *Scrum*.

Entre as classes *Cycle* e *Project* aparece a classe ***Release*** representando o conceito de versões vindo da metodologia XP. Ainda no metamodelo proposto, aparece a classe ***Team*** que representa a equipe de trabalho do XP, *Scrum* e FDD e a classe ***Stakeholder*** que representa os recursos humanos utilizados em projetos. Esta classe existe em todas as metodologias inclusive no PMBOK. Porém neste ela é uma agregação de uma classe maior chamada ***Resource*** que se divide em *Stakeholder* e *PhysicalResource*.

Diante do que foi exposto na figura 18 e detalhado anteriormente, pode-se afirmar que o metamodelo proposto contempla um conjunto de características e conceitos das três metodologias ágeis de desenvolvimento de software e do metamodelo de gerência de projetos estudado, o que permite a criação de uma aplicação para melhor gerenciar projetos de software utilizando metodologias ágeis.

4.5. Visão geral do ambiente para gerência de software em metodologias ágeis

Com base no metamodelo apresentado e baseado em seus principais conceitos e

princípios, será dada uma visão geral do ambiente pró-ativo para gerência de software em metodologias ágeis que está sendo proposto. Diante disso, o metamodelo já foi projetado para ser modular o suficiente permitindo que seja instanciado em um determinado projeto, gerando um modelo concreto de acordo com as características da metodologia estabelecidas pelo seu gerente.

A partir deste modelo concreto, será gerado um modelo gerencial responsável por armazenar informações coletadas bem como dados resultantes do processo de rastreabilidade de artefatos de software, proposto por Dall'Oglio (2010). Neste trabalho, o autor procura melhorar a gestão da mudança de requisitos por meio da implementação de uma ferramenta *web* apoiada por agentes de software que controlam com precisão a informação de rastreabilidade e suportam de forma pró-ativa a gestão da mudança de requisitos e a análise de impactos durante todo o ciclo de desenvolvimento de software.

E, por fim, será desenvolvida uma aplicação que deverá primeiramente disponibilizar ao gerente do projeto uma interface para que ele possa cadastrar os recursos, as atividades, os ciclos de desenvolvimento e outras informações sobre o projeto. A partir destes dados cadastrados, será possível utilizá-los como referência para consolidar os dados coletados pelos agentes de software, dando informações gerenciais precisas ao gerente do projeto de acordo com as metas pré-estabelecidas.

Com base nesta proposta, a figura 19 pretende dar uma visão geral do trabalho que está sendo proposto.

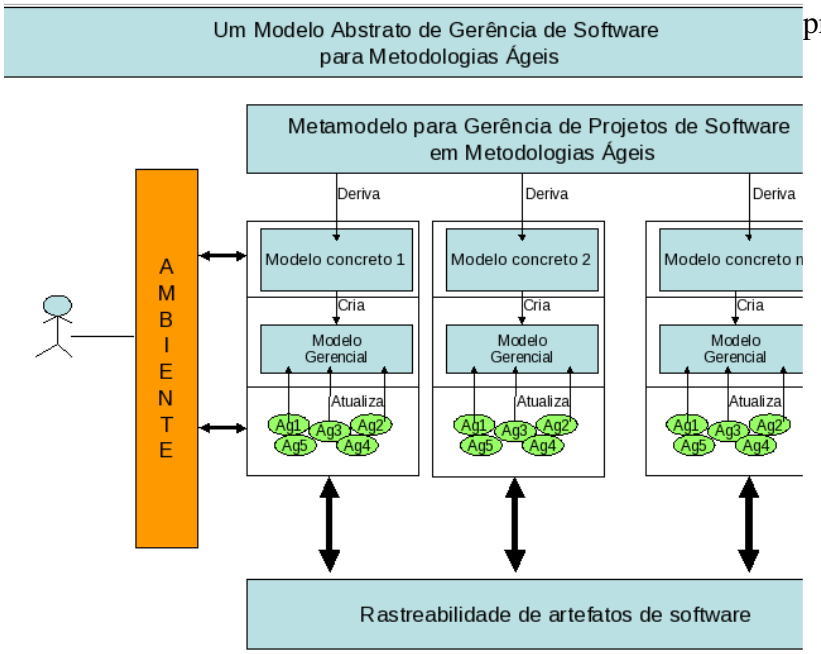


Figura 19: Visão geral do trabalho proposto

Diante da visão geral do ambiente pró-ativo para gerência de software em metodologias ágeis que está sendo proposto, será apresentada na sequência uma breve descrição dos principais objetivos do ambiente. Além disso, serão detalhados os agentes de software que irão monitorar o ambiente e a integração entre eles assim como também a arquitetura da aplicação proposta.

4.6. Objetivos do ambiente

O ambiente pró-ativo para a gerência de software em metodologias ágeis será composto por um conjunto de agentes responsáveis por atualizar informações no modelo conceitual e uma aplicação gerencial rodando sobre estas informações dando suporte ao gerente do projeto.

Para DeLoach e Wood (2001), ao longo dos últimos anos houve muitas tentativas em criar ferramentas e metodologias para o desenvolvimento de sistemas multi-agentes. Entretanto, a maioria delas focou muito mais em descrições abstratas e conceitos sobre arquiteturas específicas de agentes do que em uma metodologia de projeto. A metodologia MaSE, de outra forma, procura cobrir todo o ciclo de vida do desenvolvimento de um sistema multi-agente, desde a análise, o projeto até o desenvolvimento. Desta forma, a arquitetura de agentes necessária para o cumprimento dos objetivos do presente trabalho será desenvolvida na metodologia MaSE.

Segundo DeLoach e Wood (2001), a primeira fase da metodologia MaSE é a descoberta dos objetivos baseada na especificação inicial de um sistema. O resultado é um conjunto de metas estruturadas de forma hierárquica, como mostrado na figura 20. Uma vez definidos os objetivos do sistema, a probabilidade de mudança neles é menor nas atividades subsequentes.

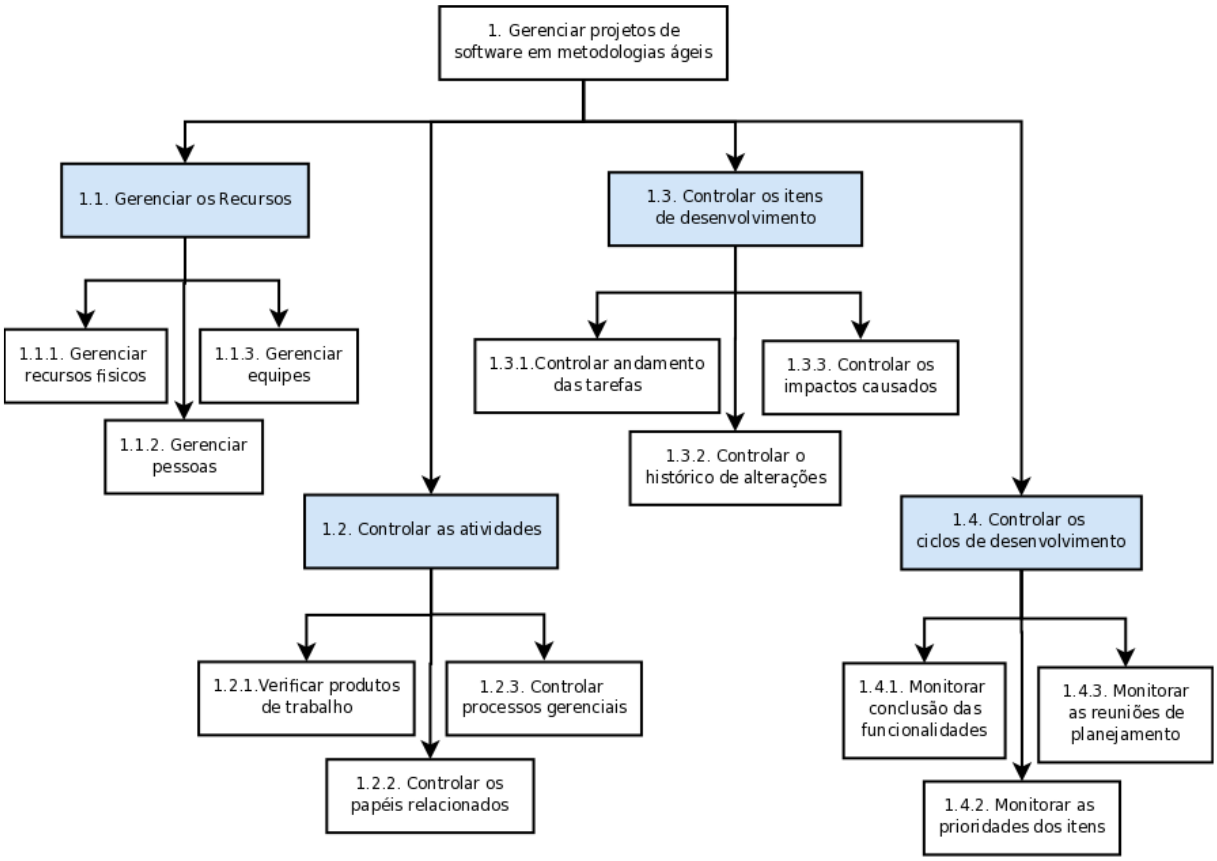


Figura 20: Objetivos do ambiente

O objetivo principal do ambiente é gerenciar projetos de software em metodologias ágeis. Para tal, este objetivo deve ser desmembrado em quatro responsabilidades distintas e necessárias para o sucesso da aplicação da metodologia proposta, que são descritos a seguir:

- **Gerenciar os recursos:** Objetivo responsável por gerenciar os recursos necessários para um projeto, tais como recursos físicos, pessoas e equipes.
- **Controlar as atividades:** Objetivo responsável por controlar as atividades gerenciais do projeto em determinada fase como verificar produtos de trabalho, controlar a atribuição de papéis e processos gerenciais.
- **Controlar os itens de desenvolvimento:** Objetivo responsável por controlar o andamento das tarefas específicas de determinada atividade, ou seja, procura controlar o andamento destas tarefas, o histórico das alterações e efetua a medição dos impactos gerados.
- **Controlar os ciclos de desenvolvimento:** Objetivo responsável por confrontar as funcionalidades concluídas em determinado ciclo com as que haviam sido previstas. Além disso, permite monitorar a ocorrência das reuniões de planejamento e se os itens do respectivo ciclo estão sendo cumpridos dentro da prioridade estabelecida para eles.

4.7. Modelagem dos agentes

O ambiente será composto por um conjunto de agentes de software com a função de implementar os objetivos traçados. Diante disso, foram definidos quatro agentes responsáveis por atualizar informações no modelo conceitual: *ResourceManager*, *ActivityController*, *BacklogItemController* e *CycleController* que serão descritos abaixo:

- ***ResourceManager*:** O agente *ResourceManager* irá gerenciar os recursos do projeto tais como recursos físicos, pessoas e equipes. Será este agente que identificará caso alguma pessoa deixará de ser integrante da equipe ou se algum recurso físico não estiver contemplando as expectativas do projeto. Além disso, este agente poderá indicar o grau de relevância que determinado recurso tem no projeto bem como indicar as tarefas que estão fortemente dependentes deste recurso a fim de fornecer ao gerente um alerta para situações que podem vir a ser críticas.
- ***ActivityController*:** O agente *ActivityController* irá controlar as atividades gerenciais do projeto verificando se elas estão sendo cumpridas dentro do prazo previsto. Ele irá monitorar também os papéis atribuídos a cada atividade bem como as pessoas responsáveis. Neste ponto haverá uma comunicação entre os agentes *ResourceManager* e *ActivityController* a fim de monitorar uma possível perda de recurso relevante que impacte diretamente na conclusão de determinada atividade.
- ***BacklogItemController:*** O agente *BacklogItemController* irá controlar os itens de desenvolvimento verificando principalmente o andamento das tarefas frente às atividades a que ela pertence. Este agente terá uma integração com a aplicação de rastreabilidade de artefatos de software (DALL'OGLIO, 2010) utilizando um repositório de controle de versões para armazenar os *logs* de avanços obtidos no projeto a fim de possibilitar extrair informações gerenciais importantes para o gerente do projeto. Além disso, o agente *BacklogItemController* se comunicará com agente o *ResourceManager* a fim de monitorar os recursos de equipe alocados para determinada tarefa.
- ***CycleController:*** O agente *CycleController* irá controlar os ciclos de desenvolvimento. Baseado nos itens definidos para aquele ciclo ele irá confrontar o que foi planejado com o que foi realizado apresentando análises previsões ao gerente do projeto.

Além destes agentes, serão utilizados recursos externos tais como um repositório de

controle de versão para armazenar as dados processados pelo agente *BacklogItemContoller* com o intuito de interagir com o software de rastreabilidade de artefatos (DALL'OGLIO, 2010) além de uma comunicação configurável com sistemas externos a fim de coletar informações a respeito de disponibilidade de recursos físicos e humanos para o projeto.

A segunda fase da metodologia MaSE é transformar esta hierarquia de objetivos em papéis. Os papéis definem as classes de agentes e representam os objetivos da aplicação durante a etapa de projeto. Assim, cada objetivo é garantido por um papel desempenhado por um agente, conforme figura 21:

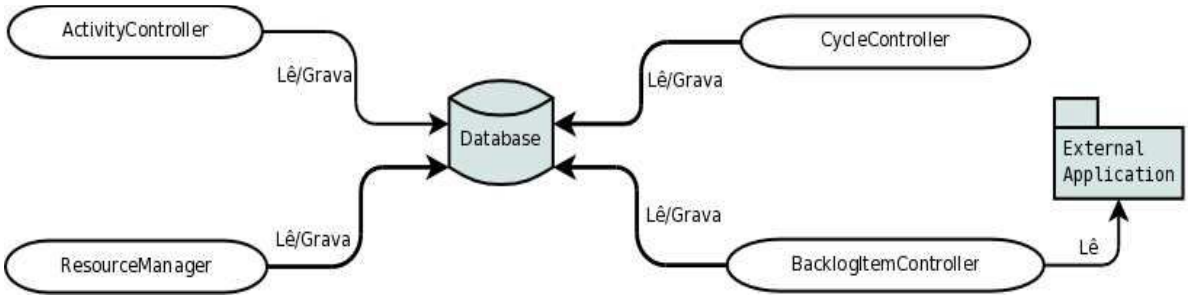


Figura 21: Papéis dos agentes

Para DeLoach e Wood (2001), em situações onde é possível combinar os objetivos similares que compartilham um grande grau de coesão, estes podem ser mapeados para o mesmo papel dentro do diagrama. Interfaces para recursos externos como bancos de dados e interfaces com outras aplicações, necessitam um papel em separado para lidar com esta situação a fim de encontrar um grau de relacionamento que possa ser compartilhado entre os dois papéis.

4.8. Arquitetura do ambiente

A ambiente proposto será apoiado por quatro agentes de software. Além disso, haverá uma aplicação desenvolvida para *Web* que disponibilizará uma interface para o gerente do projeto configurar as características desejáveis que serão utilizadas em determinado projeto de software. A aplicação irá comunicar-se somente com os agentes, que provêem todos os dados necessários para efetuar a gerência de projetos de software em metodologias ágeis. A figura 22 apresenta os principais componentes da arquitetura do ambiente proposto:

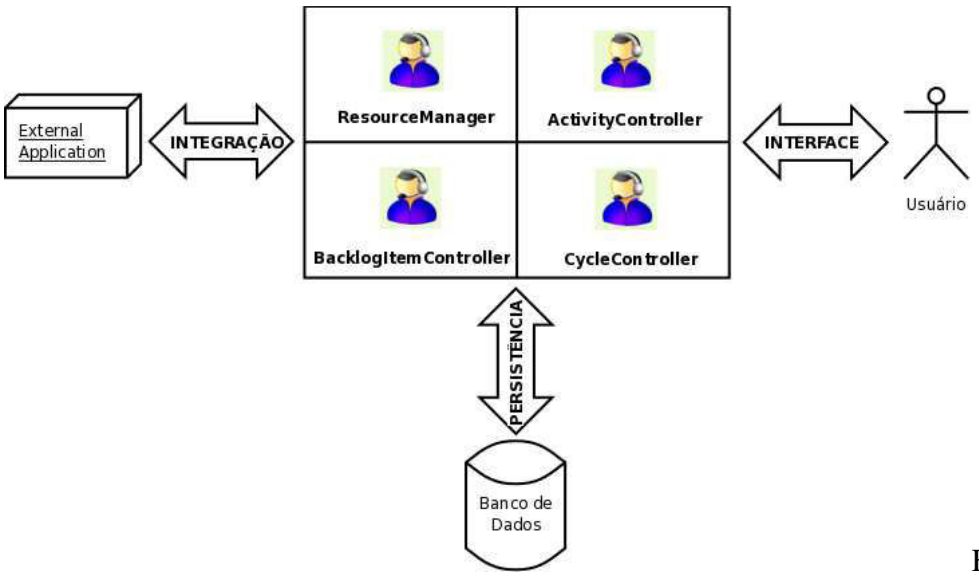


Figura 22: Arquitetura do ambiente

Abaixo serão detalhados os principais componentes da arquitetura do ambiente proposto:

- **Agentes:** O ambiente é composto por quatro agentes: *ResourceManager, ActivityController, BacklogItemController* e *CycleController,* que são responsáveis pelos objetivos definidos anteriormente;
- **Integração:** Representa a camada lógica responsável pela conversa entre os agentes e as aplicações externas;
- ***External Application:*** Representa aplicações externas que podem ser consultadas pelos agentes para coletar informações para o modelo gerencial;
- **Persistência:** Representa a camada lógica responsável por recuperar e armazenar informações no banco de dados;
- **Banco de dados:** Representa a base de dados da aplicação, que irá armazenar os dados coletados pelos agentes de software.
- **Interface:** Representa a interface que será utilizada pelo gestor planejar e acompanhar o andamento do projeto;
- **Usuário:** Representa o gerente do projeto que poderá obter informações gerenciais atualizadas.

