Universidade Federal do Estado do Rio de Janeiro
Escola de Informática Aplicada
Bacharelado de Sistemas de Informação

Hive
Um processo de software colaborativo

Autor: Leonardo Wallace Ramos Couto
Orientador: Gleison Santos

Novembro/2012

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Autor: Leonardo Wallace Ramos Couto

Orientador: Gleison Santos

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

_____

_____

O(s) autor(es) deste Projeto autoriza(m) a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardados os direitos autorais conforme legislação vigente.

Rio de Janeiro, _____de_____ de _____.

_____

Leonardo Wallace Ramos Couto

Conteúdo

Índice de ilustrações

Resumo

Cada vez mais as equipes de desenvolvimento e as organizações se tornam mutáveis, tornando os processos e metodologias desatualizados para suas necessidades. Em parte, isso pode ser atribuído à falta de participação dos profissionais na definição do processo de trabalho e na sua evolução contínua.

Uma da formas de se garantir que o processo evolua de acordo com os envolvidos e as suas necessidades é a existência de colaboração entre profissionais a fim de tornar os processos utilizados e o conhecimento gerado de fácil acesso para os participantes.

Esta monografia apresenta o *Hive*, umprocesso de software colaborativo que fornece padrões, boas práticas e diretrizes que auxiliam a adaptação do próprio processo ao perfil da equipe, da organização e na gestão do conhecimento.

1. Introdução

1.1 Motivação

O uso de processos de software auxilia equipes a construírem produtos de software com boa qualidade. A definição desses processos precisa estar adequada para a sua realidade e também refletir boas práticas de engenharia de software. Este processo deve ser adaptado a todos os segmentos de desenvolvimento, considerando o tamanho das equipes, o porte dos projetos e tecnologias envolvidas.

Nós vivemos em um mundo caracterizado pela evolução e estamos cada vez mais conscientes que esses processos evolutivos são críticos para a inovação tecnológica. Isso é particularmente verdade para sistemas complexos de software, pois esses sistemas não necessariamente existem em um contexto tecnológico sozinhos, ao invés disso são incorporados com organizações humanas dinâmicas (Curtis, 1988).

A definição de tais processos não é trivial. Desta forma, uma má escolha de uma metodologia de desenvolvimento de software pode ocasionar em um processo não condizente com a realidade da empresa e gerar resistência das pessoas que teoricamente deveriam segui-los. Mesmo o processo sendo bem definido, provavelmente não é aplicável a todos os contextos, sendo desejável a definição de diferentes versões. A descrição dos processos deve ser apoiada por diretrizes e boas práticas de forma integrada. Para que o uso do processo seja efetivo, as pessoas envolvidas devem conhecê-lo, utilizá-lo e, sobretudo contribuir para a sua evolução.

1.2 Problema

Grande parte das organizações que desenvolvem software tenta padronizar os seus processos para maximizar o desempenho de sua equipe e melhorar as suas estimativas e a qualidade do software produzido. Um problema comum após implantações de processos de desenvolvimento são os profissionais que não seguem o que foi definido. Uma forma de tentar minimizar uma situação como esta é incentivá-los a participarem diretamente de todas as fases da modelagem e descrição do processo ou adoção de

uma metodologia. Espera-se com isso que os processos sejam mais fiéis à realidade, pois serão os próprios envolvidos que irão utilizá-lo.

Um fator que pode fazer com que o processo caia em desuso é a identificação de novas necessidades da organização. Sendo assim, o processo de desenvolvimento deve ser flexível, já que as empresas também são. É necessário, portanto, manter o processo atualizado com as necessidades da organização e de seus profissionais fazendo com que eles colaborem para a sua atualização e melhoria.

Organizações têm obtido sucesso utilizando ferramentas colaborativas incentivando que todos participem da evolução de processos internos (Baldwin, 2002). Uma das vantagens de uma ferramenta como essa é o auxílio na colaboração da equipe. Dessa forma, as versões do processo podem ser geradas mais rapidamente, facilitando a sua adaptação às novas necessidades.

As empresas possuem diferentes necessidades, logo é importante que um processo de software seja flexível o suficiente para se adaptar a todas. Metodologias, em geral, possuem um foco específico, mas não fornecem uma estrutura para que elas mesmas evoluam de acordo com o seu uso ou a evolução da própria organização que a utiliza.

Um ponto importante, mas muitas vezes negligenciado pelas equipes de desenvolvimento, é a colaboração na concepção e implementação da solução e, principalmente, no compartilhamento do conhecimento adquirido ou gerado. Equipes segmentadas possuem elos fracos e acabam por segregar conhecimento dentro do próprio ambiente de trabalho, comprometendo a evolução dos profissionais e do time como um todo. Por isso é importante que a colaboração esteja presente não apenas na definição e evolução do processo em si, mas também durante a sua execução.

1.3 Solução proposta

O *Hive* tem como objetivo descrever um processo de software colaborativo que que permita a interação das pessoas envolvidas nas atividades englobadas e a flexibilização de suas descrições de acordo com o contexto abordado, além da associação de outras fontes de informação e compartilhamento de conhecimento.

1.4 Estrutura do texto

Esta monografia está organizada em 5 capítulos, incluindo esta introdução. No Capítulo 2 (Revisão da literatura), serão abordados conceitos que serão necessários para o entendimento do processo. Além disso, serão apresentados os processos de software, metodologias e conjunto de práticas utilizadas atualmente, muitos desses utilizados como base para o *Hive*.

Após serem abordadas as bases para entendimento, no Capítulo 3 (*Hive*), o processo será descrito textualmente e em forma de modelagem utilizando *BPMN – Business Process Modeling and Notation* (OMG.org, 2011), assim como os papéis envolvidos e artefatos gerados.

Com o processo descrito, o capítulo 4 (Desenvolvimento) apresenta uma implementação das etapas do processo no contexto de um exemplo de uso do *Hive*. Por fim, o Capítulo 5 apresenta as considerações finais.

2. Revisão da Literatura

2.1 Processo de desenvolvimento de software

Processo de desenvolvimento de software, também conhecido como ciclo de desenvolvimento de software, é uma estrutura definida para o desenvolvimento de um produto de software, termos similares incluem "ciclo" e "processo de software". Há diversos modelos para esses processos, cada um descrevendo sua abordagem para uma variedade de tarefas ou atividades que ocorrem durante o processo.

Alguns consideram o modelo de ciclo de vida um termo mais genérico e processo de desenvolvimento de software um termo mais específico. Por exemplo, há diversos processos de desenvolvimento de software que se encaixam no modelo de ciclo de vida espiral (Elliott, 2004).

2.2 Interações humanas e colaboração

Colaboração, no aspecto da tecnologia da informação parece ter muitas definições. Entender as diferenças nas interações humanas é necessário para ter certeza que as tecnologias apropriadas sejam utilizadas para atender as necessidades dessas interações. Há três meios primários de interação humanas (Pinnadyne, 2009):

- Interação Conversacional - É a troca de informações entre dois ou mais participantes onde o principal propósito da interação é a descoberta ou construção de relacionamento. Não há nenhuma entidade centralizadora, mas sim uma troca livre de informação sem restrições definidas (Pinnadyne, 2009).
- Interação Transacional - Envolve a troca de transações onde a principal função da entidade Transacional é alterar o relacionamento entre os participantes. A transação restringe ou define o novo relacionamento. Por exemplo: um participante troca dinheiro por bens e se torna um cliente (Pinnadyne, 2009).

