



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

Histimate - Ferramenta de Apoio a Estimativas em Projetos de Desenvolvimento Ágeis

Marcela Soares Farias
Roberta Andreza Almeida dos Santos

Orientador
Gleison dos Santos Souza

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2013

Histimate - Ferramenta de Apoio a Estimativas em Projetos de Desenvolvimento Ágeis

Marcela Soares Farias
Roberta Andreza Almeida dos Santos

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.

Aprovada por:

Prof. Gleison dos Santos Souza, D.Sc. (UNIRIO)

Prof. Alexandre Luis Correa, D.Sc. (UNIRIO)

Prof. Mariano Pimentel, D.Sc. (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

DEZEMBRO DE 2013

Agradecimentos

Marcela Farias: *À minha família, pela compreensão e apoio durante o tempo em que estive dedicada a este trabalho. Ao Luciano Fernandes, pela motivação e apoio durante este período.*

Aos meus amigos e companheiros de trabalho, pela compreensão e cessão de disponibilidade para o desenvolvimento deste trabalho.

À minha dupla neste trabalho e grande amiga Roberta, sem a qual este trabalho não seria possível.

Ao nosso orientador, Prof. Gleison Santos, pelo auxílio, paciência e compreensão ao longo de todo o desenvolvimento deste trabalho.

Agradeço a todos vocês.

Roberta Santos: *À minha família pelo apoio e amor incondicional durante toda a vida. Em especial aos meus pais e ao meu irmão que estiveram comigo em todos os momentos. Obrigada por acreditarem em mim, me apoiarem nas minhas decisões e me ajudarem a não desistir. Vocês são meus exemplos e minhas inspirações.*

À minha parceira neste trabalho e querida amiga Marcela. Sem seu apoio e confiança a realização deste trabalho seria muito mais difícil.

Ao querido Professor Gleison Santos, pela paciência, contribuição e compreensão nos momento mais difíceis. Tenho certeza que acertamos na escolha para um papel tão importante quanto o de orientador.

A todos que, de alguma forma, passaram por minha vida profissional. Os ensinamentos aprendidos com vocês foram de fundamental importância para o

desenvolvimento deste trabalho. Um agradecimento especial por entenderem e apoiarem minha dedicação a este projeto.

Aos amigos que acompanharam toda essa jornada. Aos que compreenderam que, nesse momento, a ausência seria necessária. Pelo incentivo, preocupação e disponibilidade.

Obrigada por estarem sempre presentes na minha vida.

Aos amigos da UNIRIO, que estiveram comigo durante todos esses anos, me apoiando e incentivando. Essa trajetória não seria a mesma sem a amizade e companhia de vocês.

À todos vocês, meu muito obrigada.

RESUMO

O processo de estimativas em projetos que utilizam métodos ágeis não é trivial e por muitas vezes, as equipes se deparam com problemas decorrentes de estimativas imprecisas, pois as histórias e tarefas acabam por ser mais ou menos complexas do que foi estimado durante o planejamento. Este trabalho apresenta a Histimate, uma ferramenta para criação de bases de estimativas, que a partir do uso contínuo, busca apoiar o planejamento dos ciclos iterativos de um projeto e auxiliar o aprendizado da equipe, facilitando a detecção de possíveis inconsistências nas estimativas.

Palavras-chave: Gerência de projetos, métodos ágeis, base de estimativas, estimativas

ABSTRACT

The estimation process for projects that use agile methods is not trivial, and teams are often faced with problems caused by inaccurate estimates because stories and tasks turn out to be more or less complex than what was estimated in planning. The present study presents Histimate, a tool for creating estimates databases. The continued use of this tool supports the planning of the iterative cycles of a project and also contributes to the team's learning by facilitating the detection of possible inconsistencies in the estimates.