Para o desenvolvimento da aplicação proposta, foram utilizadas as tecnologias listadas a seguir. Deve-se citar que todas são tecnologias livres e independentes de plataforma:

- **Apache:** O Apache[3] é um servidor de páginas *Web* mundialmente reconhecido por sua qualidade e segurança.

3 http://www.apache.org

- **PHP:** O PHP[4] é uma linguagem de programação de última geração voltada originalmente para a *Web* com grandes recursos de orientação a objetos.
- **PostgreSQL:** O PostgreSQL[5] é um banco de dados relacional robusto, com um grande poder de armazenamento e segue o padrão SQL (*Structured Query Language*).

4.9. Modelo entidade-relacionamento

Depois de consolidado o metamodelo abstrato para a gerência de software para metodologias ágeis representado em UML e também a arquitetura do ambiente com suas interfaces de comunicação e agentes, pode-se realizar seu mapeamento para uma estrutura relacional por meio da aplicação de certos padrões ORM (*Object-Relational Mapping*). Os padrões utilizados neste trabalho para o mapeamento objeto-relacional foram retirados do catálogo de *Design Patterns* para uso em aplicações corporativas de Fowler (2002).

Dentre os padrões utilizados neste trabalho para a criação do modelo relacional, pode-se citar o padrão *Class Table Inheritance*, utilizado para mapear os relacionamentos de herança, como os encontrados entre as classes *Resource*, *Stakeholder* e *PhysicalResource*, o padrão *Foreign Key Mapping*, para mapear os relacionamentos de composição, como os encontrados entre as classes *Project* e *Phase.* Já para mapear relacionamentos de agregação, como o encontrado entre as classes *Activity* e *Resource,* foi utilizado o padrão *Association Table Mapping*.

Alguns relacionamentos de herança, como os encontrados a partir das classes *Activity* e *Requisites* puderam ser mapeados por meio do padrão *Single Table Inheritance*, tendo uma tabela de apoio para representar o tipo de objeto armazenado na estrutura.

A figura 23 representa o modelo entidade-relacionamento consolidado que foi utilizado como base para a criação da aplicação que será detalhada na sequência.

4 http://www.php.net
5 http://www.postgresql.org

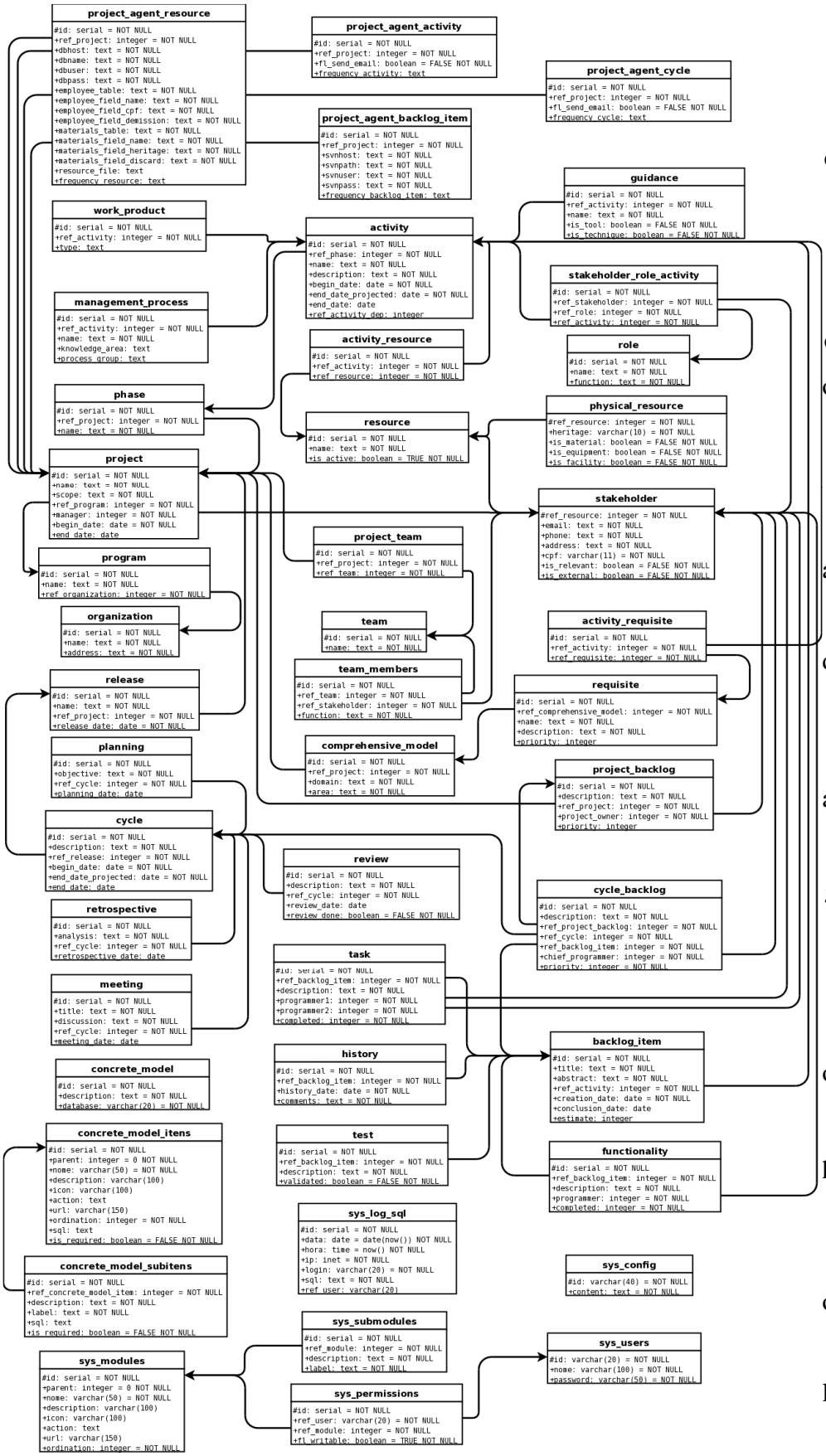


Figura 23: Modelo entidade-relacionamento consolidado

O modelo entidade-relacionamento consolidou todas as relações mapeadas nos metamodelos de metodologias ágeis e de gerência de projetos estudados. No entanto, para que a aplicação fosse desenvolvida e permitisse toda a dinamicidade proposta, foram adicionadas mais três tabelas que são responsáveis por

gerenciar os modelos concretos instanciados e armazenar as opções possíveis de serem selecionadas. Estas três tabelas estão identificadas pelos nomes:

- ***concrete_model:*** Armazena os modelos concretos instanciados a partir do modelo abstrato;
- ***concrete_model_itens:*** Armazena os itens possíveis de serem selecionados para a instância dos modelos abstratos;
- ***concrete_model_subitens:*** Armazena as características específicas de cada um dos itens possíveis de serem selecionados.

Além disso, existem outras seis tabelas que foram definidas com o prefixo "*sys"* e pertencem a estrutura do *framework* utilizado para o desenvolvimento. São elas:

- ***sys_users***: Armazena os usuários que possuem acesso a aplicação;
- ***sys_modules***: Armazena os módulos ou menus que aquele modelo concreto possui;
- ***sys_permissions***: Armazena os módulos que um respectivo usuário tem habilitado;
- ***sys_submodules***: Armazena as abas que cada módulo da aplicação possui habilitado;
- ***sys_config***: Armazena configurações parametrizáveis da aplicação;
- ***sys_log_sql***: Armazena os *logs* de operação da aplicação.

Para armazenar as configurações dos agentes de software do modelo foram definidas quatro tabelas adicionais responsáveis por armazenar os parâmetros de configuração de cada agente de software:

- ***project_agent_resource***: Armazena os parâmetros configurados para o agente de recursos tais como a identificação do *host*, base de dados, usuário e senha e das tabelas e campos do sistema que será utilizado para sincronizar as informações do modelo ou o caminho do arquivo XML de sincronização;
- ***project_agent_backlog_item***: Armazena os parâmetros configurados para o agente de itens tais como a localização do repositório de controle de versões;
- ***project_agent_activity***: Armazena os parâmetros configurados para o agente de atividades;
- ***project_agent_cycle***: Armazena os parâmetros configurados para o agente de ciclos.

O restante das tabelas do modelo entidade-relacionamento consolidado são criadas baseadas na configuração definida pelo gerente do projeto no momento de instanciar o modelo abstrato. A interface para se fazer isso será detalhada na seção seguinte onde será possível visualizar nitidamente como o processo ocorre.

4.10. A aplicação desenvolvida

Apesar de vários conceitos da aplicação já terem sido apresentados nos tópicos anteriores, neste tópico serão demonstradas e detalhadas as demais funcionalidades da aplicação desenvolvida.

A figura 24 procura demonstrar a arquitetura da aplicação desenvolvida. Ela constitui-se de uma aplicação *Web* apoiada por agentes de software implementados como *Web Services* que são responsáveis em alimentar o modelo relacional.

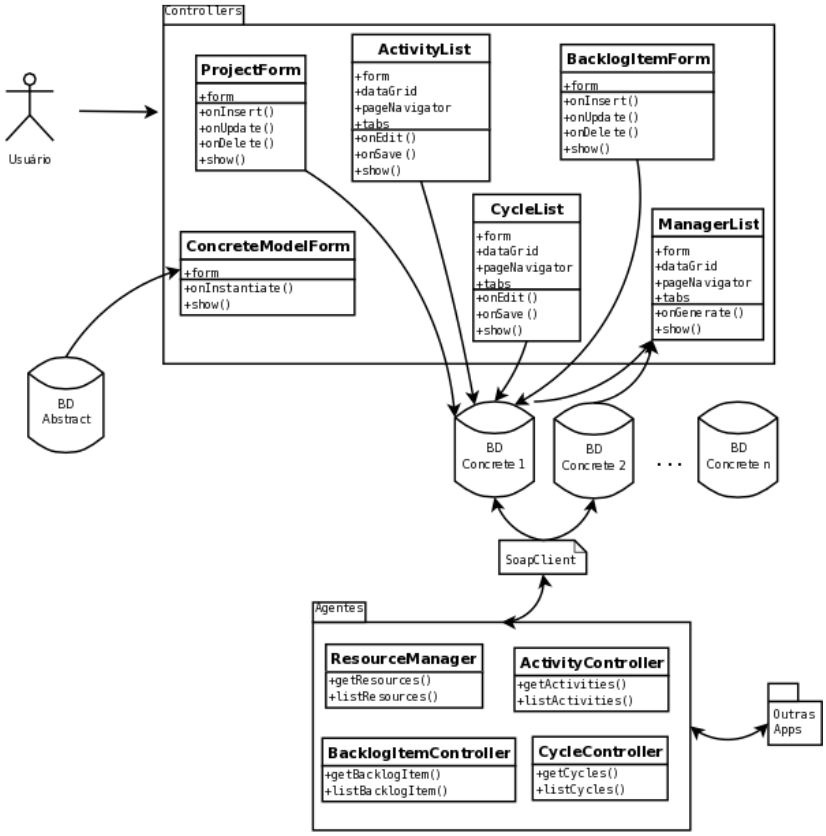


Figura 24: Arquitetura da aplicação

A aplicação é formada por um conjunto de classes de interface que implementam o padrão MVC (*Model View Controller*), tais como *ProjectForm, ConcreteModelForm, ActivityList, BacklogItemForm, CycleList, ManagerList,* dentre outras. Ao todo, são cerca de 20 classes de controle de interface.

A aplicação apresenta duas interfaces distintas. Uma interface que é utilizada para a gerência do modelo abstrato permitindo que o gerente do projeto instancie o modelo abstrato através do menu “Criar modelo concreto” podendo escolher as características que desejar que seu modelo concreto possua. A outra interface da aplicação tem como visão a interface que gerencia o modelo concreto permitindo administrar o projeto dentro das características escolhidas para o mesmo.

A figura 25 apresenta o menu principal da aplicação desenvolvida tendo como visão a

interface que gerencia o modelo abstrato permitindo instanciar modelos concretos que são definidos pelo próprio gerente do projeto no momento da criação:

Figura 25: Interface de gerência do modelo abstrato

A figura 26 apresenta o menu principal da aplicação desenvolvida tendo como visão a interface que gerencia o modelo concreto que permite o gerente do projeto as seguintes seções de administração do projeto:

- **Cadastros básicos:** Nesta seção são disponibilizadas as interfaces para cadastros de organizações, programas, recursos físicos e humanos que serão utilizados no cadastro do projeto. Após cadastrado o projeto é possível definir todos os requisitos, atividades, itens e ciclos de desenvolvimento.
- **Agentes:** Nesta seção é possível configurar quatro agentes de software que irão alimentar o modelo concreto instanciado.
- **Relatórios:** Permite extrair relatórios gerenciais configuráveis bem como visualizar o gráfico de *gantt* dinâmico que representa a situação atual do projeto.
- **Administração:** Gerência de usuários e permissões de acesso na aplicação.

A seguir, serão demonstradas algumas de suas principais funcionalidades.

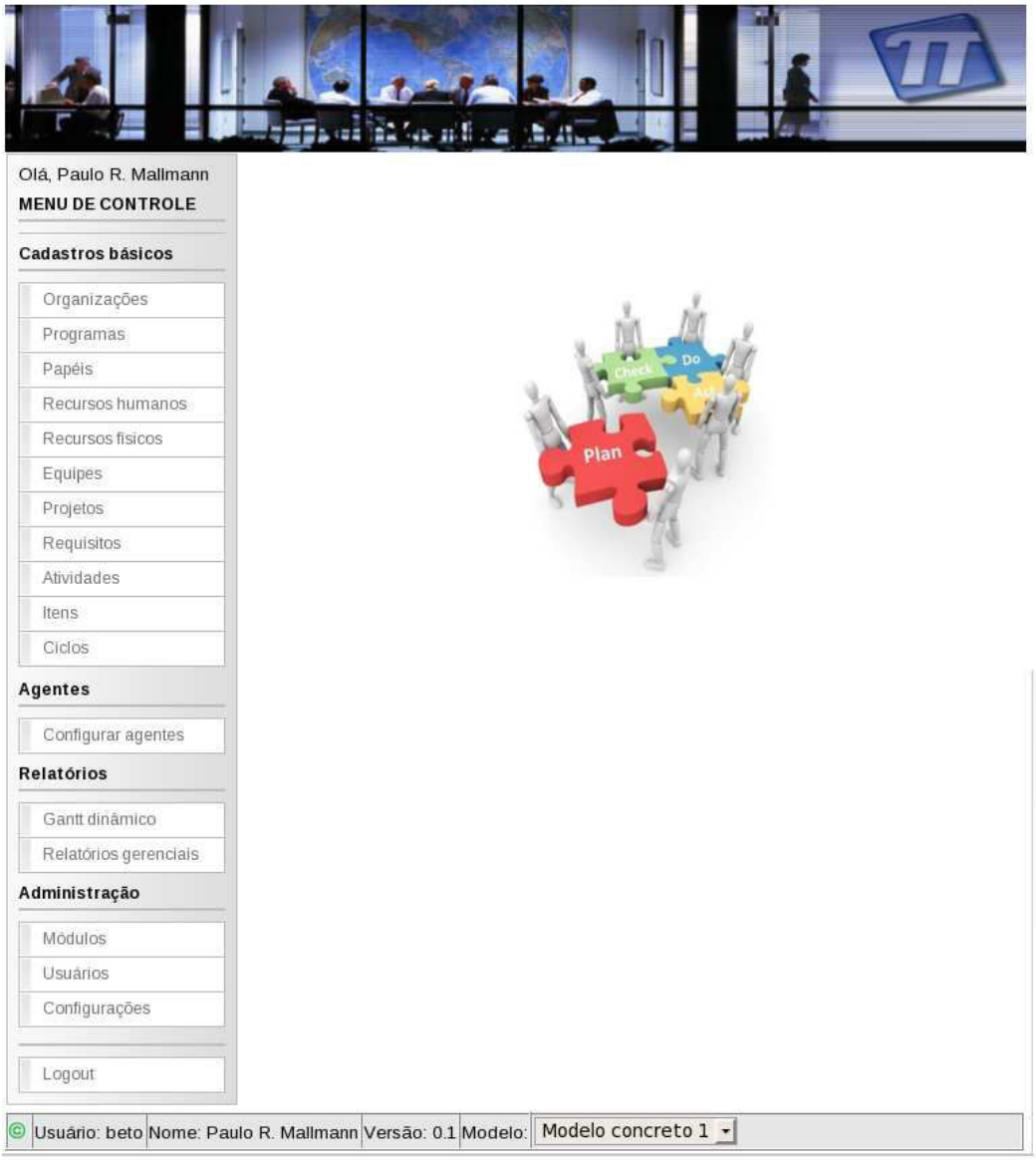


Figura 26: Interface de gerência do modelo concreto

4.10.1. Instância do modelo abstrato

Uma das características mais importantes da aplicação é justamente a possibilidade de permitir que o gerente de projeto possa instanciar, de acordo com a sua necessidade, características específicas do metamodelo abstrato criando modelos concretos de gerenciamento de projetos. Esta característica tornou a aplicação bastante dinâmica e permitiu visualizar na prática as vantagens que o gerente de projeto pode vir a ter caso consiga escolher os métodos que mais se adaptam a sua realidade.