- Interação Colaborativa - Em uma interação colaborativa a principal função do relacionamento dos participantes é alterar a entidade Colaboração. Por exemplo: o desenvolvimento de uma idéia, a criação de um design, atingir um objetivo comum. Tecnologias de colaboração reais disponibilizam a funcionalidade para os participantes para alterarem um resultado? gravar ou gerenciar documentos, organizar discussões, auditar históricos e outros mecanismos feitos para capturar as contribuições dos participantes em um conteúdo gerenciável são exemplos de tecnologias de colaboração (Pinnadyne, 2009).

2.3 Metodologia

Por definição, metodologia é um conjunto de práticas, técnicas, procedimentos e regras utilizadas por alguém que trabalha em uma disciplina (PMBoK 4th Edition, 2009).

De forma genérica, uma metodologia não descreve métodos específicos, apenas a natureza e tipos de processos para serem seguidos em um dado procedimento ou para atender um objetivo. Esses processos constituem um framework genérico construtivo, e serão subdivididos em subprocessos, combinados ou terão sua sequência alterada (Katsicas, 2009).

De forma mais detalhada, uma metodologia é um estrutura conceitual que é coerente e internamente consistente, e por ser composta de um modelo e um princípio aceito, é utilizada para analisar ou organizar informação para diferenciar ou razoavelmente incluir um princípio central comum que permitirá o desenvolvimento ou validação lógica de postulados teóricos (Herrman, 2009).

2.4 Metodologias Ágeis

As metodologias ágeis de desenvolvimento surgiram em meio ao ano de 2001 por meio de dezessete especialistas, que formaram a *Agile Software Development*

Alliance (Agile Manifesto, 2001), a entidade possui mais de quatro mil membros pelo mundo e apóia a todos que exploram e aplicam métodos ágeis de desenvolvimento.

Os métodos ágeis são regidos pelo manifesto ágil, ou Manifesto for *Agile Software Development* (Agile Manifesto, 2001). Além disso, também possui alguns princípios que servem de base para todas as suas práticas (Agile Manifesto, 2001).

2.4.1 Manifesto Ágil

O manifesto ágil define as diretrizes das metodologias ágeis, que são colocadas em contraponto aos das metodologias tradicionais, como podemos ver abaixo (Agile Manifesto, 2001):

- Indivíduos e interações mais que processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

De acordo com os autores, o fato de as metodologias tradicionais estarem em contraponto com os das metodologias ágeis não significa que serão desprezados.

No manifesto ágil encontra-se a síntese das práticas seguidas nas metodologias ágeis (Agile Manifesto, 2001):

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando à vantagem competitiva para o cliente;
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho;

- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
- Software funcionando é a medida primária de progresso;
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
- Contínua atenção a excelência técnica e bom design aumentam a agilidade;
- Simplicidade a arte de maximizar a quantidade de trabalho não realizado é essencial;
- As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis;
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo;

2.4.2 Scrum

O Scrum é um framework que segue os princípios das práticas ágeis. Baseia-se em ciclos de 30 ou menos dias chamados *Sprints*, que atendem entregas definidas. Um esquema do Scrum pode ser visto abaixo na Figura 1. É composto pelas Equipes Scrum, seus papéis associados, eventos, artefatos e regras. Cada componente no framework serve a um propósito específico e é essencial para a utilização e sucesso do Scrum (Schwaber & Sutherland, 1991-2011).

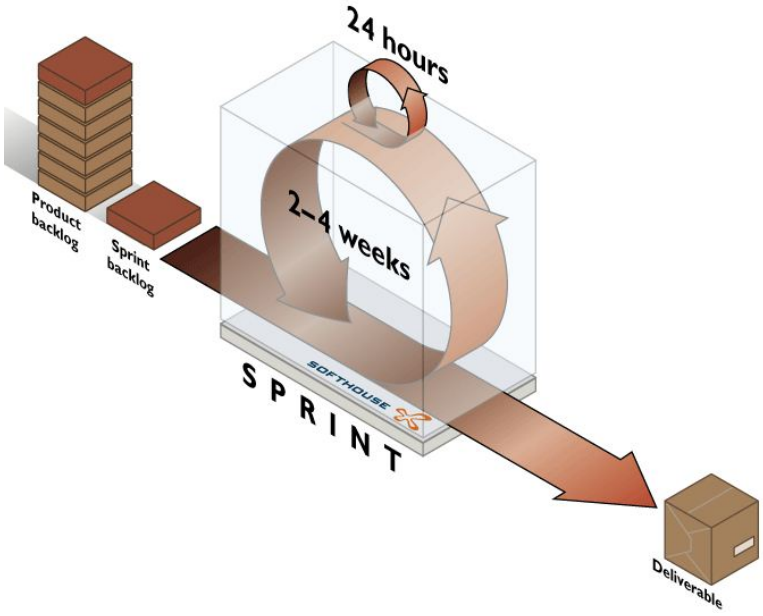

Figura 1- Etapas do processo do Scrum

Os papéis desempenhados pelos envolvidos são:

- Equipe Scrum: profissionais envolvidos no desenvolvimento da solução, normalmente composta por cinco a nove pessoas;
- *Product Owner*: normalmente representado pelo cliente, é o responsável pela visão do negócio e definição de prioridades no *Product Backlog*;
- *Scrum Master*: seu papel é facilitar ao máximo o trabalho da Equipe, assegurar a prática correta da metodologia e a verificação do andamento das atividades;

O *Product Owner* define as funcionalidades ou mudanças do produto no *Product Backlog*. Esta lista reflete as necessidades e prioridades do cliente ou do nicho em que ele atua.

Em um *Sprint*, os itens que devem ser entregues são tratados no *Sprint Backlog*. Nesta fase as atividades se tornam responsabilidade da equipe.

São realizadas reuniões diárias para acompanhamento junto ao *Scrum Master*, para que seja verificado o que foi feito no dia anterior, o que se pretende fazer no dia e se há algum empecilho na execução de alguma tarefa.

Ao final do *Sprint*, a Equipe demonstra o produto para o *Product Owner*, que em conjunto revisará os resultados obtidos. Estando os itens do *Product Backlog* prontos, inicia-se um novo *Sprint*.

2.4.3 Extreme Programming (XP)

O XP (Extreme Programming, 2009), assim como o Scrum segue os princípios das práticas ágeis além da ênfase na simplicidade, gerência de requisitos e refatoração de código. É mais adequado para equipes de até doze desenvolvedores e projetos com requisitos vagos. Um esquema do XP pode ser visto abaixo na Figura 2.

Algumas peculiaridades do XP devem ser frisadas, como, por exemplo, a codificação em dupla (programação por pares ou *Pair Programming*). De acordo com estudos realizados pelos idealizadores, o desempenho de um desenvolvedor sozinho é bastante semelhante ao de dois atribuídos para a mesma implementação, porém por haver um Condutor, que conduz a codificação e um Navegador, que o auxilia, espera-se que a qualidade do código seja superior (XProgramming, 1997-1999). Outro ponto é o foco na melhoria contínua do software, onde é dito que qualquer desenvolvedor que se depare com um código que pode ser refatorado deve fazê-lo, porém sem alterar o seu funcionamento. Uma das recomendações da XP é que os participantes desenvolvam orientados a teste (*Test Driven Development*) (Beck, 2003), ou seja, implementar testes antes mesmo implementação da funcionalidade.