Keywords: Project management, agile methodologies, estimation base, estimation

Índice

1	Introdução.....	1
1.1	Contexto	1
1.2	Motivação	1
1.3	Objetivo	2
1.4	Estrutura do texto	3
2	Fundamentação teórica.....	4
2.1	Métodos ágeis.....	4
2.2	Manifesto ágil.....	5
2.3	Scrum.....	6
2.4	Kanban.....	10
2.5	Extreme programming (XP).....	11
2.6	Uso de histórias para definição do escopo.....	15
2.7	Estimativas em projetos de software tradicionais.....	16
2.8	Estimativas em projetos de software ágeis.....	18
2.9	Considerações finais.....	21
3	Especificação e funcionalidades da ferramenta.....	22
3.1	A ferramenta.....	22
3.2	Modelagem	23
3.2.1	Definição da dinâmica de um projeto ágil.....	23
3.2.2	Diagrama de Classes.....	24
3.2.3	Histórias de usuário	27
3.3	Tecnologias utilizadas	28
3.3.1	Linguagem de programação	29
3.3.2	Framework web	31
3.3.3	Framework <i>front-end</i>	32
3.3.4	Controle de versão.....	33
3.3.5	Testes	34
3.3.6	Banco de dados.....	35
3.4	Funcionalidades.....	36
3.4.1	Acesso ao sistema.....	36
3.4.2	Criação do projeto	38
3.4.3	Planejamento e detalhamento do <i>sprint</i>	43
3.4.4	Execução do <i>sprint</i>	47
3.4.5	Conclusão do <i>sprint</i>	49

3.5	Comparação com ferramentas similares.....	51
3.6	Considerações finais.....	51
4	Conclusão.....	53
4.1	Considerações finais.....	53
4.2	Principais contribuições.....	53
4.3	Trabalhos futuros.....	54

Índice de Figuras

Figura 1 - Exemplo de Gráfico Sprint Burndown.	8
Figura 2 - Exemplo de Quadro Kanban.	11
Figura 3 - Exemplos de histórias da ferramenta Histimate.	15
Figura 4 - Cone da Incerteza.	17
Figura 5 - Diagrama de classes da ferramenta.	25
Figura 6 - Representação esquemática da arquitetura MVC.	32
Figura 7 - Tela de login.	37
Figura 8 - Tela de cadastro.	37
Figura 9 - Tela de edição de cadastro.	37
Figura 10 - Tela de <i>Dashboard</i>	38
Figura 11 - Tela de criação de projeto.	38
Figura 12 - Janela modal de cadastro de equipe.	39
Figura 13 - Janela modal de criação de status.	39
Figura 14 - Janela modal de criação de estimativas de histórias.	40
Figura 15 - Janela modal de criação de estimativa de tarefa.	40
Figura 16 - Tela de cadastro do projeto com informações preenchidas.	41
Figura 17 - Tela de <i>backlog</i> do projeto.	41
Figura 18 - Continuação da tela de <i>backlog</i> do projeto.	42
Figura 19 - Janela modal de criação de história.	42
Figura 20 - Tela de visualização de <i>sprints</i>	43
Figura 21 - Tela de criação do <i>sprint</i>	43
Figura 22 - Tela de <i>sprint</i> com status em planejamento.	44
Figura 23 - Janela modal de estimativa de história.	44
Figura 24 - Tela de <i>sprint</i> com status em detalhamento.	45
Figura 25 - Janela modal de criação de tarefas.	45
Figura 26 - Mensagem informativa de ausência de base histórica de histórias com mesmo tamanho.	46
Figura 27 - Mensagem com resultado da análise das estimativas das tarefas de uma história. ...	46
Figura 28 - Mensagem de resultado da análise de verificação do tamanho do <i>sprint</i>	47
Figura 29 - Tela de <i>sprint</i> com status em execução.	48
Figura 30 - Janela modal de edição de tarefa.	48
Figura 31 - Apresentação de tarefas no último status do <i>board</i>	49
Figura 32 - Tela de edição de tarefa concluída.	49
Figura 33 - Janela modal de confirmação da finalização do <i>sprint</i>	50
Figura 34 - Tela de visualização de <i>sprint</i> com status concluído.	50

Índice de Tabelas

Tabela 1 - Tabela comparativa de ferramentas similares	51
--	----

1 Introdução

1.1 Contexto

As metodologias ágeis surgiram da necessidade de atender às crescentes pressões das organizações por inovação, produtividade (prazos cada vez mais curtos), flexibilidade e melhoria no desempenho/qualidade dos projetos de desenvolvimento de Software (Steffen, 2012). Em geral, atendem aos princípios descritos no Manifesto Ágil (Beck *et al*, 2001).