A figura 27 apresenta a interface que o gerente de projeto tem a disposição para instanciar o modelo abstrato:

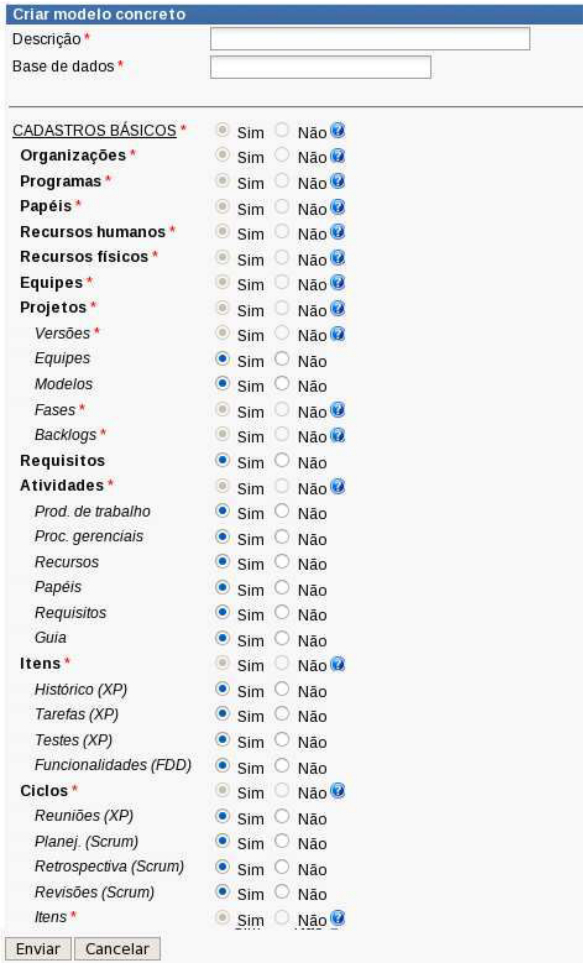


Figura 27: Instância do modelo abstrato

O usuário define uma descrição para o modelo concreto e o nome da base de dados que armazenará a seleção feita por ele. Alguns itens são obrigatórios e por isso já estão selecionados e não podem ser desmarcados. Os outros itens ficam a cargo do gerente do projeto decidir em utilizá-los ou não de acordo com a necessidade.

Assim que selecionadas as devidas opções a interface é processada e é criado um banco de dados com o nome que foi definido pelo usuário no formulário. Em seguida é adicionado mais um item a *ComboBox* que aparece na barra inferior da aplicação contendo a descrição informada pelo usuário. Para passar a utilizar este modelo concreto, basta selecionar este modelo na *ComboBox* e a interface se adaptará ao modelo concreto instanciado.

4.10.2. Cadastro de recursos

A aplicação disponibiliza interfaces distintas para o cadastro de recursos humanos e recursos físicos. Ambos são recursos para a aplicação, mas de acordo com o modelo definido, cada cadastro possui as suas especificidades.

Na interface de cadastro de recursos humanos apresentada na figura 28, são solicitadas

informações como nome, email, telefone, endereço e o número do CPF. Além disso, existe uma marca indicando se o recurso é relevante e se ele é externo. Existe também uma marca indicando se o respectivo recurso está ativo. Baseado no número do CPF, considerado o identificador único deste recurso, é possível fazer um elo de ligação com um sistema externo de recursos humanos a fim de manter atualizada a informação de disponibilidade do recurso.

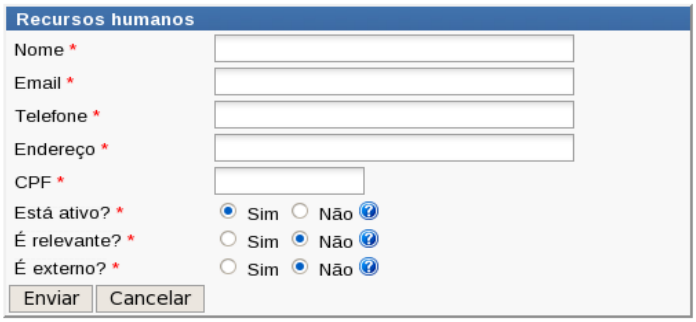


Figura 28: Interface de cadastro de recursos humanos

Na interface de cadastro de recursos físicos apresentada na figura 29, são solicitadas informações como nome e o número do patrimônio. Também é solicitado para identificar se aquele recurso físico trata-se de um material de consumo, equipamento ou recurso facilitador. Existe também uma marca indicando se o respectivo recurso está ativo. O número do patrimônio é utilizando como identificador único deste recurso e através dele é possível fazer um elo de ligação com um sistema de patrimônio externo a fim de manter atualizada a informação de disponibilidade do respectivo recurso físico.

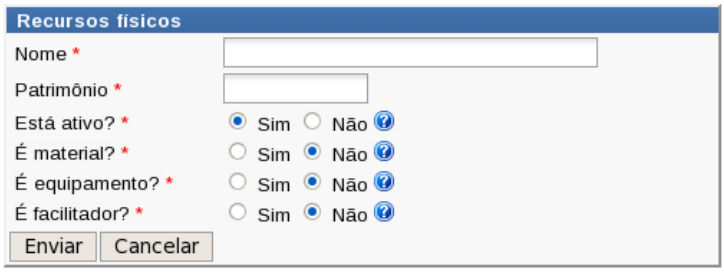


Figura 29: Interface de cadastro de recursos físicos

4.10.3. Cadastro do projeto

Após ter cadastrado uma organização e um programa é possível cadastrar o projeto que será gerenciado por aquele modelo concreto instanciado.

Em função do cadastro de projeto ter um grande número de campos a serem informados, seu cadastro foi dividido em abas que agrupam os campos referentes ao mesmo assunto. As abas “Projeto”, “Versões”, “Fases” e “Backlogs” são obrigatórias enquanto que as abas “Equipes” e “Modelos” irão aparecer dependendo se foram selecionadas no momento de

criar o modelo concreto.

A figura 30 apresenta a tela de cadastro de projetos que na sequência terá um detalhamento das informações contidas em cada aba.

Figura 30: Cadastro de projetos

Na primeira aba chamada “Projeto” são informados o nome e o escopo do projeto bem como o programa ao qual ele se relaciona. Além disso, baseado nos recursos humanos cadastrados anteriormente, é possível selecionar o gerente do projeto, estipular a data inicial e projetar uma data final.

A segunda aba chamada “Versões” é o local onde é possível cadastrar as versões que o projeto terá bem como as datas de lançamento das mesmas.

Na terceira aba chamada “Equipes” são vinculadas as equipes ao projeto.

Na quarta aba chamada “Modelos” é possível cadastrar os domínios e as áreas de conhecimento dos modelos utilizados no projeto.

A quinta aba chamada “Fases” é o local onde são definidas as fases do projeto.

Finalmente, a sexta aba chamada “Backlogs” permitirá definir em quantas etapas de implementação o projeto será dividido definindo quem será o responsável por cada uma delas bem como a sua prioridade frente ao projeto.

4.10.4. Cadastro das atividades

Na interface de cadastro das atividades é possível começar a visualizar as

características próprias de cada modelo concreto instanciado, ou seja, as abas do cadastro de atividades são dinâmicas de acordo com a seleção feita pelo gerente do projeto no momento de instanciar o modelo abstrato. Somente a aba "Atividade" é obrigatória. As abas "Produtos de trabalho", "Processos gerenciais", "Recursos", "Papéis", "Requisitos" e "Guias" irão aparecer dependendo da seleção feita no momento de instanciar o modelo abstrato.

A figura 31 apresenta a tela de cadastro de atividades que é vinculada a uma fase do projeto:

Figura 31: Cadastro de atividades

Na figura apresentada, a tela de cadastro de atividades está com todas as abas habilitadas, ou seja, no momento de instanciar o modelo abstrato para este projeto foram selecionados todos os itens relacionados a gerência de atividades.

Na primeira aba chamada "Atividade" é definido o nome, a descrição e a fase do projeto ao qual aquela atividade está relacionada. Além disso, é cadastrado uma data inicial e uma data de previsão para o término daquela atividade. Nesta aba também é possível relacionar uma atividade dependente. Ainda existe um campo data final que é preenchido automaticamente pelo agente *ActivityController* quando este detectar que todos os itens desta atividade estiverem concluídos.

Na segunda aba chamada "Produtos de trabalho" são cadastrados os produtos de trabalho tais como diagramas, especificação, casos de uso, etc.

Na terceira aba chamada "Processo gerenciais" é cadastrado uma descrição para cada processo bem como a área de conhecimento e os grupos de processos.

Na quarta aba chamada "Recursos" é possível relacionar recursos físicos e humanos necessários para a conclusão da respectiva atividade.

Na quinta aba chamada "Papéis" são definidos os papéis que cada recurso humano poderá ter na execução da atividade.

Na sexta aba chamada “Requisitos” é possível relacionar requisitos a respectiva atividade.

Na sétima e última aba chamada “Guias” são definidas documentações e guias gerados durante aquela atividade identificando se são ferramentais ou técnicos.

4.10.5. Cadastro dos itens

Na tela de cadastro dos itens também é contemplada a dinamicidade do modelo concreto instanciado, ou seja, as abas do cadastro de itens são dinâmicas de acordo com a seleção feita no momento de instanciar o modelo abstrato. Somente a aba “Itens” é obrigatória. As abas “Histórico”, “Tarefas”, “Testes” e “Funcionalidades” irão aparecer dependendo da seleção feita no momento de criar o modelo.

A figura 32 apresenta a interface do cadastro de itens que é vinculada a uma atividade do projeto:

Figura 32: Cadastro de itens

Da mesma forma como apresentado no cadastro de atividades, a tela de cadastro de itens está com todas as abas habilitadas o que significa que foram selecionados todos os itens relacionados a gerência de itens no momento de instanciar o modelo abstrato.

Na primeira aba chamada “Item” é definido um título, um resumo e a atividade do projeto ao qual aquele item está relacionado. Além disso, é definida uma data de criação e é possível cadastrar uma estimativa de tempo para a conclusão daquele item. Ainda existe um campo data de conclusão que é preenchido automaticamente pelo agente *BacklogItemController* quando este detectar que todas as tarefas deste item estiverem concluídas.

A segunda aba chamada “Tarefas” é específica da metodologia ágil XP e devido a isso permite estipular uma descrição para a tarefa bem como o par de programadores que está

escalado para atendê-la. Ainda nesta aba existe uma informação dizendo qual o percentual de conclusão desta tarefa que é alimentado também pelo agente *BacklogItemController.*

A terceira aba chamada “Funcionalidades” é específica da metodologia FDD permite cadastrar uma descrição e o programador responsável por aquela tarefa. Da mesma forma como nas tarefas do XP, o agente *BacklogItemController* alimentará o campo percentual de conclusão automaticamente.

Na quarta aba chamada “Histórico” é permitido armazenar os comentários descritos pelos programadores da tarefa ou funcionalidade bem como a data de ocorrência de cada um dos históricos.

Na quinta e última aba chamada “Testes” são informados os testes efetuados bem como é possível identificar se eles já estão validados.

4.10.6. Cadastro dos ciclos

O cadastro de ciclos abstrai os conceitos de iteração do XP, *Sprint* do Scrum e *ConstructionCycle* da FDD. Antes de iniciar os cadastros dos ciclos, já devem ter sido definidas na interface de cadastro de projeto as suas versões e datas de disponibilização.

Nesta interface as abas “Ciclos” e “Itens” são obrigatórias. As abas “Reuniões”, “Planejamento”, “Revisões”, e “Retrospectiva” irão aparecer dependendo da seleção feita no momento da instância do modelo.

A figura 33 apresenta a interface do cadastro de ciclos que é vinculada a uma versão do projeto:

Figura 33: Cadastro de ciclos

Na primeira aba chamada “Ciclo” é definida uma descrição para o ciclo e ele é vinculado a uma versão do projeto. Além disso, é definida uma data de início e uma data prevista de fim para o ciclo. Ainda existe um campo data de fim que é preenchido automaticamente pelo agente *CycleController* quando este detectar que todos os itens deste ciclo estiverem concluídos.

A segunda aba chamada “Itens” permite fazer a relação de itens com etapas do projeto. Para cada relacionamento destes adicionado é possível definir um programador chefe que irá conduzir os trabalhos de desenvolvimento dos itens relacionados bem como a prioridade frente aquele ciclo que está se definindo.

A terceira aba chamada “Reuniões” é específica da metodologia XP e permite armazenar o título, um breve resumo e a data das reuniões ocorridas no respectivo ciclo de desenvolvimento.

Na quarta aba chamada “Planejamento” relaciona-se as reuniões de planejamento próprias da metodologia *Scrum* onde é traçado um objetivo e lançado uma data alvo para atingí-lo.

Na quinta aba chamada “Revisões” é possível detalhar as descrições das reuniões de revisão do planejamento feitas durante o ciclo.

A sexta e última aba chamada “Retrospectiva” é específica da metodologia *Scrum* e permite informar uma análise detalhada após o término do ciclo indicando quais as técnicas que devem se manter e o que necessita melhorar para os próximos ciclos.

4.10.7. Configurações dos agentes

Após a estruturação do projeto através do cadastro das atividades, itens, ciclos e do projeto propriamente dito, é necessário configurar os agentes de software para que eles iniciem o trabalho de alimentar o modelo concreto instanciado.

A configuração dos agentes de software é feita a nível de projeto permitindo com isso que o gerente possa definir, por exemplo, frequências diferentes de atualização caso ele tenha criado mais de um projeto em um mesmo modelo concreto.

A interface de configuração dos agentes segue o mesmo padrão de abas que foi utilizado nos cadastros. A figura 34 apresenta esta interface que a seguir será detalhada:

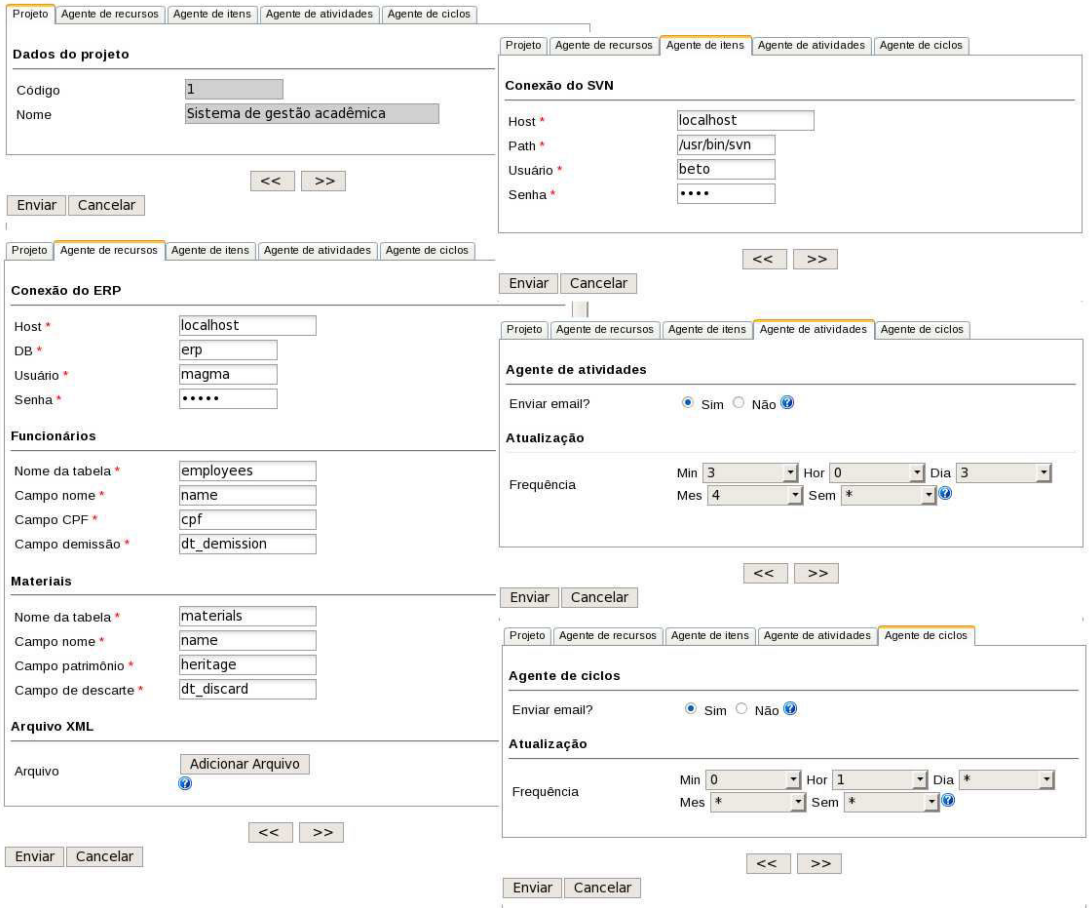


Figura 34: Interface de configuração dos agentes de software

A primeira aba chamada “Projeto” tem função de identificar qual projeto está tendo seus agentes configurados.