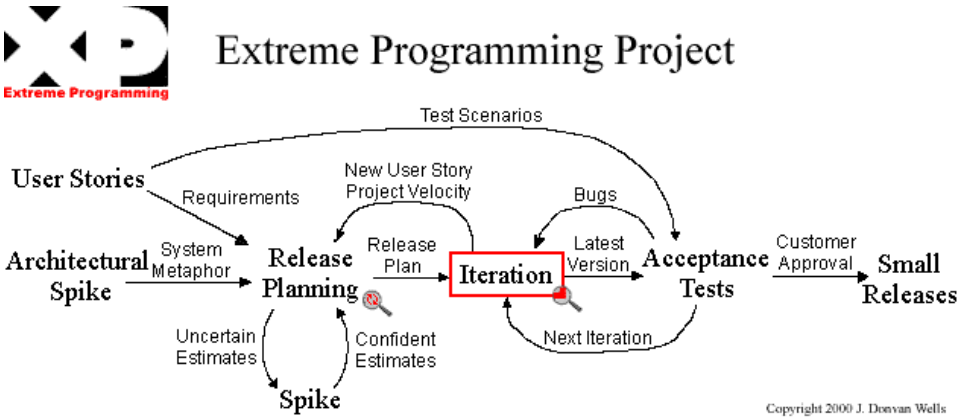

Figura 2 - Extreme Programming em um diagrama com notação informal

Além dessas características próprias do XP, há outras práticas que fazem parte dos seus princípios:

- Jogo de Planejamento (*Planning Game*): As interações do projeto são semanais e realizadas em uma reunião onde o cliente e os desenvolvedores participam;

- Pequenas Versões (*Small Releases*): gerar pequenas versões funcionais do projeto para facilitar a aceitação por parte do cliente;
- Metáfora (*Metaphor*): utilizar uma comunicação simples entendendo a realidade do cliente;
- Projeto Simples (*Simple Design*): implementar códigos simples, atendendo as necessidades daquela versão;
- Time Coeso (*Whole Team*): o cliente é também um componente da equipe.
- Testes de Aceitação (Customer Tests): o cliente realiza testes em conjunto com os analistas para validar uma solução desenvolvida;
- Ritmo Sustentável (*Sustainable Pace*): realizar horas extras apenas quando trouxerem produtividade, procurar uma rotina saudável de trabalho;
- Reuniões em pé (*Stand-up Meeting*): realizar reuniões em pé, para evitar a perda de foco e abordar apenas tarefas que foram realizadas e pendentes;
- Posse Coletiva (*Collective Ownership*): o código-fonte não possui dono, logo não é necessária autorização para alterá-lo;
- Padrões de Codificação (*Coding Standards*): definir padrões de codificação que devem ser seguidos por toda a equipe;
- Integração Contínua (*Continuous Integration*): sempre integrar novas funcionalidades o mais rápido possível à versão atual do sistema;

Os papéis envolvidos no XP são (Extreme Programming, 2009):

- Gerente de Projeto: responsável pelos assuntos administrativos do projeto, pela participação do cliente, além de filtrar demandas não relevantes aos desenvolvedores;
- *Coach*: responsável pelas questões técnicas do projeto e por garantir as práticas da XP;
- Analista de Teste: responsável por garantir a qualidade do software através de testes escritos;
- Redator Técnico: responsável pela documentação do projeto;
- Desenvolvedor: responsável em analisar, projetar e codificar o sistema;

2.5 Metodologias Tradicionais

As metodologias ditas tradicionais são nada mais do que "não-agéis". Porém, isso não significa que possuam desempenho inferior às anteriormente citadas. Cada processo de software possui seu espaço em diferentes organizações de acordo com as necessidades. A seguir são descritos alguns modelos e implementações, de onde se derivaram a grande maioria das metodologias tradicionais atuais.

2.5.1 Desenvolvimento em cascata

No desenvolvimento em cascata o projeto segue uma série de passos ordenados e ao final de cada fase é realizada uma revisão e o desenvolvimento não prossegue até que o produto gerado esteja atendendo os requisitos definidos. A Figura 3 apresenta um esquema desse modelo, originado de indústrias de bens manufaturados e que ganhou muito espaço quando ainda não havia processos de desenvolvimento de software formalmente definidos.

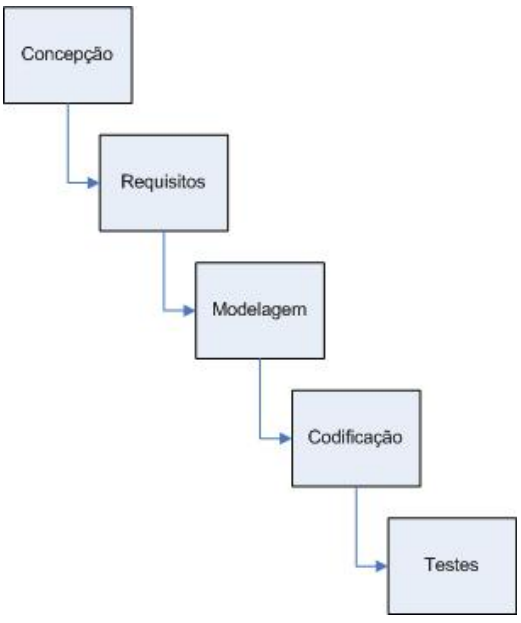

Figura 3 - Processo de desenvolvimento em cascata

De acordo com os entusiastas deste modelo, mais tempo gasto nas fases iniciais do projeto significa menos tempo perdido nas fases finais. E um requisito não

levantado ou uma correção ignorada até a fase de codificação custará de cinquenta a duzentas vezes mais para ser corrigida posteriormente (McConnell, 1996).

2.5.2 Prototipagem Evolutiva

A metodologia de prototipagem evolutiva prevê a criação de um protótipo e ciclos de refinamento dele antes que se inicie o ciclo de desenvolvimento da aplicação. A Figura 4 apresenta um esquema dessa metodologia. A implementação da funcionalidade só se inicia quando o protótipo é definitivamente aceito pelo cliente. Alguns dos seus princípios são (Centers for Medicare & Medicaid Services (CMS) Office of Information Service, 2008):

- Tentar reduzir o risco do projeto, subdividindo o projeto em segmentos menores para facilitar o desenvolvimento;
- O cliente é envolvido durante o desenvolvimento do produto, reduzindo os riscos de haver incoerências no resultado final esperado;
- *Mockups* de menor escala são desenvolvidos, seguidos de interações com o cliente para o desenvolvimento, evoluindo o protótipo para atender aos requisitos levantados;
- Apesar de a maioria dos protótipos ser desenvolvida com o intuito de serem descartados, em alguns casos é possível evoluir o protótipo para um produto final;

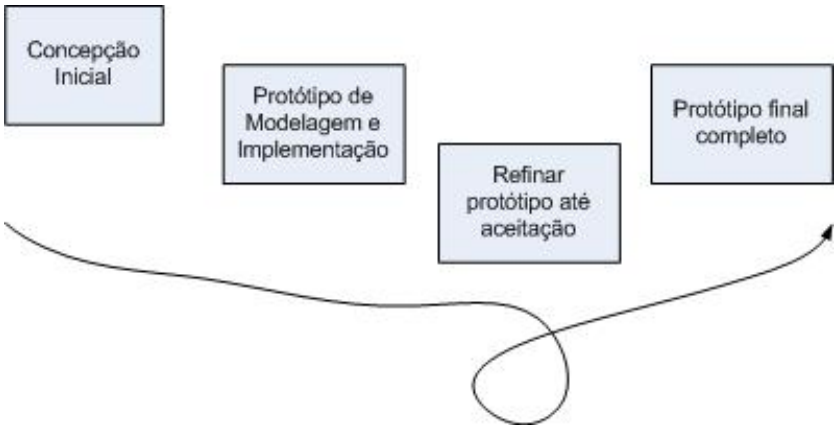

Figura 4 - Processo de desenvolvimento baseado em prototipagem evolutiva

2.5.3 Rational Unified Process

O RUP (Rational, 2000) é um processo de que fornece uma metodologia disciplinada de trabalho, com base em um conjunto de ferramentas proprietárias da IBM Rational e diversos artefatos. É um modelo voltado para empresas com grande volume de comunicação externa e na própria equipe de desenvolvimento, com abundância de artefatos de documentação, em grande parte baseado na UML (*Unified Modeling Language*).