Pode-se observar uma crescente adoção de metodologias ágeis para gestão e planejamento de projetos de software. Muitas empresas buscam adotar algumas dessas metodologias impulsionadas pelos casos de sucesso relatados, por exemplo, Locaweb, ITM, Ikwa (AgilCoop), Globo.com (Bardusco, 2008) e Spotify (Rebelo, 2013). Entre as metodologias ágeis mais populares para gerenciamento de projetos atualmente, pode-se destacar o Scrum, o Kanban e Extreme Programming (XP) (Steffen, 2012).

1.2 Motivação

Apesar de ser cada vez mais comum, o uso destas metodologias enfrenta dificuldades, pois suas adoções implicam em mudanças nos hábitos dos envolvidos. As práticas previstas por vezes podem ir de encontro a algumas culturas comportamentais da organização. Assim, é preciso que haja uma disposição a fazer determinadas adaptações para adotar tais métodos (Hastie, 2009). Da mesma forma, espera-se comprometimento dos funcionários em adaptar-se às mudanças propostas e a buscar a eficácia que é proposta pelos princípios ágeis (Beck *et al*, 2001).

Outro problema identificado com frequência em equipes que utilizam estas metodologias é a dificuldade em estimar as histórias e tarefas a serem executadas durante um projeto de software (Shore e Warden, 2007). Muitas vezes, são definidas

estimativas que não se confirmam, pois as tarefas e histórias são mais ou menos complexas do que estimado no início do projeto. Embora o Manifesto Ágil (Beck *et al.*, 2001) afirme que a equipe aprende com o tempo, não é trivial estabelecer bases para a coleta de informação sobre esforço das tarefas e das estimativas a serem usadas nos projetos.

As organizações precisam definir uma base histórica de estimativas de modo a apoiar os planejamentos dos projetos. Os dados armazenados nesta base devem auxiliar a equipe do projeto nas estimativas de histórias e tarefas semelhantes a outras já estimadas anteriormente. Dessa forma, o registro histórico das estimativas também acaba por auxiliar o aprendizado da equipe, visto que torna possível a identificação de possíveis inconsistências nas estimativas.

1.3 Objetivo

O objetivo deste trabalho é apoiar a definição de estimativas de histórias em um projeto de software que use ciclos iterativos frequentes, comuns quando se utilizam metodologias ágeis. Para tanto, será criada uma ferramenta específica que possibilite a descrição do escopo do projeto, além de apoiar a estimativa das tarefas para composição do escopo das iterações. A partir do uso contínuo da ferramenta, será possível a criação de uma base de estimativas para apoiar futuros planejamentos e também para avaliar se a estimativa foi adequada durante a execução do projeto.

Para apoiar a criação da base de histórica de estimativas a serem utilizadas por organizações que desenvolvam projetos ágeis, a ferramenta possibilitará:

- A seleção de histórias que compõem cada ciclo iterativo;
- A realização das estimativas das histórias e, conseqüentemente, do ciclo de desenvolvimento;
- O registro do esforço estimado para as tarefas;
- O registro do esforço real das tarefas;
- A indicação se, historicamente, o desempenho da equipe naquele projeto é compatível com a estimativa feita para um ciclo específico.

1.4 Estrutura do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo I: Apresenta a introdução do trabalho, indicando seu contexto, a motivação para implementá-lo e seu objetivo;
- Capítulo II: Apresenta uma revisão da literatura sobre a origem dos métodos ágeis, além de dar ênfase nas metodologias ágeis de desenvolvimento de software mais utilizadas: Scrum, Kanban e XP. Logo depois, apresenta-se o uso de histórias na definição de escopo de um ciclo iterativo e são apresentadas as características das estimativas em projetos tradicionais e ágeis;
- Capítulo III: Apresenta a ferramenta criada no escopo desta monografia, sua modelagem e especificação, as tecnologias utilizadas na implementação e as funcionalidades, além de fazer uma comparação com ferramentas similares.
- Capítulo IV: Apresenta as principais contribuições, sugere possibilidades de aprofundamento posterior e reúne as considerações finais.

2 Fundamentação teórica

As organizações demonstram um interesse constante na melhoria de seus processos e metodologias de desenvolvimento. Buscando tal melhoria, o uso de metodologias ágeis vem sendo amplamente difundido para a gerência e planejamento dos projetos de software.

Neste capítulo serão apresentados uma visão geral de métodos ágeis, o Manifesto Ágil e algumas das metodologias mais utilizadas atualmente, além do uso de histórias para definição do escopo e características das estimativas em projetos tradicionais e ágeis.