A segunda aba chamada “Agente de recursos” define todas as configurações para que o agente de recursos possa agir. Este agente sincroniza as informações de recursos humanos e recursos físicos de acordo com a frequência de atualização definida também nesta interface. Esta atualização pode ser feita de duas maneiras distintas:

- **Sincronização via conexão direta com uma aplicação externa:** Esta maneira de trabalhar exige que o gerente do projeto configure o endereço, o banco de dados, o usuário e a senha de conexão com a aplicação externa que é a fonte oficial das informações. Além disso, devem ser definidas as tabelas e o nomes dos campos que armazenam os funcionários e os materiais que serão utilizados como base para a sincronização.
- **Sincronização via arquivo XML:** Esta maneira de trabalhar é mais simples pois basta que o gerente do projeto informe um arquivo XML que o agente sempre utilizará este como fonte oficial das informações para sincronizar a base de dados. O formato deste arquivo XML é bem simples e está apresentado na figura 35:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<resource>
 <stakeholder>
  <name>Paulo Roberto Mallmann</name>
  <identifier>96632330010</identifier>
  <is_inactive/>
 </stakeholder>
 <stakeholder>
  <name>Viviane Berner</name>
  <identifier>72077492015</identifier>
  <is_inactive/>
 </stakeholder>
 <stakeholder>
  <name>Pablo Dall Oglio</name>
  <identifier>97979058020</identifier>
  <is_inactive>2010-11-03</is_inactive>
 </stakeholder>
 <physical>
  <name>Computador</name>
  <identifier>1234567890</identifier>
  <is_inactive>2010-11-01</is_inactive>
 </physical>
</resource>
```

Figura 35: Arquivo XML modelo

A terceira aba chamada “Agente de itens” define as configurações para que o agente de itens possa agir. Este agente faz uso de uma interface gráfica desenvolvida especialmente para o programador utilizar como ferramenta de envio de arquivos para o repositório de versões. E nesta mesma interface, que será detalhada a seguir, é possível informar outros dados que serão armazenados no modelo concreto relacionado ao projeto que se está trabalhando. No entanto, para que o agente de itens possa ser utilizado perfeitamente é necessário configurar o endereço, o caminho, o usuário e a senha do repositório de controle de versões. Definidas estas informações ele poderá ser utilizado pelos programadores sempre que for necessário colocar em produção determinadas funcionalidades ou correções.

A quarta aba chamada “Agente de atividades” define as principais variáveis para que o agente de atividades possa ser executado. Este agente permite que o gerente configure se deseja receber email toda vez que ele for executado bem como é possível definir a frequência de execução deste agente.

A quinta e última aba chamada “Agente de ciclos” define as principais configurações do agente de ciclos. Da mesma forma como o agente de atividades, ele também permite que o gerente configure se deseja receber email toda vez que ele é executado bem como é possível estabelecer a frequência de execução deste agente.

4.10.8. O funcionamento dos agentes de software

O ambiente conta com quatro agentes de software que foram desenvolvidos para

manter o modelo concreto atualizado: *ResourceManager, ActivityController, BacklogItemController* e *CycleController.* Cada um destes agentes interage com uma parte do modelo que será detalhado em sequência.

Cada um dos agentes se comunica com o modelo através de uma arquitetura de *Web Services*, formada por um conjunto de tecnologias de padrão aberto, que interagem sob uma plataforma de internet. No meio de todo o processo está o protocolo SOAP (*Simple Object Access Protocol*) provendo a comunicação entre as aplicações. SOAP é um protocolo herdeiro do padrão XML que encapsula um conjunto de regras para descrição de dados e processos, através de um mecanismo simples para definir a semântica de uma aplicação através de um modelo de empacotamento e um mecanismo de codificação e foi projetado para a troca de informações em um ambiente descentralizado através do protocolo de comunicação HTTP e do formato XML (CHAVDA, 2004).

Dessa forma, para uma aplicação trabalhar com *Web Services*, basta a compatibilidade com SOAP, tanto no lado do cliente que cria o documento XML com a informação necessária para invocar o serviço quanto no lado do servidor que é o responsável por executar a mensagem como um interpretador (CHAVDA, 2004).

O agente *ResourceManager* interage com a parte do modelo que armazena os recursos humanos e os recursos físicos. Este agente de software, baseado nas configurações definidas para ele mantém sincronizados os recursos do modelo concreto através de uma conexão direta com um sistema externo ou via um arquivo XML. Ambos os métodos foram exemplificados na seção configuração dos agentes. A figura 36 apresenta a saída gerada pelo agente de sincronização de recursos:

```
[beto@beto ResourceManager]$ php ResourceManager.php

Sincronizando recursos
3 - Andrigo Dametto                     - Ativo
4 - Nataniel Rabaioli                   - Ativo
1 - Paulo Roberto Mallmann              - Ativo
7 - Viviane Berner                      - Ativo
6 - Maria Malheiros                     - Ativo
5 - Rodrigo Luciano Gattermann          - Inativo
2 - Pablo Dall Oglio                    - Inativo
9 - Quadro branco                       - Ativo
8 - Computador                          - Inativo
Recursos sincronizados via conexão no ERP.
```

Figura 36: Saída da execução do agente de recursos

O agente *ResourceManager* faz a conexão ou a leitura do arquivo XML e compara com os recursos alocados no projeto. Para recursos humanos é utilizado como elo de ligação o CPF das pessoas e para recursos físicos o número de patrimônio. Diante disso, é feita a

sincronização da base de dados do modelo concreto instanciado na frequência definida na configuração do agente de recursos.

O agente *ActivityController* tem o foco em manter atualizadas as atividades do projeto. Este agente basicamente fica monitorando o percentual de conclusão dos itens de desenvolvimento vinculados a atividade mantendo a mesma atualizada. Toda vez que ele é executado, é feita uma varredura nas atividades verificando o quanto de cada uma já está concluído e no caso de estar totalmente finalizada, seta a data final na respectiva atividade. A figura 37 apresenta a saída gerada pelo agente de atualização das atividades:

```
[beto@beto ActivityController]$ php ActivityController.php

ATUALIZANDO ATIVIDADES DO PROJETO Sistema de gestão acadêmica
Definir os currículos          - 25% concluído
Oferecer o horário             - 60% concluído
Matricula dos alunos           - 20% concluído
Gerar cobrança                 - 0% concluído
Testar e documentar            - 67.5% concluído
PROJETO Sistema de gestão acadêmica ATUALIZADO COM SUCESSO!!!
```

Figura 37: Saída da execução do agente de atividades

O agente *ActivityController* consegue identificar o percentual de conclusão das atividades baseado nos itens que elas possuem relacionadas. Ele também está preparado para agir na configuração definida pelo gerente no momento de criar o modelo concreto e na frequência definida na interface de configuração do agente de atividades.

O agente *BacklogItemController* é uma interface criada especialmente para o desenvolvedor utilizar em seu ambiente de trabalho. Este agente age diretamente sobre as tarefas de desenvolvimento agrupando os conceitos encontrados nas metodologias ágeis XP, *Scrum* e FDD. A figura 38 apresenta a interface do agente de itens:

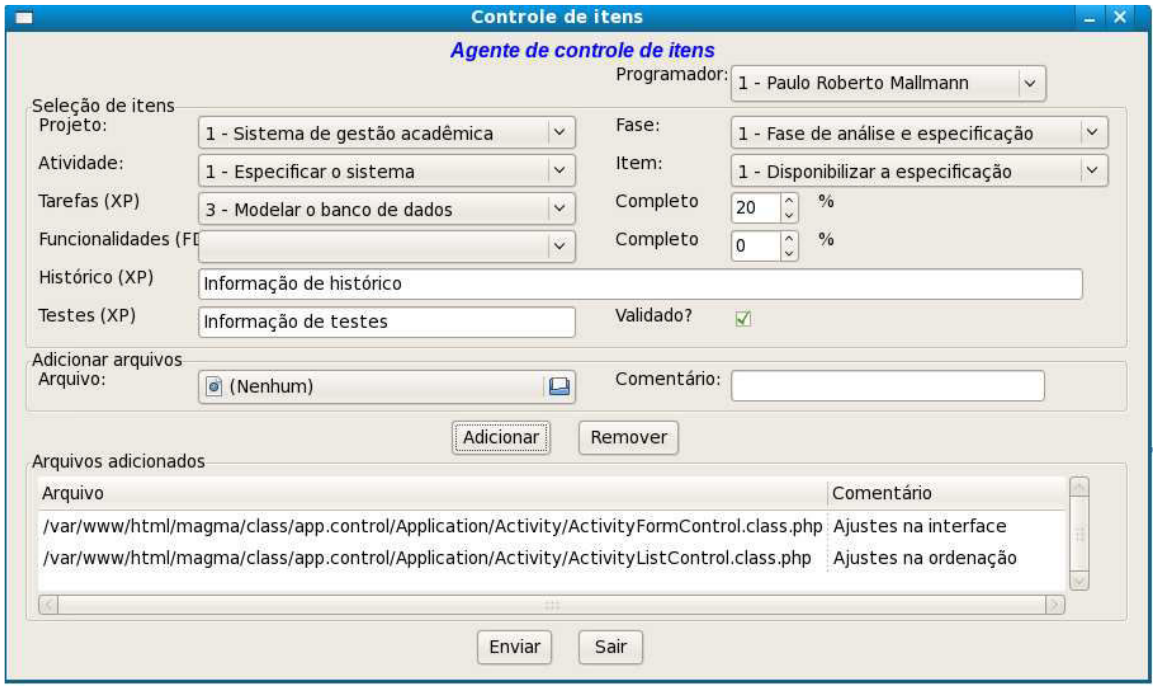


Figura 38: Interface do agente de itens

A interface do agente *BacklogItemController* foi desenvolvida utilizando a linguagem de programação PHP-GTK[6]. Toda a camada de comunicação com o modelo concreto é feita através de *WebServices*. A interface é totalmente dinâmica e os campos aparecerão de acordo com o que foi selecionado na criação do modelo concreto. Além disso, tomou-se um cuidado especial na criação da interface para que ela fosse intuitiva, evitando que o programador lance informações que possam gerar inconsistências no modelo. Para isso as *ComboBox* de atividades, itens, tarefas e funcionalidades se ajustam a medida que o desenvolvedor vai selecionando os campos anteriores. O campo percentual de conclusão também tem um controle que não permite ultrapassar o que realmente falta para concluir a tarefa ou funcionalidade. Além disso, a interface está integrada com o controle de versões e faz o envio automático dos arquivos para o repositório.

O agente *CycleController* tem o foco em manter atualizados os ciclos de desenvolvimento do projeto. Este agente basicamente fica monitorando o percentual de conclusão dos itens de desenvolvimento vinculados ao ciclo mantendo o mesmo atualizado. É feita uma leitura em todos os itens vinculados ao ciclo toda vez que ele é executado. Caso todos os itens estiverem concluídos, o agente preenche a data final no ciclo de desenvolvimento. A figura 39 apresenta a saída gerada pelo agente de atualização dos ciclos do projeto:

6 http://gtk.php.net