É baseado em elementos que descrevem o que será produzido, o conhecimento necessário e a explicação passo a passo descrevendo especificamente como os objetivos do desenvolvimento serão atingidos. Os principais elementos são:

- Papéis (quem) – Um papel define um conjunto de habilidades, competências e responsabilidades;
- Produtos de trabalho (o quê) – Um produto de trabalho representa o resultado de uma tarefa, incluindo todos os documentos e modelos produzidos durante o processo;
- Tarefas (como) – Uma tarefa descreve uma unidade de trabalho atribuída a um papel que venha a gerar um resultado mensurável;

  Em cada interação as tarefas são divididas em nove disciplinas (Figura 5) (Rational, 2000):

  1. Seis disciplinas de desenvolvimento:
     a. Modelagem de processos
     b. Requisitos
     c. Análise e design
     d. Implementação
     e. Teste
     f. Implantação
  2. Três disciplinas de suporte
     a. Configuração e gerência de mudança
     b. Gerência de projeto
     c. Ambiente

Figura 5 - Disciplinas do RUP de acordo com o fluxo do processo

Cada ciclo de desenvolvimento no RUP é dividido em quatro fases (Rational, 2000), como pode ser visto na Figura 6:

- Concepção – durante a fase de concepção se estabelece o caso do negócio e o escopo do projeto;
- Elaboração – o propósito da fase de elaboração é analisar o domínio do problema, estabelecer um plano de desenvolvimento em nível de arquitetura, um plano de projeto e eliminar os elementos de maior risco;
- Construção – durante a fase de construção todos os componentes e funcionalidades são desenvolvidos e integrados no produto e testados;
- Transição – o propósito da fase de transição é transferir o software para a utilização dos usuários. Tendo o produto sido entregue, problemas e novos pedidos de desenvolvimento normalmente ocorrerão. Esta fase se inicia apenas quando o projeto está maduro o suficiente para utilização do usuário final;

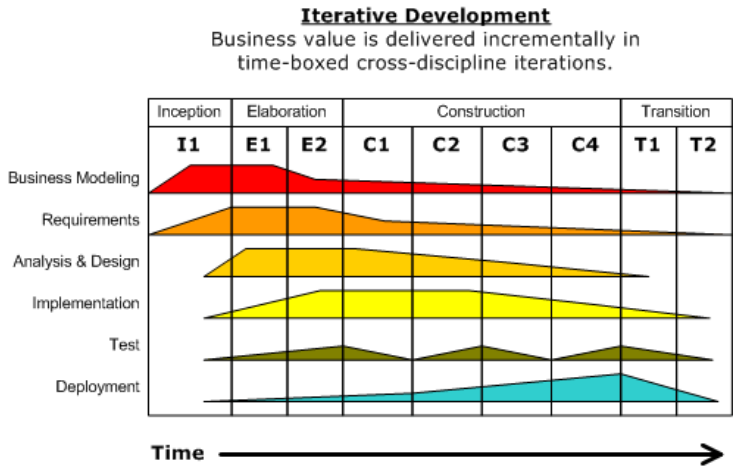

Figura 6 - Processo RUP em um gráfico bidirecional das disciplinas em relação ao ciclo de vida

3. Hive

O *Hive* é um processo de software colaborativo, que disponibiliza um fluxo de atividades permitindo a colaboração da equipe em soluções, procedimentos e documentações.

3.1 Preparação

Para começar a utilizar o *Hive*, e realmente aproveitar as vantagens de um processo colaborativo, é necessário que os profissionais estejam dispostos a colaborar. A colaboração depende dos envolvidos; não é um processo automatizado. Portanto, uma equipe que não compartilha conhecimento e opinião acabará por tornar o processo estagnado e sem grande valia.

Contribuir para conhecimento aberto é uma barreira para a grande maioria dos profissionais. Há muita competição nos ambientes empresariais, o que acaba por tornar o profissional inseguro em compartilhar conhecimento e contribuir para a equipe. Compartilhar conhecimento faz não só o profissional crescer, mas como todos os envolvidos com as atividades. É muito comum nas equipes de desenvolvimento haverem profissionais responsáveis por determinados projetos mesmo anos após sua implantação. Como o conhecimento está segregado nos projetos, é mais cômodo deixar que ele cuide de eventuais problemas relacionados àquele projeto. Ao longo do desenvolvimento do software, diversos artefatos são gerados justamente com o intuito de compartilhar conhecimento, porém, grande parte deles se desatualiza e o restante da equipe não têm contato com tal documentação.

Envolver as pessoas que efetivamente executam as tarefas nas decisões de como estas tarefas devem ser feitas auxilia no desenvolvimento das habilidades dos participantes e do produto final gerado. Mas, para que tudo isso ocorra da maneira que se espera, todos têm que estar, de fato, envolvidos no processo de concepção e em um clima de cooperação.

3.2 Motivação

Os profissionais de nível estratégico (por exemplo, diretores, coordenadores, presidente etc.) de uma organização esperam que um processo de desenvolvimento adequado resulte em lucro, diminuição de riscos e um melhor planejamento estratégico, entre outros aspectos.

Os profissionais de desenvolvimento esperam que suas qualidades sejam mais bem aproveitadas em um bom ambiente de trabalho para a equipe, tornando-a mais produtiva e, de fato, satisfeita com os resultados. Mas, para que isso de fato ocorra, é ideal que a equipe seja composta por pessoas que se importam com o resultado final do produto e queiram de fato construir software de alta qualidade.

É indicado que se ofereça para as pessoas que executam tarefas a oportunidade de decidir como elas serão executadas e avaliadas, dar a responsabilidade e depositar confiança na equipe sobre os métodos de trabalho e soluções.

3.3 Papéis

Os papéis representam um conjunto de responsabilidades que um membro do time assume. Na prática, isso significa que qualquer membro da equipe pode ser por exemplo um Gerente de Atividades, que pode exercer também o papel de um Engenheiro de Software; a disponibilidade de perfis e profissionais determinará isso para cada ambiente.

De forma abrangente, não há distribuição de papéis melhor ou pior, as necessidades da equipe e do projeto irão determinar e moldar a composição da equipe. Os papéis recomendados para a utilização do *Hive* são os seguintes: Arquiteto de Software, Engenheiro de Software, Moderador e Gerente de Atividades.

3.3.1 Arquiteto de Software

O arquiteto de software deve ser uma pessoa com maturidade, visão e experiência que permita solucionar problemas complexos rapidamente e tomar decisões que determinem o rumo do desenvolvimento do produto.
As responsabilidades do papel de Arquiteto de Software são (OpenUP, 2012):

- Gerenciar os requisitos não-funcionais;
- Definir a arquitetura do software, ou seja, definir como o problema será resolvido e introduzir diretrizes, princípios e liderança no aspecto técnico do projeto;
- Compartilhar e discutir a solução com toda a equipe;
- Escolher as tecnologias a serem utilizadas;
- Garantir a qualidade do software em relação à arquitetura definida. É importante citar que todos os engenheiros de software são também responsáveis pela qualidade do seu código;
- Acompanhar e apoiar a equipe no decorrer do desenvolvimento;

Diversos arquitetos são experientes codificadores. Logo, é comum que muitos acabem codificando tanto quanto um engenheiro de software, isso de certa forma ajuda na evolução da equipe. Em um cenário considerado ideal, toda a sua equipe deve ser capaz de discutir soluções em nível arquitetural contribuindo para a solução.

Para pequenos projetos não é necessária a presença de um profissional com o papel exclusivo de Arquiteto. Porém, é aconselhável que haja ao menos a presença de um Engenheiro de Software com mais experiência para guiar o desenvolvimento em nível de arquitetura. Caso o projeto utilize um *framework* para desenvolvimento de software com a sua estrutura original, a presença de um Arquiteto de software se torna menos relevante.