2.1 Métodos ágeis

Segundo Sommerville (2011), o fracasso de muitos grandes projetos de software nas décadas de 60 e 70 foi o primeiro indicativo das dificuldades na gerência de projetos. O produto normalmente era entregue com atraso, não era confiável, custava várias vezes mais do que previam as estimativas originais e, muitas vezes, exibia características precárias de desempenho. Ainda segundo Sommerville (2011), esses projetos, diferentemente do que se possa pensar, não fracassaram porque os gerentes ou os programadores eram incompetentes. Pelo contrário, esses projetos grandes e desafiadores atraíam pessoas que eram consideradas de capacitação acima da média. Na verdade, a falha residia na abordagem de gerenciamento utilizada, visto que técnicas de gerenciamento provenientes de outras disciplinas da engenharia eram aplicadas e mostravam-se ineficazes para a atividade de desenvolvimento de software.

A partir da década de 1990 foram desenvolvidas noções de abordagens ágeis como parte de uma reação contra os métodos tradicionais, que eram vistos como burocráticos e lentos (Sommerville, 2011). Tais abordagens foram fortemente influenciadas pelas melhores práticas da indústria japonesa, particularmente pelos

princípios implementados pelas companhias Honda e Toyota. O Sistema Toyota de Produção ficou conhecido como Produção Enxuta ou *Lean Manufacturing* e surgiu com o propósito de aumentar a eficiência da produção eliminando desperdícios. Ao longo do tempo, um grande número de ferramentas e técnicas foram desenvolvidas para permitir às organizações a aplicação dos conceitos e práticas do Lean (Santos, 2011).

2.2 Manifesto ágil

Em 2001, um grupo de 17 profissionais veteranos na área de software decidiu se reunir nos EUA para discutir formas de melhorar o desempenho de seus projetos. O grupo chegou ao consenso de que alguns princípios eram determinantes para a obtenção de bons resultados. Os resultados deste encontro foram a identificação de 12 princípios e a publicação do Manifesto Ágil (Beck et al, 2001) que apresenta quatro premissas:

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

Fadel e Silveira (2010) apontam os 12 princípios do Manifesto:

1. A prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
2. As mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
3. Frequentes entregas do software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
4. As pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
5. Os projetos devem ser construídos em torno de indivíduos motivados. Dando o ambiente e o apoio necessário e confiança para fazer o trabalho;

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

2.3 Scrum

O Scrum foi concebido como uma metodologia de gerenciamento de projetos e é baseado nas informações apresentadas por Takeuchi e Nonaka (1986), que observaram que projetos que utilizavam equipes pequenas e multidisciplinares produziam melhores resultados e associaram estas equipes à formação do Scrum, uma jogada do Rugby.

O Scrum aplicado ao desenvolvimento de software foi introduzido em 1993, por Jeff Sutherland e, em 1996, Ken Schwaber formalizou a definição de Scrum e ajudou a implantá-lo no gerenciamento de projetos de software ao redor do mundo. Desde então, o Scrum se tornou uma das principais alternativas às metodologias tradicionais e tem sido adotado por gestores que buscam garantir a obtenção do melhor produto possível e por engenheiros que buscam garantir que farão seu trabalho da melhor forma possível, sendo usado em milhares de projetos por todo o mundo (Schwaber e Beedle, 2002).

Segundo Ken Schwaber e Jeff Sutherland (2013), o Scrum é uma estrutura processual que visa apoiar o desenvolvimento e manutenção de produtos complexos, empregando uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

De acordo com o Guia do Scrum (Schwaber e Sutherland, 2013), o coração desta metodologia é o *sprint*, um *time-box* de um mês ou menos, durante o qual um “Pronto”, versão incremental potencialmente utilizável do produto, é criado. *Sprints* têm durações coerentes em todo o esforço de desenvolvimento. Um novo *sprint* inicia imediatamente após a conclusão do *sprint* anterior.

Quando o item do *backlog* do produto ou um incremento é descrito como “Pronto”, os integrantes devem ter um entendimento compartilhado do que significa o trabalho estar completo, assegurando a transparência. Esta é a “Definição de Pronto” para o time Scrum e é esperado que esta definição seja expandida de acordo com a maturidade do time para incluir critérios mais rigorosos de qualidade (Schwaber e Sutherland, 2013). Como a percepção de “Pronto” depende do entendimento dos integrantes do time, esta definição pode variar entre os times Scrum.