```
[beto@beto CycleController]$ php CycleController.php

ATUALIZANDO CICLOS DO PROJETO Sistema de gestão acadêmica
Especificação e cadastros básicos       : Ciclo não concluído!!!
Cadastros e processos acadêmicos        : Ciclo não concluído!!!
Processos financeiros                   : Ciclo não concluído!!!
Documentação e finalização              : Ciclo não concluído!!!
CICLOS DO PROJETO Sistema de gestão acadêmica ATUALIZADOS COM SUCESSO!!!
```

Figura 39: Saída da execução do agente de ciclos

O agente *Cycle Controller* está preparado para agir de acordo com a configuração definida pelo gerente no momento de criar o modelo concreto e na frequência definida na interface de configuração do agente de ciclos.

4.10.9. Relatórios gerenciais

Com o objetivo de disponibilizar uma visualização rápida das informações do projeto contidas no modelo, foi elaborada uma interface para extrair relatórios gerenciais que é apresentada na figura 40. Ela permite que o próprio gerente monte o seu relatório através de uma série de parametrizações que serão explicadas a seguir.

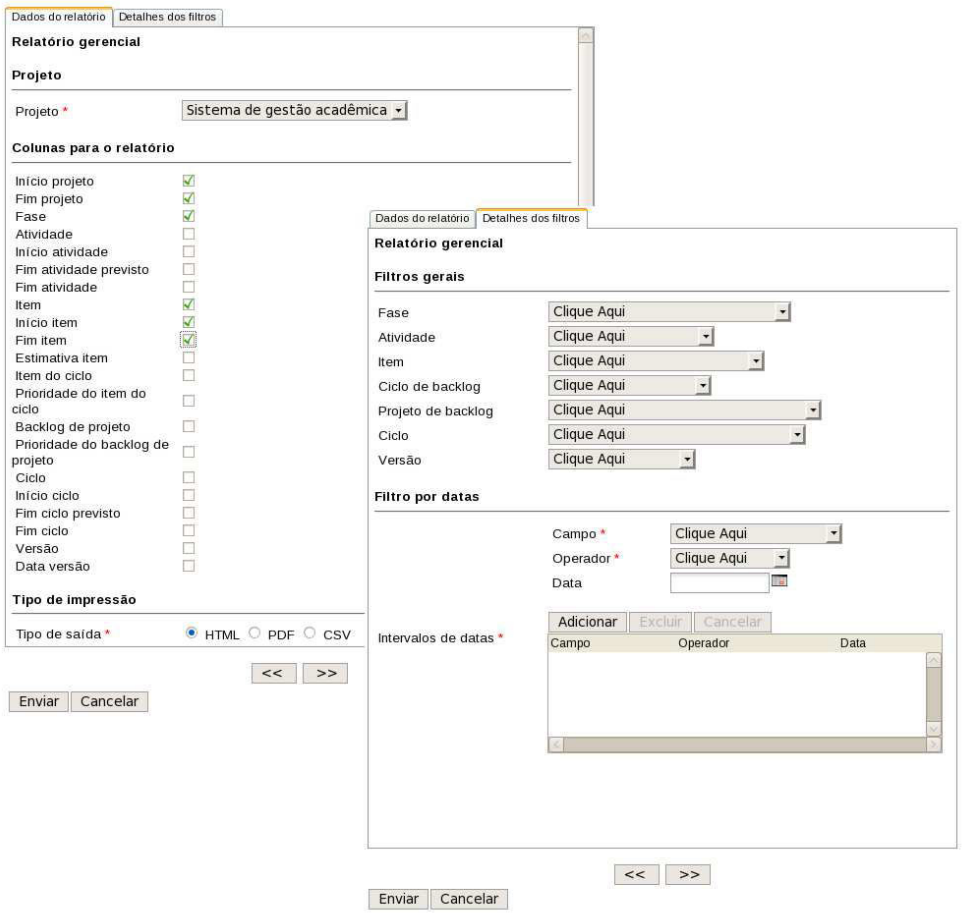


Figura 40: Interface de parametrização de um relatório gerencial

A interface de extração de relatórios gerenciais disponibiliza duas abas distintas. A primeira chamada “Dados do relatório” solicita a identificação do projeto e permite selecionar quais colunas

deverão aparecer no relatório. Além disso, nesta primeira aba ainda é possível que o gerente escolha o tipo de saída do relatório entre as opções HTML, PDF e CSV. Já a segunda aba chamada "Detalhes dos filtros" permite que o gerente combine alguns filtros para chegar ao relatório desejado. A figura 41 apresenta um relatório dinâmico extraído da aplicação.

 # Relatório gerencial
Projeto: Sistema de gestão acadêmica

Projeto	Início projeto	Fim projeto	Fase	Item	Início item	Fim item
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de análise e especificação	Disponibilizar a especificação	2010-06-01	2010-11-16
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de desenvolvimento	Cadastro de pessoas	2010-08-01	
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de desenvolvimento	Cadastro de currículos	2010-10-01	
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de desenvolvimento	Oferecer as turmas	2010-11-01	
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de desenvolvimento	Efetuar a matrícula	2010-12-01	
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de desenvolvimento	Gerar o financeiro	2011-02-01	
Sistema de gestão acadêmica	2010-06-01	2011-06-01	Fase de testes e documentação	Homologar o sistema	2011-04-01	

Figura 41: Exemplo de saída de um relatório gerencial

4.10.10. Gráfico de *gantt* dinâmico

A ideia central do trabalho é possibilitar uma gerência de projetos de software de uma forma personalizada e de acordo com as necessidades de um projeto específico. Com o intuito de atender a esta premissa básica, foi desenvolvida uma interface totalmente dinâmica para apresentar ao gerente do projeto a situação real do andamento das atividades previstas no planejamento.

Para isso, foram utilizados todos os conceitos básicos de diagramas de *gantt* que são comumente utilizados para ilustrar o avanço das diferentes etapas de um projeto. Os intervalos de tempo representando o início e fim de cada fase aparecem como barras coloridas sobre o eixo horizontal do gráfico. Desenvolvido em 1917 pelo engenheiro social Henry Gantt, esse gráfico é utilizado como uma ferramenta de controle de produção. Nele podem ser visualizadas as atividades de cada membro de uma equipe, bem como o tempo utilizado para cumpri-la. Assim, pode-se analisar o empenho de cada membro no grupo, desde que os mesmos sejam associados, à atividade, como um recurso necessário ao desempenho da mesma.

A figura 42 apresenta a interface que foi desenvolvida para informar os parâmetros para a geração do gráfico de *gantt* de um projeto cadastrado na aplicação. A interface é extremamente simples requerendo somente que o usuário selecione o projeto que deseja gerar o gráfico de *gantt*.

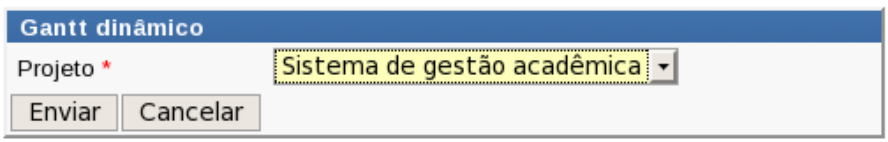


Figura 42: Parâmetros para a geração do gráfico de *gantt* dinâmico

apresenta um exemplo de gráfico de *gantt* gerado pela aplicação, que além das características tradicionais existentes neste tipo de gráfico, contempla alguns marcadores e indicadores específicos do modelo, que serão detalhados em seguida.

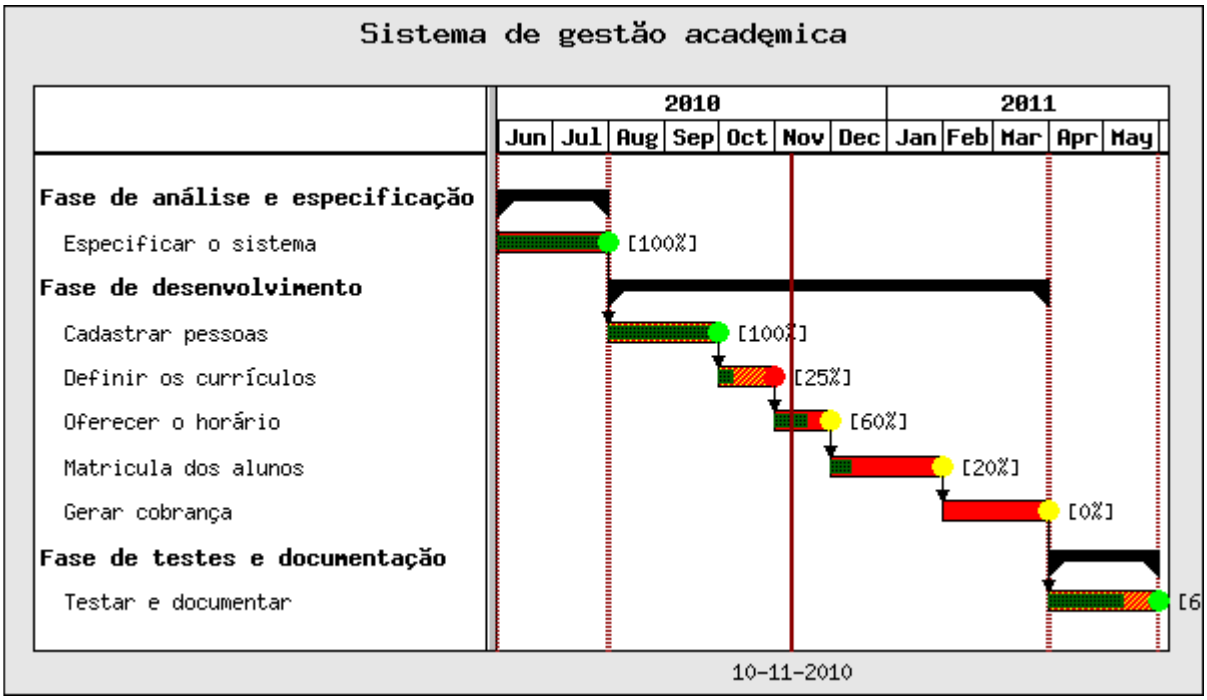


Figura 43: Gráfico de *gantt* dinâmico

O gráfico apresentado acima foi gerado dinamicamente e nele foi amplamente explorada a utilização de cores, visando alertar o gerente do projeto para pontos críticos e que requerem um replanejamento ou um acompanhamento especial. Além disso, cabe ressaltar alguns outros indicadores vindos do modelo:

- As dependências que ligam uma atividade a outra são dinâmicas e representadas pelas setas, ou seja, no cadastro das atividades é possível cadastrar as dependências entre as atividades e elas se ligam automaticamente no gráfico.
- As cores das barras das atividades também têm um significado importante. Uma *tarja verde* desenhada sobre uma barra indica o progresso daquela atividade. Caso a barra estiver com a cor *vermelha* é um indicativo que a atividade perdeu algum recurso físico ou humano importante. Se a barra estiver *laranja,* quer dizer que a atividade está sob controle e dentro do planejado.
- Os círculos ao final de cada barra também têm um significado especial. Caso o círculo estiver *vermelho* quer dizer que o prazo de conclusão da atividade foi excedido ou esta atividade perdeu um recurso importante. Se o círculo estiver *amarelo* é um indicativo que algum recurso desta atividade foi perdido, mas está dentro do prazo de conclusão e por isso ainda é possível fazer um replanejamento

para reverter a situação e não impactar no cronograma do projeto. E se o círculo estiver *verde* a atividade já está concluída ou está tudo conforme o planejado.

Todos os indicadores são dinâmicos e respeitam as informações contidas no modelo concreto instanciado e populado pelos agentes de software.

4.11. Conclusões da seção

O presente capítulo apresentou a visão completa da solução desenvolvida ao longo deste trabalho para resolver o problema relacionado a gerência de projetos de software em ambientes de desenvolvimento que utilizem metodologias ágeis.

Inicialmente foi feita uma breve contextualização dos principais metamodelos de metodologias ágeis estudados assim como também um estudo do metamodelo de gerência de projetos que foi adotado. Em seguida, foi dada uma visão geral do metamodelo de gerência de projetos de software em metodologias ágeis, fazendo um detalhamento classe por classe da estrutura que foi proposta. Apresentado o metamodelo, foi dada uma visão geral do ambiente para gerência de software em metodologias ágeis detalhando os principais agentes de softwares que foram projetados assim como também a arquitetura definida para o desenvolvimento da aplicação. E finalmente foi apresentada a aplicação desenvolvida que teve como base os conceitos contidos no metamodelo e na arquitetura proposta. Foram apresentadas as principais características da aplicação bem como detalhadas as principais telas e funcionalidades da mesma. Foi feita uma referência especial aos resultados obtidos com os primeiros experimentos que permite fazer uma comparação com os resultados obtidos em outros trabalhos semelhantes, o que será feito no capítulo seguinte.

Ficou claro que o ambiente proposto não pretende eliminar a interação humana do processo de gerência de projetos. O objetivo principal do ambiente é dar ao gerente do projeto uma visão mais dinâmica do andamento do projeto para que, em caso de algum problema, imediatamente ele visualize este alerta e tenha parâmetros necessários para realocar uma atividade ou uma tarefa evitando que isso afete o cronograma original do projeto. Do ponto de vista tecnológico, uma contribuição interessante foi a forte utilização de agentes de software que contribuíram para dar vida ao metamodelo proposto e resolver problemas de engenharia de software.

5. ANÁLISE COMPARATIVA ENTRE OS TRABALHOS

Neste capítulo será feita uma sistematização dos trabalhos relacionados com o trabalho desenvolvido a fim de apresentar o que a aplicação desenvolvida traz de vantagem e de inovação em relação a outros trabalhos avaliados.

5.1. Comparação entre os trabalhos

A análise comparativa deste trabalho com os trabalhos relacionados apresentados no Capítulo 3 se dá em relação a comparação dos modelos propostos pelos autores e das funcionalidades disponíveis nos protótipos apresentados para apoiar as atividades de gerência de projetos. No Capítulo 3 foram relacionados cinco trabalhos que agora serão apresentados novamente juntamente com o trabalho proposto. Para tal, a tabela 7 apresenta os trabalhos que serão comparados.

Tabela 7: Trabalhos a serem comparados

Trabalho	Descrição
T1	Uma proposta de integração entre métodos ágeis e métodos tradicionais
T2	Uma comparação entre métodos ágeis e o desenvolvimento iterativo incremental
T3	Uma avaliação comparativa de *frameworks* para metodologias ágeis
T4	Um sistema multi-agente baseado em métricas para gerência de projetos
T5	Proposta de integração entre gerência de projetos e desenvolvimento de software
TP	Um modelo abstrato de gerência de software para metodologias ágeis

A tabela 8 apresenta os critérios utilizados para comparar os trabalhos relacionados. A escolha destes critérios procurou observar pontos relevantes apresentados nos diversos trabalhos, tais como, utilização de metodologias ágeis, uso de agentes de software, definição de modelos concretos personalizados e registros de implementações práticas.

Tabela 8: Critérios de comparação entre os trabalhos

Critério	Descrição
A	Modelo com suporte as principais metodologias ágeis
B	Permite que o modelo abstrato seja instanciado de acordo com a necessidade;
C	Possui agentes de software que mantêm o modelo concreto atualizado;
D	Suporta diferentes projetos permitindo gerenciar projetos distintos;

Critério	Descrição
E	Suporta utilização *online* podendo ser utilizado pela *web;*
F	Permite sincronização de recursos do projeto com aplicativos externos;
G	Disponibiliza interface de atualização de itens para os desenvolvedores;
H	Gerenciamento pró-ativo sem ser necessário que o gerente acione a aplicação;
I	Oferece estatísticas gerenciais customizáveis pelo gerente;
J	Apresenta um *gantt* dinâmico com a real situação do projeto;
K	Fornece alertas ao gerente do projeto em relação aos principais riscos;

Com base nos critérios e trabalhos selecionados, é possível estabelecer uma matriz contendo a comparação dos trabalhos com os critérios selecionados, como pode ser visto na tabela 9. Nas linhas tem-se os critérios e nas colunas os trabalhos. Quadros com valor (X) indicam que o critério foi atendido, quadros com o valor (-) indicam que o critério não foi atendido ou não se aplica porque não houve registros de implementação prática.

Tabela 9: Comparação entre os trabalhos

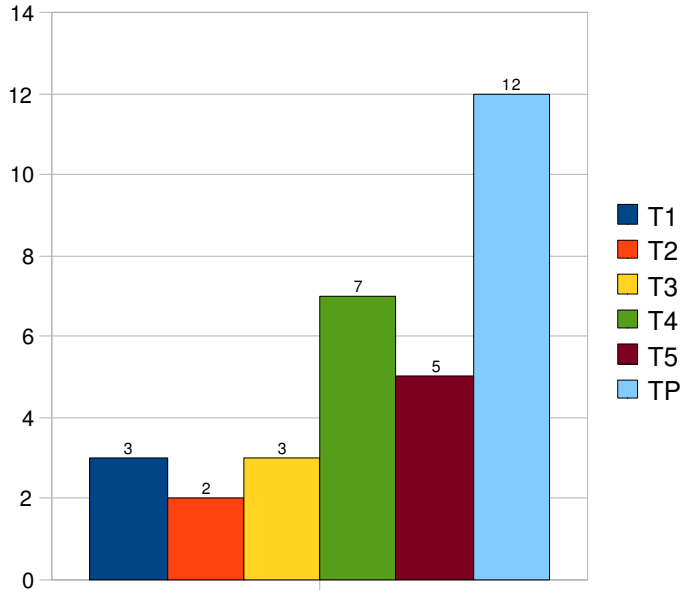
Critério	T1	T2	T3	T4	T5	TP
A	**X**	**X**	**X**	**-**	**-**	**X**
B	**-**	**-**	**-**	**-**	**-**	**X**
C	**-**	**-**	**-**	**X**	**-**	**X**
D	**X**	**X**	**X**	**X**	**X**	**X**
E	**-**	**-**	**-**	**X**	**-**	**X**
F	**-**	**-**	**-**	**-**	**X**	**X**
G	**-**	**-**	**-**	**X**	**-**	**X**
H	**-**	**-**	**-**	**-**	**-**	**X**
I	**X**	**-**	**X**	**X**	**X**	**X**
J	**-**	**-**	**-**	**X**	**X**	**X**
K	**-**	**-**	**-**	**X**	**X**	**X**

Por meio da matriz de comparação entre os trabalhos estudados e o trabalho proposto, pode-se perceber que o trabalho proposto é o que contempla o maior número critérios que são diretamente relacionados a gerência de projetos de software utilizando metodologias ágeis. Baseado nesta matriz é possível observar que o trabalho proposto é o único que permite que o modelo abstrato seja instanciado de acordo com a necessidade de cada empresa. Além disso, a

dinamicidade do modelo e dos agentes de software definidos trazem uma característica ímpar ao trabalho que permite um gerenciamento pró-ativo do projeto sem ser necessário que o gerente acione manualmente uma aplicação.

O T2 foi o trabalho que atendeu ao menor número de critérios em função de ter sido um estudo de caso sem registros de implementação prática. A mesma afirmação vale para os trabalhos T1 e T3 que atenderam somente a 3 critérios cada um. No entanto, dos trabalhos que apresentaram registros de implementação prática, que é o caso dos trabalhos T4 e T5, nenhum atendeu menos de cinco dos doze critérios analisados. Nenhum dos trabalhos, com exceção do trabalho proposto, atendeu a mais de sete critérios. A figura 44 apresenta em forma de gráfico a quantidade de critérios atendidos por cada trabalho.

Figura 44: Critérios atendidos por cada trabalho



5.2. Conclusões da seção

Ao analisar os resultados da avaliação comparativa de cada trabalho, nota-se que o trabalho proposto atende ao maior número de critérios. Tal característica se justifica no fato de que o ambiente foi projetado para tal, ou seja, o ambiente apresentado neste trabalho é mais específico em seu nicho de atuação, o que lhe permite uma maior aderência às necessidades de gerência de projetos de software em metodologias ágeis.

6. ESTUDO DE CASO

Neste capítulo será apresentado como se deram as atividades de verificação e validação do ambiente desenvolvido garantindo que ele atende às suas especificações e é plenamente funcional quando inserido em um ambiente real de uso. Neste sentido, uma estratégia foi realizar testes funcionais implantando a aplicação em uma organização externa e garantindo que o ambiente de produção tenha as mesmas condições do ambiente onde foram feitas as experimentações.

Antes da realização do estudo de caso, ocorreu uma espécie de validação interna paralela ao seu desenvolvimento. Foi preparado um projeto fictício com atividades, itens e ciclos de desenvolvimento bem como foram alocados recursos físicos e humanos a cada uma das atividades. Esta validação interna serviu como prova dos conceitos desenvolvidos mas não trabalhou com uma massa de dados muito grande.

6.1. Contextualização do ambiente

A organização externa escolhida para a validação real do ambiente foi uma instituição privada de ensino superior chamada Univates que está localizada na cidade de Lajeado no interior do Rio Grande do Sul a aproximadamente 120 Km da capital Porto Alegre. A instituição surgiu no ano de 1969 e durante todos estes anos dedicou-se exclusivamente ao conhecimento e ao crescimento da região onde está inserida. Atualmente a instituição conta com aproximadamente 11.000 alunos distribuídos entre os 42 cursos de graduação, 3 sequenciais, 9 técnicos, 27 pós-graduação (*lato sensu* e *stricto sensu*) além de diversos cursos de extensão oferecidos. Possui aproximadamente 350 professores e mais de 300 funcionários técnico-administrativos que executam todos os procedimentos necessários para bem atender aos alunos.

Esta instituição externa tem uma característica particular pois possui uma equipe interna de desenvolvimento de sistemas composta por analistas, programadores, *web designers* e documentadores que além de manter uma série de sistemas já existentes, desenvolvem novos projetos que visam atender a demandas da própria instituição. No momento em que a instituição externa foi escolhida, estava sendo desenvolvido um projeto de construção de um portal direcionado para empresas interessadas em recrutar estagiários, agenciar estágios e emitir segundas vias de pagamentos. Este portal serviria de meio de comunicação entre a instituição e as empresas que se mostrarem interessadas em utilizar estes serviços.

O desenvolvimento do portal iniciou em setembro de 2010 e foi implantado em

dezembro do mesmo ano. Teve o envolvimento de aproximadamente 10 usuários responsáveis pelas definições dos processos, 2 analistas de sistemas responsáveis por especificar o sistema, 3 programadores que desenvolveram as rotinas do portal, o apoio de 1 *web designer* para auxiliar na estruturação da interface do portal e 1 pessoa para documentar e escrever o manual do sistema.

A instalação e a implantação da aplicação proposta foi facilitada uma vez que o ambiente de desenvolvimento da instituição referida utiliza-se somente de ferramentas livres tais como: Linux, Apache, PHP, PostgreSQL e *Subversion*, que são as mesmas tecnologias utilizadas para o desenvolvimento da aplicação. A validação do ambiente ocorreu entre os dias 15/11/2010 e 15/12/2010. Neste período, em decorrência de observações de falhas e sugestões de melhorias a partir dos testes efetuados, vários ajustes foram feitos na aplicação permitindo concluir que ela traz resultados positivos conforme comentado na seção seguinte.

6.2. Detalhamento do cenário

O primeiro passo da montagem do cenário foi a instalação do ambiente. Esta tarefa foi bastante facilitada uma vez que a instituição referida utiliza em seu ambiente de desenvolvimento as mesmas tecnologias que foram utilizadas para o desenvolvimento da aplicação.

Tendo a aplicação instalada, foi criado nela o projeto denominado “Portal de Empresas” instanciando o modelo concreto de acordo com as necessidades do projeto. No momento da instância do modelo abstrato e criação do modelo concreto, o gerente do projeto definiu um modelo mesclando várias características do XP e do *Scrum.* A instituição escolhida já tem como prática utilizar metodologias ágeis de desenvolvimento. Normalmente são aplicados conceitos de programação em pares, integração contínua entre os desenvolvedores, reuniões de fim de expediente e principalmente a participação do usuário durante todo o processo de desenvolvimento do projeto. Além disso, existe um planejamento básico inicial que permite definir sobre ele as principais atividades e as tarefas de desenvolvimento.

A figura 45 apresenta detalhadamente os itens selecionados pelo gerente do projeto no momento da criação do modelo concreto do estudo de caso:

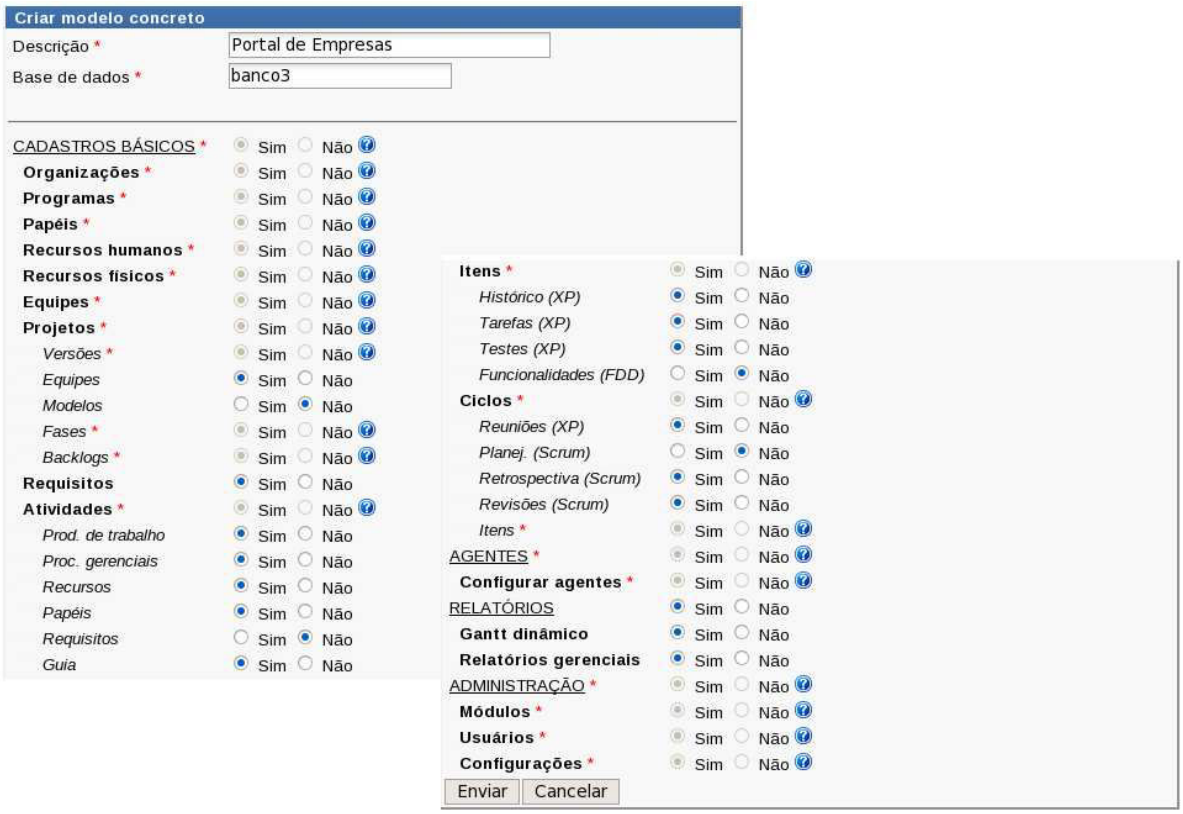


Figura 45: Instância do modelo abstrato no estudo de caso

Criado o modelo concreto, o próximo passo foi definir as fases do projeto e planejar atividades para cada uma delas. Por ser um projeto relativamente pequeno, optou-se pelo modelo tradicional de distribuição em fases criando somente uma fase adicional chamada "*Design e Layout*" devido a característica específica do projeto ser um portal de acesso externo que necessita ter os mesmos estilos e padrões dos demais portais da instituição. Seguem abaixo, as fases e suas respectivas atividades definidas para o projeto Portal de Empresas:

- **Fase de análise e especificação**
  - **Atividade de mapeamento dos requisitos:** Consistiu basicamente em reuniões com os usuários responsáveis pelas definições dos processos com o intuito de mapear os cadastros, processos e relatórios do portal e os principais requisitos do portal.
  - **Atividade de especificação do sistema:** Escrita do documento de especificação contendo os diagramas de casos de uso, diagramas de sequência e os diagramas de classes.
- **Fase de *design e layout***
  - **Atividade *design* do *layout* do portal:** Conduzida pelo *Web Designer* da

equipe em conjunto com um desenvolvedor com o propósito de definir as cores, os estilos, os menus, a disposição das notícias e montar a interface administrativa do portal.

- **Fase de desenvolvimento**
  - **Atividade migração de dados:** O objetivo desta atividade foi modelar o banco de dados definindo os relacionamentos entre as tabelas. Além disso, foi um item desta atividade migrar os dados de um sistema de recrutamento de estagiários que já existia na instituição em outra aplicação e que sua funcionalidade foi incorporada neste portal.
  - **Atividade interface de recrutamento:** Desenvolvimento da interface de recrutamento de estagiários que permite que empresas se cadastrem, disponibilizem vagas e em cima destas sejam feitos filtros de currículos disponíveis para serem empacotados e enviados para as empresas. Além da interface da empresa, foi disponibilizado uma interface para os alunos cadastrarem seus currículos para que estes possam ser filtrados e disponibilizados para as empresas.
  - **Atividade interface de agenciamento:** Atividade que requereu o desenvolvimento de uma interface para as empresas efetuarem a contratação de estagiários selecionados. Mensalmente a empresa pode ajustar os valores dos salários dos estagiários e emitir uma fatura de pagamento que quando recebida pela instituição agenciadora deste estágio, esta imediatamente repassa o valor para as contas dos estagiários.
  - **Atividade interface de emissão de boleto**: Permite que as empresas emitam segundas vias de faturas de pagamentos.
  - **Atividade integrações com ERP (*Enterprise Resource Planning*):** Integrar com o cadastro de empresas e com o sistema financeiro do ERP da instituição.
- **Fase de testes e documentação**
  - **Atividade documentação e testes:** Definição dos testes de interface e de processo bem como a execução e validação dos mesmos. Escrita do manual de usuário do sistema.

A tabela 10 apresenta as 9 atividades citadas acima com suas datas de início e previsão de conclusão definidas pelo gerente do projeto na instituição:

Tabela 10: Atividades definidas para o estudo de caso

Fase	Atividade	Data de início	Previsão conclusão
Análise e especificação	Mapeamento dos requisitos	15-09-2010	24-09-2010
	Especificação do sistema	27-09-2010	01-10-2010
Design e *layout*	*Design* do *layout* do portal	04-10-2010	08-10-2010
Desenvolvimento	Migração de dados	11-10-2010	22-10-2010
	Interface de recrutamento	18-10-2010	05-11-2010
	Interface de agenciamento	25-10-2010	19-11-2010
	Interface de emissão de boleto	22-11-2010	26-11-2010
	Integrações com ERP	22-11-2010	03-12-2010
Testes e documentação	Documentação e testes	06-12-2010	15-12-2010

Estas atividades podem ser extraídas do modelo concreto através da consulta SQL apresentada na figura 46 onde são cruzadas as tabelas *phase* e *activity* do modelo e selecionadas as colunas descrição da fase, descrição da atividade, data de início da atividade e data de previsão de conclusão da atividade. Além disso são filtradas as atividades do projeto identificado pelo código 1.