3.3.2 Engenheiro de software

O Engenheiro de software é responsável por desenvolver o sistema de acordo com a arquitetura definida, possivelmente desenvolver protótipos de interface e implementar testes automatizados para todos os níveis da solução com exceção da infra-estrutura (OpenUP, 2012).

Este papel tem uma importância muito grande em todo o processo: é essencial que os profissionais com esse papel possuam um perfil colaborativo. Os melhores Engenheiros de software trabalhando em ambientes competitivos não produzem mais do que os que trabalham cooperativamente em um ambiente colaborativo (Extreme Programming, 2009).

3.3.3 Moderador

O Moderador é o membro da equipe responsável por resolver os conflitos surgidos dos debates em relação ao processo de desenvolvimento. Além disso, modera as contribuições feitas na ferramenta colaborativa e as discussões de evolução de solução compartilhadas.

É essencial que o Moderador possua uma excelente habilidade de comunicação, devido à necessidade constante de debate e discussão com os membros da equipe.

Apesar das atribuições do Moderador, as soluções de desenvolvimento de software precisam necessariamente passar pelo Arquiteto de Software da equipe.

3.3.4 Gerente de atividades

O Gerente de atividades lidera o planejamento do projeto, coordenando a interação entre os clientes e a equipe. Define o *Sprint Backlog,* as prioridades das atividades e acompanha o seu desenvolvimento, certificando-se que os objetivos dos projetos e seus prazos estejam sendo atingidos.

O acompanhamento do Sprint pode ser feito em nível de atividade ou simplesmente em nível de *Release,* o que dará para a equipe uma maior flexibilidade para atender novos requisitos (Extreme Programming, 2009). Para projetos mais complexos ou que possuam contratos críticos que precisem ser acordados anteriormente, o acompanhamento por atividades é recomendado.

Em projetos de grande porte não é aconselhável que o Gerente de atividades acumule outro papel na equipe. Porém, caso o projeto seja com uma equipe menor e o

acúmulo de papéis seja necessário, é preferencial que o Gerente de atividades seja o Arquiteto de software ou o Moderador.

3.4 Ferramentas alternativas

Durante a descrição do processo a ferramenta utilizada como referência é uma Wiki. Nesta seção veremos algumas ferramentas alternativas com as suas vantagens e desvantagens para utilização em um processo de software colaborativo.

3.4.1 Eclipse Process Framework (EPF)

O EPF pode ser encontrado em http://www.eclipse.org/epf/ e é um pacote de plugins para modelagem de processos instalados na plataforma Eclipse.

Vantagens:

- Possui uma instalação *standalone*;
- Disponibiliza uma estrutura de processo inicial que auxilia quem não teve muito contato com modelagem de processos;

Desvantagens:

- Utiliza uma sintaxe própria;
- A colaboração é feita apenas via interações com o controle de versão;

3.4.2 Google Drive

O Google Drive é um serviço de compartilhamento de arquivos disponibilizado pela Google, que pode ser acessado através de https://drive.google.com/.

Vantagens:

- Compartilha diferentes formatos de arquivos;
- Pode ser acessado diretamente do site da Google, não sendo necessária instalação de nenhum software;

- Um documento pode ser editado por mais de um usuário ao mesmo tempo, podendo-se ver as alterações sendo realizadas em tempo real;

Desvantagens:

- Os conteúdos ficam espalhados pelos arquivos compartilhados;
- Não há como compartilhar um arquivo para uma organização, a menos que esse documento seja público;

3.5 O processo

Nesta seção iremos acompanhar todas as atividades do processo com sua descrição, atores envolvidos e sugestões de boas práticas. Abaixo pode-se visualizar na Figura 7 o fluxo do processo de colaboração *Hive* com uma notação similar ao *BPMN – Business Process Modeling and Notation* (OMG.org, 2011).

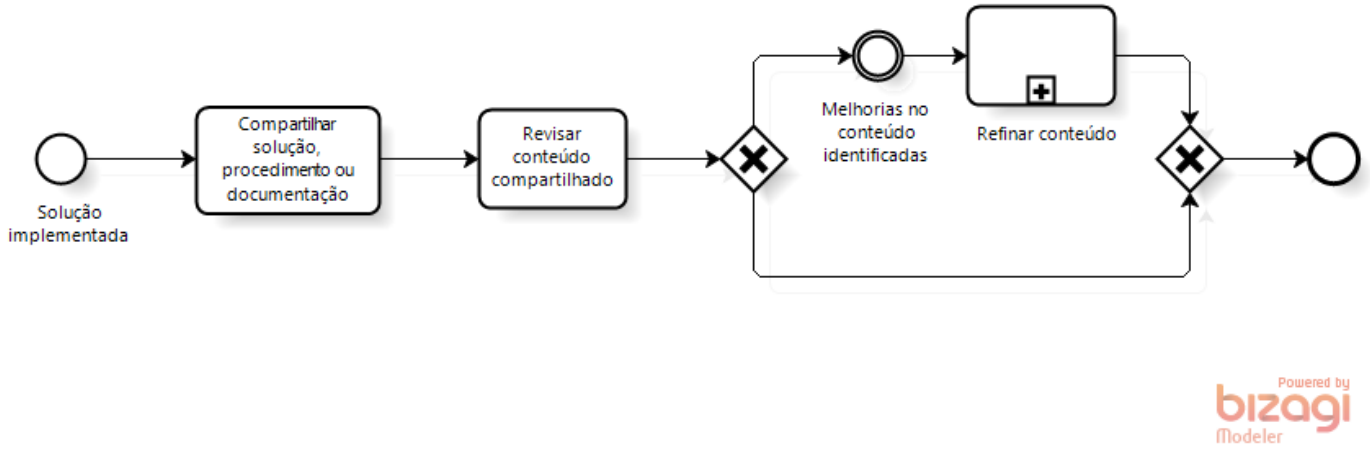

Figura 7 - Hive em notação similar ao BPMN

3.5.1 Compartilhar solução, procedimento ou documentação

- **Entrada:** Solução implementada
- **Saída:** Solução, procedimento ou documentação compartilhada
- **Atores:** Engenheiro de Software

Os Engenheiros de Software que desenvolveram a solução compartilham os métodos utilizados para desenvolvê-la na ferramenta colaborativa.

Uma solução pode ser compartilhada em diferentes níveis de granularidade. É aconselhável que pequenos procedimentos de configuração ou dicas de desenvolvimento também sejam compartilhados. Caso a equipe queira segmentar as soluções específicas de desenvolvimento em outra ferramenta, há outras opções, como,

por exemplo, ferramentas de perguntas e resposta ou mesmo de compartilhamento de trechos de código fonte de forma descritiva.

Se a equipe envolvida quiser adotar um perfil mais ágil de desenvolvimento, a documentação gerada será apenas um insumo para se iniciar o desenvolvimento, logo não será necessária a sua manutenção ao longo do tempo, tornando o seu compartilhamento e discussão em relação a ela opcional. Porém, se o projeto possuir contratos mais formais, onde seja obrigatória a presença de uma documentação atualizada por exigência de clientes, é recomendado que os artefatos sejam redigidos em uma ferramenta colaborativa.

Os requisitos do projeto definidos pelo cliente, opcionalmente podem ser também compartilhados para facilitar a colaboração. Uma boa prática em relação à manutenção dos requisitos é aproveitar-se das vantagens das ferramentas colaborativas mais atuais e disponibilizar para o cliente os requisitos definidos e o seu respectivo estado de desenvolvimento via internet.