Esta definição orienta a Equipe de Desenvolvimento na seleção de itens do *backlog* do produto durante a reunião de planejamento do *sprint*. O propósito de cada *sprint* é entregar incrementos de funcionalidades potencialmente utilizáveis. A Equipe de Desenvolvimento entrega um incremento de funcionalidade do produto a cada *sprint*. Este incremento é utilizável, assim o *Product Owner* pode escolher por liberá-lo imediatamente. Cada incremento é adicionado a todos os incrementos anteriores e completamente testado, garantindo que todos os incrementos funcionam juntos (Schwaber e Sutherland, 2013).

Os artefatos do Scrum representam o trabalho ou o valor dos vários modos que são úteis no fornecimento de transparência e oportunidades para inspeção e adaptação. São eles (Schwaber e Sutherland, 2013):

- *Backlog* do Produto - lista priorizada de tudo que pode ser necessário no produto. Conforme o projeto e o produto a ser desenvolvido evoluem, mais itens podem ser adicionados ou removidos. Um item do *backlog* do produto é chamado de história. O *Product Owner* é responsável pelo *backlog* do produto, incluindo seu conteúdo, disponibilidade e ordenação.
- *Backlog* do *sprint* - é um conjunto de itens do *backlog* do produto selecionados para o *sprint*. O time é responsável por criar e estimar as tarefas do *backlog* do *sprint* na reunião de planejamento do *sprint*, e, se necessário, criar ou remover tarefas durante a execução do *sprint*.

- Gráfico *Sprint Burndown* (Figura 1) - mede os itens do *backlog* do *sprint* restantes ao longo do tempo de um *sprint*. Possui no eixo X o número de dias do *sprint* e no eixo Y a estimativa de horas do time para completar todas as histórias selecionadas para o *sprint*. Uma linha é traçada do maior valor de X até o maior valor de Y e serve como guia da produtividade do time.

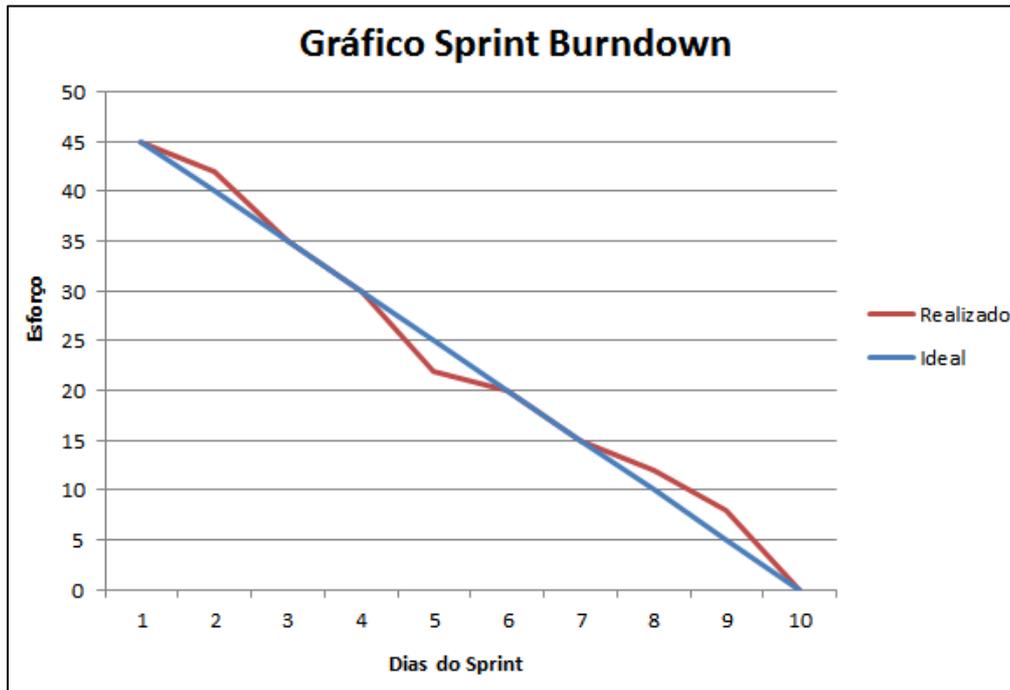


Figura 1 - Exemplo de Gráfico Sprint Burndown.

Fonte: Elaborado pelas autoras.