```
SELECT  phase.name,
        activity.description,
        activity.begin_date,
        activity.end_date_projected
FROM phase, activity
WHERE phase.id = activity.ref_phase AND
      ref_project = '1'
ORDER BY phase.id, activity.id;
```

Figura 46: Consulta SQL para listar as fases e atividades do estudo de caso

Para cada uma das atividades constantes na tabela apresentada anteriormente foram alocados itens do projeto. A tabela 11 apresenta os 16 itens definidos para as 9 atividades do projeto:

Tabela 11: Itens de atividades definidas para o estudo de caso

Atividade	Item	Data de início	Estimativa (dias)
Mapeamento dos requisitos	Mapeamento de requisitos	15-09-2010	10
Especificação do sistema	Especificação do sistema	27-09-2010	7

Atividade	Item	Data de início	Estimativa (dias)
Design do *layout* do portal	Definição do *layout* do portal	04-10-2010	3
	Criação da estrutura do portal	06-10-2010	4
Migração de dados	Criar o diagrama ER	11-10-2010	7
	Migração de dados	18-10-2010	7
Interface de recrutamento	Interface de oferecimento das vagas	18-10-2010	10
	Interface de cadastro de currículos	25-10-2010	10
	Interface de filtro de currículos	27-10-2010	7
Interface de agenciamento	Cadastro dos estagiários	25-10-2010	14
	Emissão de fatura	01-11-2010	14
Interface de emissão de boleto	Consulta financeira	22-11-2010	7
Integrações com ERP	Integração de cadastros	22-11-2010	14
	Integração com os boletos	29-11-2010	7
Documentação e testes	Execução dos testes	06-12-2010	10
	Manual do sistema	06-12-2010	10

Estes itens podem ser extraídos do modelo concreto através da consulta SQL apresentada na figura 47 onde são cruzadas as tabelas *phase, activity* e *backlog_item* do modelo e selecionadas as colunas descrição da atividade, título do item e estimativa de dias para conclusão do item. São filtrados itens de atividades pertencentes ao projeto identificado pelo código 1.

```
SELECT  activity.name,
        backlog_item.title,
        backlog_item.estimate
FROM phase, activity, backlog_item
WHERE activity.id = backlog_item.ref_activity AND
      phase.id = activity.ref_phase AND
      phase.ref_project = '1'
ORDER BY activity.id, backlog_item.id;
```

Figura 47: Consulta SQL para listar as atividades e itens do estudo de caso

Para cada um dos itens de desenvolvimento foram definidas tarefas e pessoas responsáveis por executá-las. A tabela 12 apresenta as 30 tarefas definidas para os 16 itens do projeto:

Tabela 12: Tarefas definidas para o estudo de caso

Item	Tarefa	Recurso 1	Recurso 2

Item	Tarefa	Recurso 1	Recurso 2
Mapeamento de requisitos	Mapear cadastros	Analista1	Analista2
	Mapear processos	Analista1	Analista2
	Mapear relatórios	Analista1	Analista2
Especificação do sistema	Diagramas de casos de uso	Analista2	Analista1
	Diagramas de sequência	Analista2	Analista1
	Diagramas de classes	Analista2	Analista1
Definição do *layout* do portal	Definir as cores	WebDesigner1	Programador1
	Definir os estilos	WebDesigner1	Programador1
	Montar os menus	WebDesigner1	Programador1
Criação da estrutura do portal	Definir disposição das notícias	WebDesigner1	Programador1
	Definir interface administrativa	WebDesigner1	Programador1
Criar o diagrama ER	Definir relacionamentos	Analista1	Analista2
	Modelar o banco de dados	Analista1	Analista2
Migração de dados	Criar *script* de migração	Analista1	Programador2
Interface de oferecimento das vagas	Cadastro de empresas	Programador3	Programador2
	Cadastro de vagas	Programador3	Programador2
Interface de cadastro de currículos	Interface cadastro de currículos	Programador3	Programador2
	Gerar currículos em PDF	Programador3	Programador2
Interface de filtro de currículos	Filtrar currículos disponíveis	Programador3	Programador2
	Gerar pacotes	Programador3	Programador2
	Enviar para empresas	Programador3	Programador2
Cadastro dos estagiários	Cadastro de estagiários	Programador1	Programador2
Emissão de fatura	Ajustar salários	Programador1	Programador2
	Gerar fatura de pagamento	Programador1	Programador2
Consulta financeira	Imprimir boleto em PDF	Programador2	Programador3
Integração de cadastros	Integrar com o cadastro de pessoas	Analista1	Programador2
Integração com os boletos	Integrar com o sistema financeiro	Programador2	Analista1
Execução dos testes	Criação dos testes	Documentador1	Documentador1
	Validação dos testes	Documentador1	Documentador1
Manual do sistema	Manual de usuário	Documentador1	Documentador1

A figura 48 apresenta como estas tarefas podem ser extraídas do modelo concreto

através de uma consulta SQL onde são cruzadas as tabelas *phase, activity, backlog_item, task* e *resource* do modelo e selecionadas as colunas título do item, descrição da atividade e par de recursos para atender a respectiva tarefa:

```
SELECT  backlog_item.title,
        task.description,
        resource1.name as resource1,
        resource2.name as resource2
FROM phase, activity, backlog_item, task,
     resource as resource1, resource as resource2
WHERE phase.id = activity.ref_phase AND
      activity.id = backlog_item.ref_activity AND
      backlog_item.id = task.ref_backlog_item AND
      task.programmer1 = resource1.id AND
      task.programmer2 = resource2.id
ORDER BY phase.id, activity.id, backlog_item.id, task.id;
```

Figura 48: Consulta SQL para listar os itens e as tarefas do estudo de caso

Após definidas as atividades, os itens e as tarefas, foi necessário planejar os ciclos e o lançamento das versões do projeto. Segue abaixo as versões e os ciclos de desenvolvimento definidos para o projeto:

- **Versão de demonstração**
  - **Ciclo de especificação:** Neste ciclo serão executados os itens de mapeamento de requisitos e especificação do sistema.
  - **Ciclo de *design* e *layout:*** Neste ciclo será definido e criado o *layout* do portal.
  - **Ciclo de migração:** Será criado o diagrama entidade relacionamento e feita a migração dos dados do sistema antigo.
- **Versão de validação:**
  - **Ciclo de criação de interfaces:** Este ciclo comportará a criação das interfaces de oferecimento de vagas, cadastro de currículos, cadastro de estagiários, emissão de faturas e consulta financeira.
  - **Ciclo de integração:** Neste ciclo será feita a integração com os cadastros e a integração com o sistema financeiro do ERP da instituição.
- **Versão final:**
  - **Ciclo de documentação:** Execução dos testes e homologação do sistema para uso e escrita do manual do sistema.

A tabela 13 apresenta os 6 ciclos citados acima com suas datas de início e previsão de conclusão definidas pelo gerente do projeto:

Tabela 13: Ciclos definidos para o estudo de caso

Versão	Ciclo	Data de início	Previsão conclusão
Versão demonstração	Ciclo de especificação	15-09-2010	01-10-2010
	Ciclo de *design* e *layout*	04-10-2010	08-10-2010
	Ciclo migração	11-10-2010	22-10-2010
Versão validação	Ciclo de criação de interfaces	18-10-2010	26-11-2010
Versão final	Ciclo de integração	22-11-2010	03-12-2010
	Ciclo de documentação	06-12-2010	15-12-2010

Estes ciclos podem ser extraídos do modelo concreto através da consulta SQL apresentada na figura 49 onde são cruzadas as tabelas *release* e *cycle* do modelo e selecionadas as colunas descrição da versão, descrição do ciclo, data de início do ciclo e data de previsão de conclusão do ciclo. São filtradas todas as versões do projeto identificado pelo código 1.