3.5.2 Revisar conteúdo compartilhado

- **Entrada:** Conteúdo compartilhado
- **Saída:** Conteúdo compartilhado revisado
- **Atores:** Equipe *Hive*

A Equipe *Hive* revisa o conteúdo compartilhado na ferramenta colaborativa, apontando e debatendo melhorias através de reuniões de equipe ou acrescentando sugestões no próprio texto disponibilizado. Após as considerações, caso necessário, as alterações no processo ou procedimento são revisadas e consolidadas pelo Moderador, resultando em uma nova versão do conteúdo.

3.5.3 Refinar conteúdo

- **Entrada:** Melhorias no conteúdo identificadas
- **Saída:** Melhorias refinadas

- **Atores:** Equipe *Hive*

A Equipe *Hive* revê o conteúdo compartilhado e debate melhorias. Após as considerações, caso necessário, as alterações são revisadas e consolidadas na ferramenta colaborativa pelo Moderador, resultando em uma nova versão do conteúdo. A Figura 8 apresenta esta atividade detalhada em uma notação similar ao *BPMN – Business Process Modeling and Notation* (OMG.org, 2011).

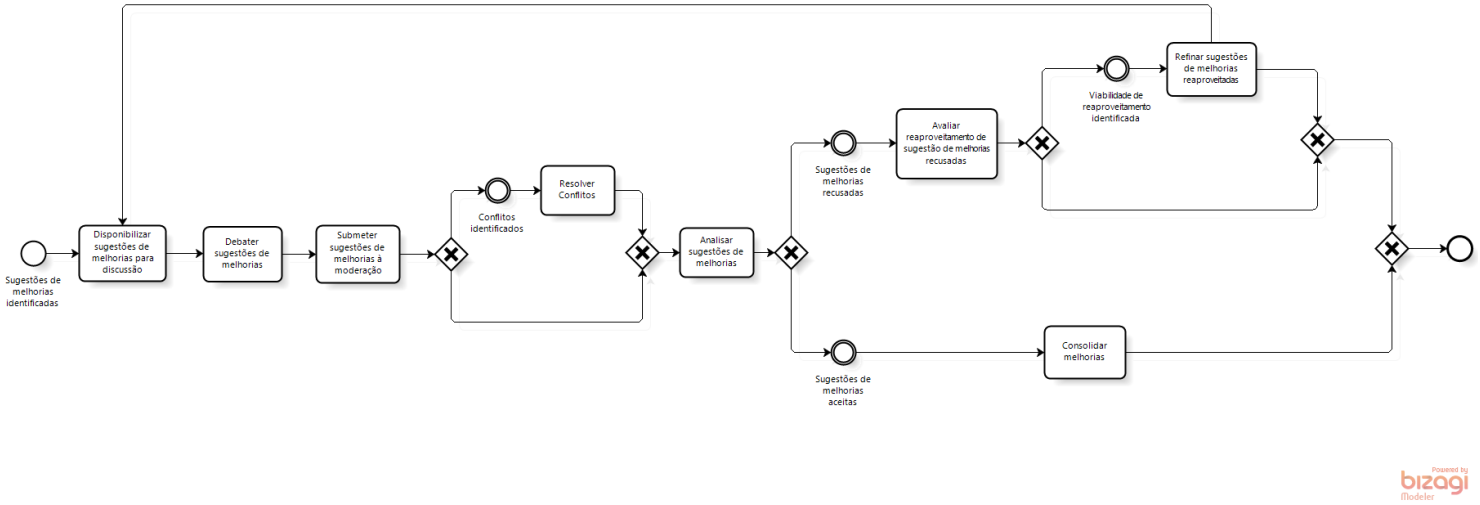

Figura 8 - Subprocesso "Refinar conteúdo"

3.5.3.1 Disponibilizar sugestões de melhorias para discussão

- **Entrada:** Sugestões de melhoria
- **Saída:** Sugestões de melhoria disponibilizadas
- **Atores:** Equipe *Hive*

A Equipe *Hive* disponibiliza as sugestões de melhoria em uma ferramenta colaborativa para visualização e debate.

3.5.3.2 Debater sugestões de melhorias

- **Entrada:** Sugestões de melhoria disponibilizadas
- **Saída:** Sugestões de melhoria debatidas e disponibilizadas para moderação
- **Atores:** Equipe *Hive*, Arquiteto e Moderador.

As sugestões de melhoria, sua organização e sua própria descrição textual são discutidas. Ao final, o processo atualizado com as sugestões de melhoria debatidas é submetido ao Moderador para validação.

3.5.3.3 Submeter sugestões de melhorias à moderação

- **Entrada:** Processo ou procedimento atualizado com as novas sugestões de melhoria
- **Saída:** Processo ou procedimento atualizado com as novas sugestões de melhoria submetido à moderação
- **Atores:** Equipe *Hive*, Arquiteto e Moderador.

O processo ou procedimento atualizado com as sugestões de melhoria debatidas é submetido ao moderador para avaliação em relação ao seu conteúdo e relevância.

3.5.3.4 Resolver conflitos

- **Entrada:** Conflitos de opinião
- **Saída:** Conflitos de opinião resolvidos
- **Atores:** Equipe *Hive* e Moderador.

O Moderador se reúne com a Equipe *Hive* para resolver as divergências surgidas através das sugestões de melhorias debatidas. Nesta reunião, é ideal que cada participante possua um tempo individual sem interrupções para expor seu ponto de vista antes que os debates se iniciem.

3.5.3.5 Analisar sugestões de melhorias

- **Entrada:** Sugestões de Melhorias
- **Saída:** Sugestões de melhorias aceitas ou recusadas
- **Atores:** Moderador.

O Moderador analisa as sugestões de melhorias quanto à sua coerência em relação ao processo atual e sua construção textual.

3.5.3.6 Avaliar reaproveitamento de sugestão de melhorias recusadas

- **Entrada:** Sugestões de melhorias recusadas
- **Saída:** Viabilidade de reaproveitamento avaliada
- **Atores:** Moderador, Equipe *Hive*

O Moderador avalia a sugestão de melhoria que foi recusada para verificar se algum conteúdo pode ser reaproveitado e agregado ao processo ou procedimento.

3.5.3.7 Refinar sugestões de melhorias reaproveitadas

- **Entrada:** Processo ou procedimento alterado com as novas sugestões de melhoria
- **Saída:** Processo ou procedimento atualizado e consolidado com as sugestões de melhorias validadas
- **Atores:** Moderador

O Moderador revisa textual e conceitualmente de acordo com o processo as sugestões de melhoria reaproveitadas.

3.5.3.8 Consolidar melhorias

- **Entrada:** Processo ou procedimento alterado com as novas sugestões de melhoria
- **Saída:** Processo ou procedimento atualizado e consolidado com as sugestões de melhorias validadas
- **Atores:** Moderador

O Moderador consolida as melhorias, gerando uma nova versão do processo, procedimento ou documentação e disponibiliza aos interessados através da ferramenta colaborativa.

4. Exemplo de uso

Nesta seção é apresentada uma implementação do *Hive* aplicada em um caso ligado ao cotidiano de um projeto de desenvolvimento de software. Durante as descrições das interações serão mostradas exemplificações gráficas das aplicações da ferramenta colaborativa no contexto abordado e dicas de utilização e boas práticas.

4.1 Cenário

Para este exemplo utilizou-se o cenário de uma empresa *Startup* nacional chamada 38Sinais, que tem como especialidade o desenvolvimento de aplicações web para público final. A empresa está com o seu primeiro projeto, que seria um software disponibilizado como serviço via web para gerência de locação de carros cujos requisitos já foram previamente levantados. Os investidores exigiram que a empresa utiliza-se algum processo de desenvolvimento e os fundadores optaram por utilizar uma implementação do *Hive* baseada no *SCRUM*.