As histórias são descrições de interações desejadas entre usuário e sistema. As descrições devem ser objetivas de modo a facilitar seu entendimento e a definição dos critérios de aceitação. Por outro lado, a história deve apresentar informações suficientes para que a equipe possa estimar com maior precisão o esforço necessário para implementá-la. Cada história pode ser dividida em tarefas, que são as atividades necessárias para que a história seja implementada.

Os papéis dentro do contexto de Scrum são os seguintes (Schwaber e Sutherland, 2013):

- *Scrum Master* - é o responsável por garantir que o Scrum seja entendido e seguido na organização, remover qualquer impedimento (elementos que estejam impedindo a equipe (ou time) de terminar uma tarefa) e auxiliar o time a ser auto-organizável. O *Scrum Master* não faz parte do time e não o gerencia;

- *Product Owner* (PO) - é o responsável por maximizar o valor do trabalho do time priorizando os itens do *backlog* do produto (lista de funcionalidades que serão desenvolvidas no projeto) e garantindo a sua visibilidade e clareza. É importante que este papel seja realizado por apenas uma pessoa por projeto de forma que não haja divergências e conflitos de opiniões. O PO tem a palavra final sobre o produto, podendo ser aconselhado por outras pessoas.
- Time Scrum - são os que de fato realizam o trabalho, transformando os itens do *backlog* do produto em incrementos de produto potencialmente entregáveis a cada *sprint*. Deve ser interdisciplinar, sem títulos (papéis) e auto-organizável. O tamanho do time deve ser pequeno (de 5 a 9 pessoas), de maneira que otimize a comunicação e produtividade.

O Scrum aplica os eventos com duração fixa (*time-boxes*) para criar regularidade e facilitar o controle, dentre esses eventos temos (Schwaber e Sutherland, 2013):

- Reunião de planejamento do *sprint* - é quando a iteração é planejada. A duração desta reunião geralmente é de 8 horas para um *sprint* de um mês de duração e acontece em duas partes. Por isso, geralmente a reunião de planejamento é realizada em duas etapas de 4 horas. Na primeira parte é definido o que será entregue como resultado do incremento do próximo *sprint* e na segunda parte, como o trabalho necessário para entregar o incremento será realizado.
- Reunião diária - tem como objetivo avaliar o andamento do *sprint* e resolver qualquer problema que possa estar ocorrendo. Esta reunião ocorre com todos de pé, com duração máxima de 15 minutos composta pelo *Product Owner*, o Scrum Master e o time Scrum.
- Revisão do *sprint* - é realizada no final do *sprint* para inspecionar o incremento e adaptar o *backlog* do produto se necessário. É a reunião onde o time mostra ao P.O. (*Product Owner*) o que foi realizado e responde a questionamentos. Neste momento, o P.O. aceita ou não o resultado de cada história do *sprint* e gera discussões que servem como entradas para o planejamento dos *sprints* seguintes.
- Retrospectiva do *sprint* - ocorre após a Revisão do *sprint* e antes da reunião de planejamento do próximo *sprint*. Esta é uma reunião com duração de três horas para um *sprint* de um mês. É uma oportunidade para o time inspecionar a si próprio e criar

um plano para melhorias a serem aplicadas no próximo *sprint*. O resultado desta reunião é uma lista de itens que foram realizados com sucesso e outra dos que devem ser melhorados para o próximo *sprint*.

2.4 Kanban

O Kanban é um método ágil de desenvolvimento de software baseado nas práticas Lean e tem como foco o trabalho em progresso, apresentando a evolução de forma visual, tornando os problemas evidentes e favorecendo uma cultura de melhoria contínua (Santos, 2011).

Ao contrário do Scrum, não são definidos papéis, práticas ou cerimônias específicas. O Kanban oferece uma série de princípios para otimizar o fluxo e a geração de valor do sistema de entrega de software (Boeg, 2012).

Segundo Boeg (2012), o Kanban enfatiza os seguintes princípios:

- Visualizar o trabalho em andamento;
- Visualizar cada passo em sua cadeia de valor, do conceito geral até o software que se possa lançar;
- Limitar o Trabalho em Progresso (WIP, do inglês “*Work in Progress*”), restringindo o total de trabalho permitido para cada estágio;
- Tornar explícitas as políticas que estão sendo seguidas;
- Medir e gerenciar o fluxo, para tomar decisões bem embasadas, além de visualizar as consequências dessas decisões;
- Identificar oportunidades de melhorias, criando uma cultura *Kaizen*, na qual a melhoria contínua é responsabilidade de todos.