```
SELECT  release.name,
        cycle.description,
        cycle.begin_date,
        cycle.end_date_projected
FROM release, cycle
WHERE release.id = cycle.ref_release AND
      release.ref_project = '1'
ORDER BY release.id, cycle.id;
```

Figura 49: Consulta SQL para listar as versões e ciclos do estudo de caso

Para cada um dos ciclos constantes na tabela apresentada anteriormente foram elencados itens do projeto que deveriam ser cumpridos para que o ciclo fosse dado como concluído. A tabela 14 apresenta os 16 itens definidos para os 6 ciclos do projeto bem como o recurso chefe de cada um deles:

Tabela 14: Itens de ciclos definidos para o estudo de caso

Ciclo	Item	Recurso Chefe
Ciclo de especificação	Mapeamento de requisitos	Analista1
	Especificação do sistema	Analista2
Ciclo de *design* e *layout*	Definição do *layout* do portal	WebDesigner1
	Criação da estrutura do portal	Programador1
Ciclo migração	Criar o diagrama ER	Analista1

Ciclo	Item	Recurso Chefe
	Migração de dados	Analista1
Ciclo de criação de interfaces	Interface de oferecimento das vagas	Programador3
	Interface de cadastro de currículos	Programador3
	Interface de filtro de currículos	Programador3
	Cadastro dos estagiários	Programador1
	Emissão de fatura	Programador1
	Consulta financeira	Programador2
Ciclo de integração	Integração de cadastros	Analista1
	Integração com os boletos	Programador2
Ciclo de documentação	Execução dos testes	Documentador1
	Manual do sistema	Documentador1

Estes itens podem ser extraídos do modelo concreto através da consulta SQL apresentada na figura 50 onde são cruzadas as tabelas *release, cycle*, *cycle_backlog*, *backlog_item* e *resource* do modelo e selecionadas as colunas nome da versão, nome do ciclo, título do item e recurso chefe do respectivo ciclo. São filtrados itens de versões pertencentes ao projeto identificado pelo código 1.