Para o desenvolvimento a equipe utiliza o framework para web *Ruby on Rails* com algumas bibliotecas que facilitam o desenvolvimento orientado a comportamento.

4.2 Equipe

A equipe fictícia é composta por quatro profissionais e possui a seguinte distribuição de papéis:

- Miguel - Gerente de atividades, Moderador
- Alice - Arquiteta de software, Engenheira de Software
- Fernando - Engenheiro de Software
- Rita - Engenheira de Software

4.3 Ferramenta

Como ferramenta colaborativa a 38Sinais optou por utilizar o GitHub (www.github.com) pelo fato de já possuir o controle de versão, wiki e controle de atividades no mesmo serviço. Há diversas outras ferramentas colaborativas, entre elas é essencial destacar a MediaWiki (http://www.mediawiki.org), utilizada na Wikipedia (http://www.wikipedia.org).

4.4 O Processo

A Figura 9 apresenta um esquema do processo utilizado pela empresa 38Sinais, onde a organização resolveu adotar uma implementação do *Hive* utilizando o SCRUM:

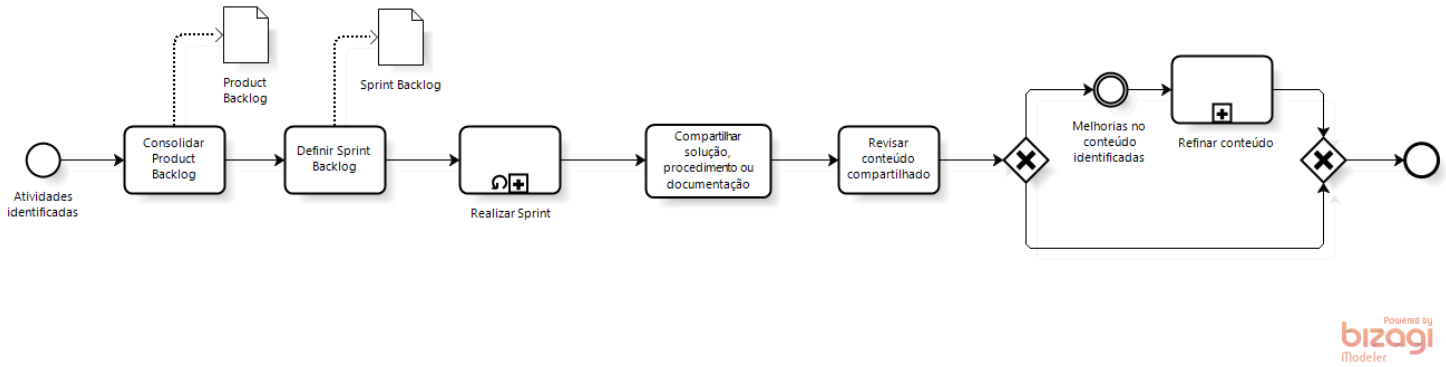

Figura 9 – Implementação do *Hive* na empresa 38Sinais utilizando o SCRUM.

Nesta seção simula-se a equipe da 38Sinais interagindo em cada uma das atividades do processo de desenvolvimento aplicando as práticas e atividades do processo de desenvolvimento de software *Hive*.

4.5 Consolidar Product Backlog

- **Entrada:** Atividades a serem desenvolvidas

- **Saída:** Product Backlog
- **Atores:** Gerente de Atividades

O Gerente de Atividades organiza as atividades identificadas agrupando-as por prioridade. As atividades podem conter outros atributos além da sua descrição, como, por exemplo: estimativa, prazo ou release do projeto. É importante mencionar que os atributos escolhidos dependem diretamente dos objetivos do projeto, da empresa e do tamanho da equipe envolvida, além da própria cultura organizacional.

Miguel, baseado nos requisitos pré-definidos definiu as atividades para o projeto apresentadas na Figura 10:

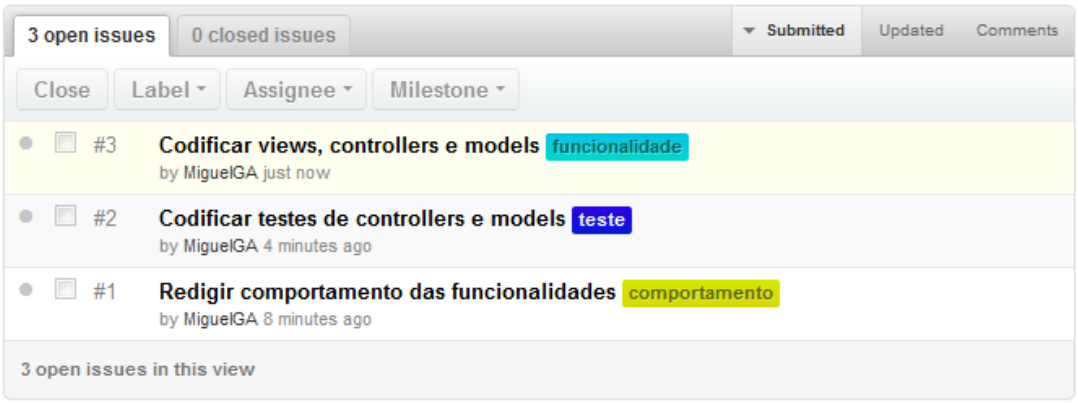

Figura 10 - Exemplo de atividades cadastradas no GitHub

As atividades acima correspondem ao *Product Backlog* do projeto com prioridade definida pelo número à esquerda (Ex: #1).

Miguel atribuiu as tarefas para os seguintes componentes da equipe:

- #1 - Redigir comportamento das funcionalidades - Rita
- #2 - Codificar testes de controllers e models - Alice
- #3 - Codificar views, controllers e models – Fernando

Cada componente a quem foi atribuída uma tarefa analisou a complexidade das atividades e estimou a duração para o desenvolvimento da solução.

É importante que se tenha em mente a diferença entre estimativa e prazo. Muitos profissionais tendem a tratá-los como se tivessem o mesmo significado, mas não têm. Estimativa é um cálculo de tempo aproximado para a conclusão da tarefa mediante uma avaliação baseada em alguns critérios.

Ao avaliar uma tarefa, é preciso que haja algum critério para o cálculo do tempo da estimativa. No *Hive* recomenda-se a utilização do critério de histórico, ou seja, estima-se o tempo de acordo com a estimativa dada em outras atividades de mesma complexidade e/ou execução parecida. Caso não haja histórico para ser consultado, aconselha-se estimar baseando-se na experiência profissional do executor da tarefa.

4.6 Definir Sprint Backlog

- **Entrada:** Atividades a serem desenvolvidas
- **Saída:** Product Backlog
- **Atores:** Gerente de Atividades e Equipe *Hive*

O Gerente de Atividades e a Equipe *Hive* definem quais atividades do *Product Backlog* serão executadas no *Sprint*. Na definição do *Sprint* é importante que a sua duração seja acordada, sendo um intervalo de uma a três semanas.