```
SELECT  release.name,
        cycle.description,
        backlog_item.title,
        resource.name
FROM release, cycle, cycle_backlog, backlog_item, resource
WHERE release.id = cycle.ref_release AND
      cycle.id = cycle_backlog.ref_cycle AND
      cycle_backlog.ref_backlog_item = backlog_item.id AND
      cycle_backlog.chief_programmer = resource.id AND
      release.ref_project = '1'
ORDER BY release.id, cycle.id, backlog_item.id;
```

Figura 50: Consulta SQL para listar itens de ciclos do estudo de caso

A pós definida s as fases, atividad es, itens, ciclos e versões para o projeto “Portal de Empresas”, o próximo passo foi parametrizar os agentes de software para que o gerente consiga acompanhar em tempo real o andamento do projeto.

Na organização onde foi executada a validação do ambiente é utilizado o sistema Microsiga[7] como software de gestão de pessoal e de patrimônio. Devido a isso, o agente de recursos foi configurado para fazer a sincronização das informações de recursos humanos e

7 http://www.totvs.com

recursos físicos com este software diariamente às 2 horas da manhã. A figura 51 apresenta os parâmetros da tela de configuração do agente de recursos:

Figura 51: Configurações do agente de recursos do estudo de caso

O agente de itens foi instalado no ambiente de desenvolvimento da organização nas estações de trabalho das pessoas que se envolveram no projeto. Desde o início do projeto foi disponibilizado um repositório de controle de versões. Todos os arquivos gerados durante a criação do projeto desde os documentos de levantamento de requisitos, especificação do sistema, códigos fontes até o manual do sistema foram enviados para o repositório *subversion* utilizando a ferramenta de envio de arquivos do agente de itens. A figura 52 apresenta como ficou a configuração do agente de itens:

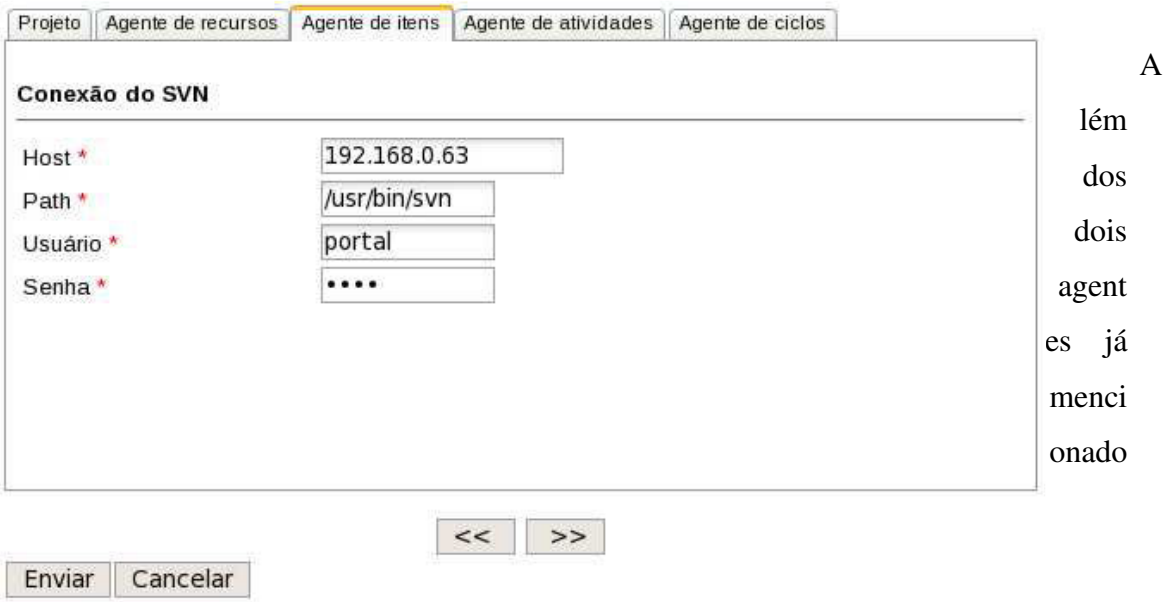


Figura 52: Configurações do agente de itens do estudo de caso

Além dos dois agentes já mencionados

s acima, os agentes de atividades e de ciclos também foram configurados pelo gerente do projeto. Eles receberam a configuração para serem executados a cada hora com o objetivo de manter o modelo concreto o mais próximo da realidade do projeto. Isso foi importante pois favoreceu ao gerente uma visão muito mais precisa do projeto, permitindo inclusive que ele pudesse tomar algumas medidas de correção durante o andamento do mesmo. Este e outros resultados obtidos serão detalhados na seção seguinte.

6.3. Resultados práticos

Como já foi comentado anteriormente, a validação do ambiente ocorreu entre os dias 15/11/2010 e 15/12/2010. Neste período foram lançadas várias versões contendo correções e melhorias na aplicação que estava sendo validada. E com o intuito de demonstrar o tamanho do projeto no qual a aplicação foi validada, a tabela 15 procura apresentar algumas variáveis que representam o tamanho do projeto que contém seis fases, nove atividades, dezesseis itens, trinta tarefas, seis ciclos e três versões. Também foram utilizados no projeto sete recursos humanos distribuídos entre dois analistas, um *web designer*, três desenvolvedores e um documentador que trabalharam um total de 1616 horas.

Tabela 15: Dados sobre o tamanho do estudo de caso

Característica	Total
Fases	3
Atividades	9
Itens	16
Tarefas	30
Ciclos	6
Versões	3
Recursos humanos	7
Horas trabalhadas	1616

Ao longo do período de monitoramento do projeto foram feitos 42 *commits* no repositório de controle de versões somando um total de 289 arquivos dentre códigos-fonte, arquivos de documentação, diagramas e imagens sem considerar os fontes do *framework* utilizado para o desenvolvimento do portal. E num determinado momento durante o monitoramento do projeto, mais especificamente no dia 23/11/2010, o *gantt* dinâmico apresentou ao gerente do projeto os seguintes alertas, conforme representados na figura 53:

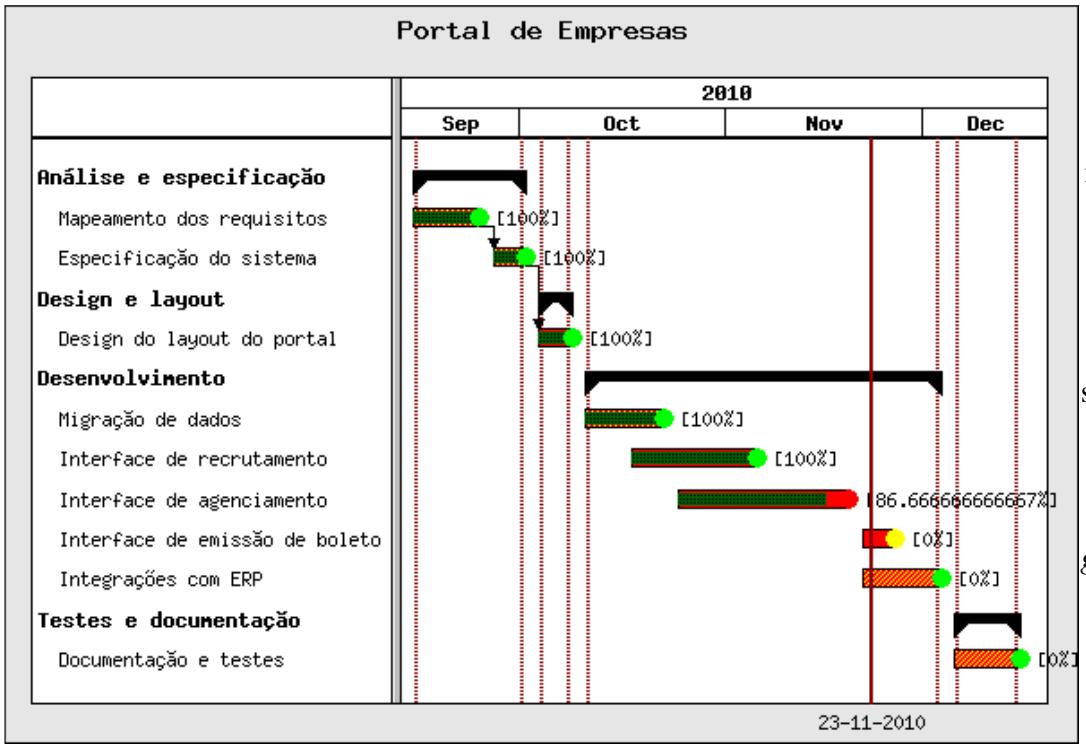


Figura 53: *Gantt* dinâmico do estudo de caso durante o monitoramento

Pelas informações apresentadas no *gantt* no dia 23/11/2010, o gerente do projeto se deparou com uma situação crítica e uma situação de alerta no projeto "Portal de Empresas". A situação crítica apareceu sobre a atividade "Interface de agenciamento" que estourou o prazo previsto de conclusão devido ao aumento do escopo de um item de desenvolvimento atrelado a esta atividade. A situação de alerta surgiu sobre a atividade "Interface de emissão de boleto" que embora ainda estava dentro do prazo de conclusão, foi detectado pelo agente de recursos a indisponibilidade de um desenvolvedor alocado nesta atividade em função dele estar fazendo um curso de aperfeiçoamento externo.

Ao final do projeto, mais especificamente no dia 15/12/2010, após execução das ações corretivas tomadas pelo gerente do projeto no dia 23/11/2010 sobre as atividades que apresentaram situações de alerta e pontos críticos (interface de agenciamento, interface de emissão do boleto e integrações com o ERP), o *gantt* dinâmico apresentou ao gerente do projeto a situação representada na figura 54:

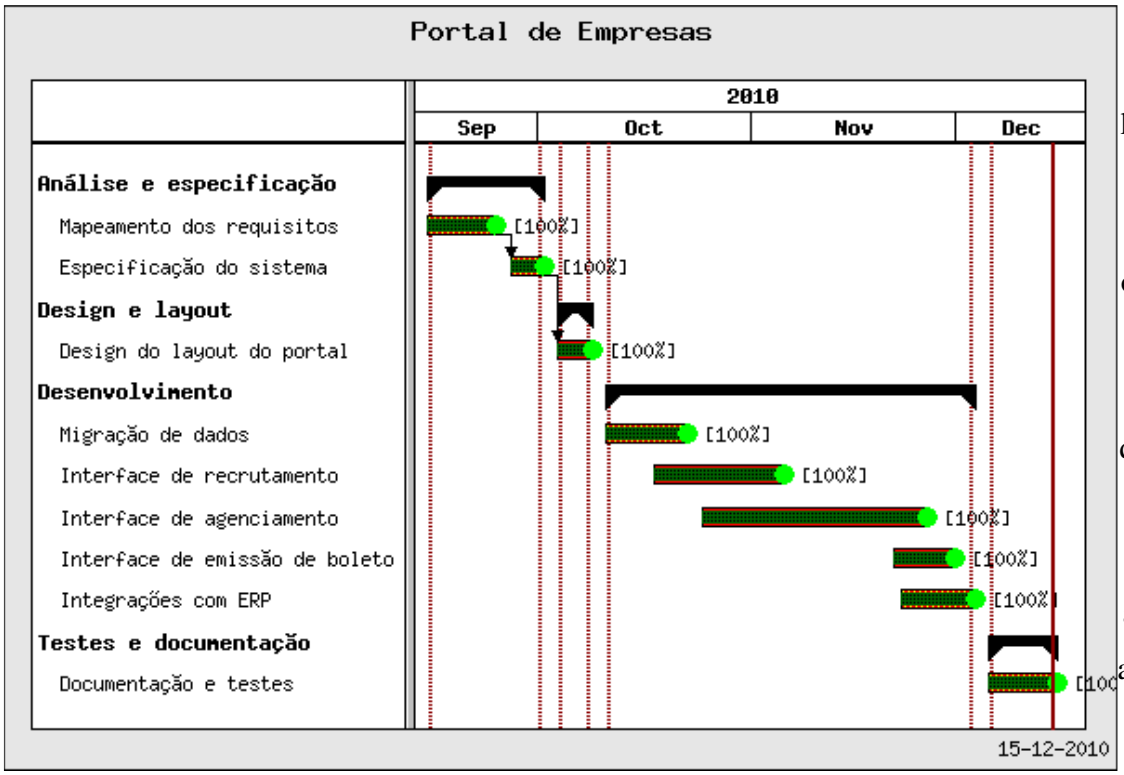


Figura 54: *Gantt* dinâmico do estudo de caso ao final do projeto

É possível observar que frente aos alertas e pontos críticos apresentados pelo *gantt* dinâmico ao gerente do projeto no dia 23/11/2010, imediatamente foi aumentado o prazo de conclusão da atividade de interface de agenciamento e pessoas foram realocadas paralelizando quase que totalmente as atividades de interface de emissão de boleto e integrações com o ERP. Diante destas ações tomadas pelo gerente, todas as atividades puderam ser cumpridas e o projeto pode ser entregue no dia 15/12/2010, conforme previsto em seu cronograma inicial.

Ao mesmo tempo que o gerente do projeto monitorava o gráfico de *gantt,* ele utilizou os recursos do relatório dinâmico disponível na aplicação, aplicando filtros e analisando se as previsões feitas sobre as atividades, itens e ciclos estavam se concretizando. A figura 55 apresenta um relatório gerencial extraído pelo gerente ao término do projeto onde são apresentadas as datas de concretização de cada atividade, item e ciclo.

# Relatório gerencial Projeto: Portal de Empresas								
Fase	**Atividade**	**Fim atividade previsto**	**Fim atividade**	**Item**	**Fim item**	**Ciclo**	**Fim ciclo previsto**	**Fim ciclo**
Análise e especificação	Mapeamento dos requisitos	2010-09-24	2010-09-25	Mapeamento de requisitos	2010-09-25	Ciclo de especificação	2010-10-01	2010-09-25
Análise e especificação	Especificação do sistema	2010-10-01	2010-09-30	Especificação do sistema	2010-10-01	Ciclo de especificação	2010-10-01	2010-09-25
Design e layout	Design do layout do portal	2010-10-08	2010-10-08	Definição do layout do portal	2010-10-07	Ciclo de design e layout	2010-10-08	2010-10-08
Design e layout	Design do layout do portal	2010-10-08	2010-10-08	Criação da estrutura do portal	2010-10-08	Ciclo de design e layout	2010-10-08	2010-10-08
Desenvolvimento	Migração de dados	2010-10-22	2010-10-22	Criar o diagrama ER	2010-10-13	Ciclo migração	2010-10-22	2010-10-22
Desenvolvimento	Migração de dados	2010-10-22	2010-10-22	Migração de dados	2010-10-22	Ciclo migração	2010-10-22	2010-10-22
Desenvolvimento	Interface de agenciamento	2010-11-19	2010-11-23	Cadastro dos estagiários	2010-11-04	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Interface de agenciamento	2010-11-19	2010-11-23	Emissão de fatura	2010-11-23	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Interface de emissão de boleto	2010-11-26	2010-11-23	Consulta financeira	2010-11-23	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Interface de recrutamento	2010-11-05	2010-11-04	Interface de oferecimento das vagas	2010-10-25	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Interface de recrutamento	2010-11-05	2010-11-04	Interface de cadastro de currículos	2010-10-30	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Interface de recrutamento	2010-11-05	2010-11-04	Interface de filtro de currículos	2010-11-04	Ciclo de criação de interfaces	2010-11-26	2010-11-23
Desenvolvimento	Integrações com ERP	2010-12-03	2010-12-02	Integração de cadastros	2010-11-30	Ciclo de integração	2010-12-03	2010-12-02
Desenvolvimento	Integrações com ERP	2010-12-03	2010-12-02	Integração com os boletos	2010-12-02	Ciclo de integração	2010-12-03	2010-12-02
Testes e documentação	Documentação e testes	2010-12-15	2010-12-14	Execução dos testes	2010-12-10	Ciclo de documentação	2010-12-15	2010-12-14
Testes e documentação	Documentação e testes	2010-12-15	2010-12-14	Manual do sistema	2010-12-14	Ciclo de documentação	2010-12-15	2010-12-14

Figura 55: Relatório gerencial do estudo de caso

Por fim, a utilização da aplicação desenvolvida no projeto “Portal de Empresas” se mostrou aplicável pois cumpriu seu objetivo principal que é manter o gerente do projeto sincronizado com o andamento das atividades em projetos que utilizam metodologias ágeis. Segundo o gerente de projetos da instituição externa onde a aplicação foi validada, a sua utilização trouxe uma segurança bem maior a sua gestão de projetos pois através dele foi possível detectar falhas mais rapidamente dando mais tempo para ajustar os impactos no cronograma, antes feitos somente através de conversas individuais ou reuniões de acompanhamento com a equipe.

6.4. Conclusões da seção

O presente capítulo apresentou a aplicação da solução desenvolvida ao longo deste trabalho em uma organização externa com o intuito de validar e verificar suas funcionalidades garantindo que ela atende às suas especificações e é plenamente funcional quando inserida em um ambiente real de uso.

Foi feita uma rápida contextualização do ambiente de execução da aplicação e um detalhamento completo do cenário utilizado para o estudo de caso. Juntamente com este detalhamento foram apresentadas as maneiras de como extrair tais informações do modelo concreto instanciado bem como suas principais características.

Ao final, foram apresentados os principais resultados alcançados com o estudo de caso que provou que a aplicação desenvolvida viabiliza o gerenciamento de projetos de desenvolvimento de software que utilizam metodologias ágeis.

7. CONCLUSÕES

Este capítulo apresenta um fechamento do presente trabalho. Com base no problema e na visão de solução apresentados ao longo do trabalho, naturalmente, surgem expectativas quanto aos potencias resultados do trabalho desenvolvido. Tais questões reforçam o caráter científico das atividades demandadas.

7.1. Resultados alcançados

A gerência de projetos de software tem o objetivo de atender as necessidades de um projeto. Desta forma, a gerência de projetos de software envolve o planejamento, o acompanhamento, o produto que está sendo desenvolvido e principalmente a metodologia que está sendo seguida. Normalmente os projetos de desenvolvimento de software possuem um enfoque gerencial que procura se estruturar através de processos e deposita importância fundamental no planejamento e no controle. Além disso, é imprescindível ter um enfoque técnico que se baseia na utilização de metodologias de desenvolvimento que agilizam uma implementação concreta e operacional. A realidade atual de desenvolvimento é composta por alterações a todo o momento com requisitos em constante evolução. Devido a isso, as metodologias ágeis têm conseguido espaço, pois têm uma certa facilidade em se adaptar a esse paradigma.

Para a criação de um ambiente de gerência de software para metodologias ágeis que permita uma gerência de projetos de software de forma personalizada e de acordo com as necessidades de um projeto específico a partir da combinação de conceitos e características de metodologias ágeis, foi necessário o desenvolvimento de um modelo abstrato que contemplasse todos os aspectos relacionados à gerência de projetos e metodologias ágeis, tais como, gerência de atividades e recursos, controle de itens e ciclos de desenvolvimento, dentre outros. A criação de um metamodelo que suporte estes aspectos permitiu a criação de uma aplicação que explora todo o seu potencial para a gerência de software em metodologias ágeis.

A maior contribuição deste trabalho foi desenvolver um modelo voltado a gerência de software em metodologias ágeis comprovando sua eficiência através do desenvolvimento de uma aplicação dinâmica e que faça uso de todos os conceitos embutidos no metamodelo proposto.

O metamodelo proposto explora um conjunto de características e conceitos das três metodologias ágeis de desenvolvimento de software mais utilizadas atualmente e do metamodelo de gerência de projetos mais conhecido. Tratam-se das metodologias ágeis XP,

Scrum e FDD e do metamodelo PMBOK de gerência de projetos. Foi pré-requisito ao projetar o metamodelo resultante deixá-lo dinâmico e modular de forma a permitir que ele possa ser instanciado e gerar um modelo concreto de acordo com as características estabelecidas pelo gerente. Esta dinamicidade dada ao modelo foi o grande diferencial do metamodelo proposto, pois em nenhum dos outros trabalhos avaliados isso é possível.

A aplicação desenvolvida implementa todas as características e conceitos suportados pelo metamodelo proposto. Ela permite que o modelo abstrato seja instanciado e de acordo com o que foi selecionado, disponibiliza interfaces dinâmicas para o cadastro de atividades, recursos, itens e ciclos de desenvolvimento e outras informações sobre o projeto. Além disso, a configuração dos agentes de software definidos para manter o modelo concreto sincronizado disponibilizam uma série de parâmetros que permitem que a aplicação possa ser instalada em ambientes reais de produção.

Após ter instanciado o modelo abstrato, feito o planejamento inicial do projeto e configurado os agentes de software, o gerente pode usufruir dos recursos disponibilizados pela aplicação tais como a visualização de um *gantt* dinâmico com apresentação de pontos críticos do projeto além de permitir a extração de relatórios gerenciais sobre qualquer dimensão do modelo, possibilitando aplicar diversos filtros. Isso traz um grande ganho em termos de gerência de projetos de software pois permite um acompanhamento mais efetivo do andamento do projeto e possibilita detectar falhas de planejamento mais rapidamente, dando mais tempo para ajustar possíveis impactos no cronograma juntamente com a equipe disponível.

O presente trabalho buscou desenvolver um modelo para gerência de projetos de desenvolvimento de softwares que utilizam metodologias ágeis. Além disso, foi criada uma aplicação que permite ao gerente do projeto ter uma visão precisa do andamento do mesmo. Já existem aplicações estabelecidas no mercado para gerência de projetos, porém não trabalhando de forma integrada com o ambiente. O que conferiu tal flexibilidade à aplicação é justamente a arquitetura de agentes que a deixou iterativa de forma a dar ao gestor uma visão macro e precisa do projeto. Desta forma, trabalhos futuros podem adicionar novas metodologias ao metamodelo proposto sem a necessidade de reescrever os recursos já existentes na aplicação.

7.2. Trabalhos futuros

O trabalho proposto atingiu plenamente as metas que haviam sido traçadas durante seu planejamento, porém no decorrer do seu desenvolvimento e durante sua validação, detectou-

se que alguns novos recursos poderiam ser adicionados, fazendo com que a aplicação fique ainda mais completa e abrangente. Dentre eles, merecem destaque:

- Desenvolver uma integração do gráfico de *gantt* dinâmico gerado na aplicação desenvolvida com uma ferramenta externa de gerência de projetos tais como o GanttProject[8] ou o PhpProjekt[9] a fim de permitir uma manipulação maior dos dados do projeto com os recursos que estas ferramentas possam oferecer;
- Ampliar o metamodelo proposto adicionando outras metodologias ágeis não contempladas nesta versão do modelo;
- Ampliar o metamodelo proposto adicionando outros métodos de gerência de projetos não contemplados nesta versão do modelo;
- Integrar através de *plugins* o agente de itens de desenvolvimento a uma IDE de programação a fim de permitir que o desenvolvedor tenha agilidade no momento de enviar uma alteração;
- Criar outros agentes de software que possam monitorar aspectos do modelo que não foram previstos na aplicação desenvolvida.
- Desenvolver uma ontologia para a representação do modelo abstrato proposto com o intuito de representar todos os seus conceitos de forma clara e objetiva;

8 http://ganttproject.biz
9 http://www.phpprojekt.com

REFERÊNCIAS

ABRAHAMSSON, P., SALO, O., RONKAINEN, J., WARSTA, J., Agile Software Development Methods: Review and Analysis. VTT, Publication, Finland, 2002.

ABRAHAMSSON, Pekka, WARSTAM Juhani, SIPONEN, Mikko T., RONKAINEN, Jussi. New Directions on Agile Methods: A Comparative Analysis. International Conference on Software Engineering, IEEE Computer Society, 2003.

AGILE MANIFESTO, Manifesto ágil. Disponível em: <http://agilemanifesto.org>, Acesso em 28/05/2010.

AMBLER, S. Modelagem ágil: práticas eficazes para a Programação Extrema e o Processo Unificado. Porto Alegre: Bookman, 2004.

BECK, Kent, ANDRES, Cynthia. Extreme Programming Explained. Publisher: Addison-Wesley Professional; 2 edition, 2004.

BOEHM, B. A View of 20th and 21st century software engineering, ICSE, 2006

BOEHM, Barry, TURNER, Richard. Management Challenges to Implementing Agile Processes in Traditional Development Organization, 2005.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A.: Tropos: An Agent-Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems. Klumer Academic Publishers, 2004.

CHANG, Wei-chun, WU, Ching-seh, SETHI, Ishwar K. A Metric-Based Multi-Agent System for Software Project Management, International Conference on Computer and Information Science, IEEE/ACIS, 2009.

CHAVDA, K. F. Anatomy of Web Service, Journal of Computing Sciences in Colleges, Kennesaw State University, January 2004.

CHIN, Gary. Agile project management: how to succeed in the face of changing project requirements. NY: Amacon, 2004.

COCKBURN, Alistair. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley Professional, 2004.

COHEN, Dennis. J.; GRAHAM, Robert. J. Gestão de projetos: MBA executivo. Trad. Afonso Celso da Cunha Serra. Rio de Janeiro: Campos, 2002.

DALL'OGLIO, Pablo. Um sistema multi-agente para a gestão da mudança de requisitos de software, Unisinos, São Leopoldo, 2010.

DELOACH, S. A., WOOD, M. F. An Overview of the Multiagent Systems Engineering Methodology. In Proceedings of the First International Workshop on Agent-Oriented Software Engineering, Limerick, Ireland, 2001.

DSDM, Dynamic Systems Development Method. Disponível em: <http://www.dsdm.org> Acesso em 28/05/2010.

FAGUNDES, Priscila Basto, DETERS, Janice Inês, SANTOS, Sandro da Silva. Comparação entre os processo dos métodos ágeis: XP, Scrum, FDD e ASD em relação ao desenvolvimento iterativo incremental. Atualidades Tecnológicas para Competitividade Industrial, Florianópolis, 2008.

FDD, Feature Driven Development. Disponível em: <http://www.featuredrivendevelopment.com> Acesso em 28/05/2010.

FININ, T., LABROU, Y., MAYFIELD, J. KQML as an agent communication language, Computer Science Department. University of Maryland Baltimore County. Baltimore, USA, 1993

FOWLER, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

GERMAIN, E., ROBILLARD, P., Engineering-based Processes and Agile Methodologies for Software Development: a Comparative Case Study, The Journal of Systems and Software, Elsevier, February 2005

HIGHSMITH, Jim. Agile software development ecosystems. Boston: Addison-Wesley, 2002.

KEE, Woi Hin, Future Implementation and Integration of Agile Methods in Software Development and Testing. International Conference on Software Engineering, IEEE Computer Society, 2006.

KERZNER, H.; Project Management – A Systems Approach to Planning, Scheduling, and Controlling, New York NY, John Willey & Sons., 2002

KOCH, Al. São ágeis e compatíveis com o PMBOK? Pittsburgh Project Management Institute. 2004. Disponível em: <http://www.pittsburghpmi.org/documents/meetings/presentations/AgileManifesto-PMBOK.pdf>. Acesso em: 30/05/2010.

KOSKELA, J., Software Configuration Management in Agile Methods, VTT Publication, Finland, 2003.

KUMAR, K.; WELKE, R. J., Methodology engineering: A proposal for situation specific methodology construction in Challenges and strategies for research in systems development, W. W. Cotterman and J. A. Senn, Eds. New York: John Wiley & Sons, 1992, pp. 257-269.

MARTIN, J. Rapid Application Development. Macmillan Publishing Co., 1991.

MARTINS, J.C.C., Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML, 2a edição revisada, Rio de Janeiro: Brasport, 2005.

MOHAN, P. E. Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective, 2005.

NANDHAKUMAR, J.; AVISON, J., The fiction of methodological development: a field study of information systems development, Information Technology & People, vol. 12, pp. 176-191, 1999

NESMA. Netherland Software Metrics Users Association. Qualidade na produção de software, 2005. Disponível em:

<www.proqualiti.org.br/revista/revista_ProQualiti_maio2005.pdf>. Acesso em 25/05/2010.

ODELL, J.; PARUNAK, H. V. D.; BAUER, B.: Extending UML for Agents. In Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, 2000.

PADGHAM, L; WIKIKOFF, M.: Prometheus: A Practical Agent-Oriented Methodology. Chapter 5 in Agent-Methodologies, edited by B. Henderson-Sellers and P.Giorgini, Idea Group, 2005.

PAETSCH, Frauke. EBERLEIN, Armin. MAURER, Frank. Requirements Engineering and Agile Software Development. International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. 2003.

PALMER, Stephen R., FELSING, John M. A Practical Guide to Feature Driven Development. Prentice Hall, 2002.

PARAISO, E. C. Concepção e Implementação de um Sistema Multiagente para Monitoração e Controle de Processos Industriais, Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial), CEFET - PR, 1997.

PMBOK – A Guide to the Project Management Body of Knowledge, 2008.

PMI - Project Management Institute. Disponível em: <http://www.pmi.org/info/default.asp>. Acesso em: 20/05/2010.

PRADO, Darci Santos do. Planejamento e controle de projetos. Belo Horizonte: Desenvolvimento Gerencial, 1998.

PRESSMAN, R. Engenharia de Software. McGraw-Hill, 2001.

QUMER, A., HENDERSSON-SELLERS, B., Comparative Evaluation of XP and Scrum Using the 4D Analytical Tool (4-DAT), In Proc. of the European and Mediterranean Conference on Information Systems (EMCIS), Spain, 2006.

RAO, A. S., GEORGE, M. P. BDI Agents: From Theory to Practice, In Proccedings of the First International Conference on Multi-Agent Systems (ICMAS 1995), São Francisco, MIT Press, pp. 312-319, 1995.

ROSITO, M. C, CALLEGARI, D. A, BASTOS, R. M. Gerência de Projetos e Processos de Desenvolvimento de Software: uma proposta de integração. In: SBSI 2008 - Simpósio Brasileiro de Sistemas de Informação, Rio de Janeiro, 2008.

RUSSEL, S., NORVIG, P. Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995.

SCHNEIDER, Ricardo Luiz. Fundamentos da Engenharia de Software. Disponível em: <http://www.dcc.ufrj.br/~schneide/es/2000/1/>. Acesso em: 21/05/2010.

SCHWABER, K., Agile Project Management with Scrum. Microsoft Press, 2004.

SCHWABER, K., BEEDLE, Mike. Agile Software Development with Scrum. Prentice Hall, 2002.

SLINGER, M. The software project manager's bridge to agility. Boston: Peterson Education, 2006.

SOMMERVILLE, I. Engenharia de software. São Paulo: Addison-Wesley, 2003.

STAPLETON, Jennifer. DSDM: A framework for business centered development. Addison Wesley Professional, 1997.

TAROMIRAD, Masoumeh, RAMSIN, Raman. An Appraisal of Existing Evaluation Frameworks for Agile Methodologies, International Conference and Workshop on the Engineering of Computer Based Systems, IEEE Computer Society, 2008.

THOMSETT, R. Radical Project Management. NJ: Prentice Hall, 2002.

VALERIANO, Dalton L. Gerência em Projetos – Pesquisa, Desenvolvimento e Engenharia. São Paulo: Makron Books, 1998.

VAZQUEZ, C.E., SIMÕES, G.S., ALBERT, R.M., Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software, 3a edição, São Paulo: Editora Érica, 2005.

VIEIRA, M.F., Gerenciamento de Projetos de Tecnologia da Informação, Rio de Janeiro: Editora Elsevier – Campus, 2003.

WALDROP, M. Mitchell. Complexity: The Emerging Science at the Edge of Order and Chaos. Simon & Schuster, 1992.

WILLIAMS, L., KERBS, W., LAYMAN, L., ANTON, A., Toward a Framework for Evaluating Extreme Programming, In Proc. of the 8th International Conference on Empirical Assessment in Software Engineering (EASE 04), Edinburgh, 2004.

WOOLDRIDGE, Michael. Intelligent Agents, In: A Modern Approach to Distributed Artificial Intelligence, G. Weiss (ed.), The MIT Press, 1999.

WU, Ching-seh, Software Metric-based Intelligent Agents for Software Project Risk Prevention, Journal of Crisis Management, pp10-16, Volume1, Number 2, June 2004

XP, Extreme Programming. Disponível em: <http://www.extremeprogramming.org> Acesso em: 22/05/2010.