Para o cenário observado Miguel d efiniu o conjunto de atividades que farão parte do primeiro *Sprint,* como pode ser visto na Figura 11:

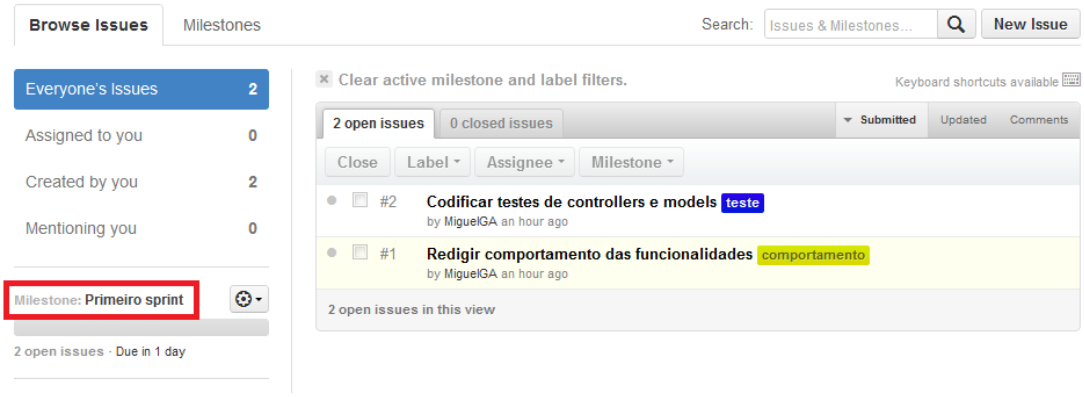

Figura 11 - Exemplo de agrupamento de atividades por Sprint no GitHub

Este *Sprint* ficou definido com duração de duas semanas, baseado nas estimativas dadas pelos executores das atividades do *Product Backlog*.

4.7 Realizar Sprint

- **Entrada:** Sprint Backlog
- **Saída:** Solução implementada
- **Atores:** Equipe *Hive*

A Equipe *Hive* desenvolve as atividades definidas para o Sprint no intervalo determinado, tendo em vista os requisitos a serem atendidos.

No exemplo que está sendo acompanhado, Rita e Fernando e os Engenheiros de Software realizaram as atividades que lhe foram atribuídas no Sprint.

4.8 Compartilhar solução, procedimento ou documentação

Rita compartilha na wiki a documentação da biblioteca em que ela se baseou para desenvolver a solução, como pode ser vista na Figura 12.

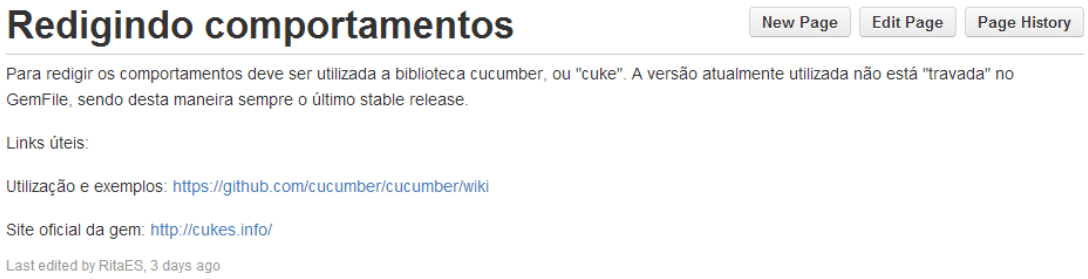

Figura 12 - Exemplo de documentação compartilhada em uma ferramenta colaborativa

4.8.1 Revisar conteúdo compartilhado

A equipe revisa a documentação que foi compartilhada por Rita e verifica que o conteúdo está incompleto e pode ser complementado.

4.8.2 Refinar Conteúdo

4.8.2.1 Disponibilizar Sugestões de melhoria para discussão

A equipe disponibilizou as melhorias que poderiam complementar a documentação que Rita compartilhou, como pode ser vista na Figura 13.

Figura 13 - Sugestões de melhorias na Wiki

4.8.2.2 Submeter sugestões de melhorias à moderação

Após terem disponibilizado as sugestões para a documentação que Rita havia compartilhado, estas são submetidas à moderação.

Nas ferramentas de wiki mais atuais o moderador recebe notificações automáticas quando há alterações de conteúdo, caso seja necessário.

4.8.2.3 Resolver Conflitos

Miguel verifica que as sugestões dadas são complementares, não havendo dessa maneira conflitos a serem resolvidos.

4.8.2.4 Analisar sugestões de melhorias

Miguel analisa as sugestões de melhoria e as aceita, de acordo com o que foi discutido com a equipe.

4.8.2.5 Consolidar Melhorias

Miguel consolida as sugestões de melhoria em uma nova versão da documentação disponibilizada na Wiki, conforme pode ser visto na Figura 14.

Figura 14 - Melhorias consolidadas na Wiki

5. Conclusão

No capítulo 1 foi abordada uma introdução ao *Hive,* sua motivação e preparação para ser utilizado. No capítulo 2 foram vistos conceitos sobre metodologias e processos de software utilizados atualmente, além de outros pontos necessários para uma base de conhecimento. No capítulo 3 foi abordado o processo do *Hive*, suas práticas e papéis envolvidos. No capítulo 4 estudou-se um exemplo de utilização do processo em uma pequena empresa de desenvolvimento de software, onde pudemos verificar que as soluções dadas pelos componentes da equipe se tornaram mais dinâmicas com a utilização de uma ferramenta colaborativa.

Dessa forma, mesmo que sem perceber a equipe realiza revisões dos entregáveis, debate melhorias para implementações e disponibiliza conteúdos de melhor qualidade, que podem ser utilizados futuramente para consulta ou mesmo treinamento de novos profissionais. O processo, os procedimentos e as documentações se tornam evolutivos, tornando a equipe mais participativa não apenas no "fazer" mas também no "como deve feito".

Os ganhos no uso de um processo colaborativo podem ser nitidamente percebidos conforme a prática dos envolvidos na sua utilização. A equipe ganha em conhecimento, se torna mais responsável pelo métodos e pelo resultado final se tornando cada vez mais auto gerenciável.

5.1 Trabalhos futuros

O *Hive* pode ser evoluído com implementações utilizando-se de outros processos de desenvolvimento conhecidos. Uma implentação com uma metodologia mais tradicional seria de grande valia.

Evoluir o processo e adaptá-lo não apenas para processos de desenvolvimento mas também para outros tipos de processos, até mesmo os não ligados à engenharia de software, sendo que esses já se utilizam de métodos, documentação e conhecimento da equipe para seram realizados.

6. Referência Bibliográfica

(28 de 09 de 2009). Fonte: Extreme Programming: http://www.extremeprogramming.org/

(01 de 06 de 2012). Fonte: OpenUP: http://epf.eclipse.org/wikis/openup/

Agile Manifesto. (2001). Fonte: www.agilemanifesto.org

Baldwin, H. (24 de Junho de 2002). *Collaboration greases company's business gears.* Fonte: Tech Republic: www.techrepublic.com

Beck, K. (2003). *Test-Driven Development by Example.*

Centers for Medicare & Medicaid Services (CMS) Office of Information Service. (27 de 03 de 2008). Selecting a development approach.

Curtis, B. (1988). A Field Study of the Software Design. *Computing Pratices* .

Elliott, G. (2004). *Global Business Information Technology: an integrated systems approach.* Pearson Education.

Herrman, C. S. (06 de 03 de 2009). Fundamentals of Methodology, Definitions and First Principles.

Katsicas, S. K. (2009). *Computer and Information Security Handbook.* Morgan Kaufmann Publications Elsevier Inc.

McConnell, S. (1996). *Rapid Development: Taming Wild Software Schedules.* Microsoft Press.

OMG.org. (2011). *BPMN 2.0 Specification.*

Pinnadyne. (09 de 10 de 2009). *Collaboration Made Easy.* Fonte: http://www.pinnadyne.com/index.php?option=com_content&view=article&id=74&Itemid=80

PMBoK 4th Edition. (2009).

Rational. (03 de 07 de 2000). *RUP Best Practices for Software.* Fonte: IBM: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Schwaber, K., & Sutherland, J. (1991-2011). *The Scrum Guide.* Fonte: Scrum: http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf

Sutherland, K. S. (1991 - 2011). *The Scrum Guide.* Fonte: Scrum.org: http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011.pdf

XProgramming. (1997-1999). *Pair Programming*. Fonte: Extreme Programming: http://www.extremeprogramming.org/rules/pair.html