A organização visual é uma das principais características do Kanban e por isso a ferramenta mais utilizada para a aplicação de Kanban é o quadro branco, com *post-its* ou cartões que representam as histórias. O quadro é dividido em colunas que sinalizam os diferentes status de uma história e deve ser dividido de acordo com as necessidades do processo adotado pela organização, visto que a metodologia não define um padrão. Ao visualizar o fluxo e estabelecer limites de trabalho em progresso, garante-se que não

é possível introduzir trabalho além da capacidade do sistema de processar este trabalho. É necessário que se complete o trabalho existente antes que um novo possa ser iniciado. Isto resulta em funcionalidades sendo processadas pelo sistema, com base na capacidade dele, ao invés de serem empurradas com base em previsões ou demandas (Boeg, 2012). A Figura 2 apresenta um exemplo do quadro utilizado no Kanban.

Tarefas	Especificação	Pronto para desenvolvimento	Em desenvolvimento			Revisão de código	Teste local	Teste em homologação	Concluído
			Planejado	Em andamento	Concluído				
									

Figura 2 - Exemplo de Quadro Kanban.

Fonte: Elaborado pelas autoras a partir de Boeg, 2012.

2.5 Extreme programming (XP)

O Extreme Programming (XP) é uma metodologia ágil de desenvolvimento de software que surgiu nos Estados Unidos ao final da década de 90, com foco na agilidade de equipes e na criação de sistemas de melhor qualidade. Esta metodologia busca dividir as iterações em menor escala, onde o planejamento é constantemente revisto e repriorizado com o cliente, objetivando sempre entregar maior valor no software, aumentando o retorno de investimento (*Return on Investment*) (Wildt e Lacerda, 2010). Para alcançar as melhorias citadas, o XP propõe um conjunto de valores, práticas e princípios.

Os valores do XP são:

- **Comunicação:** Tem como objetivo garantir que todos os envolvidos em um projeto se compreendam da melhor maneira possível (Teles, 2006), evitando possíveis falhas no entendimento devido a ruídos na comunicação entre o cliente e a equipe do projeto.
- **Coragem:** Propõe a confiança nos mecanismos de segurança utilizados para proteger o projeto. Ao invés de acreditar que os problemas não ocorrerão e fazer com que a coragem se fundamente nesta crença, projetos XP partem do princípio de que

problemas irão ocorrer, inclusive aqueles mais temidos. Entretanto, a equipe utiliza redes de proteção que possam ajudar a reduzir ou eliminar as consequências destes problemas (Teles, 2005). Os testes automatizados e as curtas iterações que permitem a pronta detecção de falhas de entendimento ou implementação são exemplos de mecanismos de segurança do XP.

- Feedback: O XP é organizado em ciclos curtos de feedback de modo a possibilitar aos usuários solicitar funcionalidades e aprender sobre elas através de software funcionando em prazos curtos. Assim, o usuário pode detectar eventuais falhas o mais breve possível, quando tende a ser mais barato corrigi-las (Teles, 2005).
- Respeito: Saber ouvir, saber compreender e respeitar o ponto de vista do outro é essencial para que um projeto de software seja bem sucedido (Teles, 2006).
- Simplicidade: Visa assegurar que a equipe se concentre em fazer, primeiro, apenas aquilo que é claramente necessário e evite fazer o que poderia vir a ser necessário, mas ainda não se provou essencial (Teles, 2006).

As práticas são descritas a seguir (Wildt e Lacerda, 2010):

- Equipe (*Whole Team*): todos em um projeto XP são partes da equipe, integrando os clientes à equipe de desenvolvimento. Um membro da equipe pode assumir mais de um papel.
- Clientes no Local (*Client on Site*): Para total funcionamento do XP, é necessário que o cliente se integre à equipe de desenvolvimento;
- Jogo do Planejamento (*Planning Game*): Nesta prática são definidas as histórias que descrevem as funcionalidades a serem implementadas, bem como estimativas e prioridades. É um feedback para que cliente possa dirigir o projeto, podendo-se ter uma ideia clara do avanço do projeto minimizando os riscos;
- Testes de Aceitação (*Customer Tests*): São testes elaborados pelo cliente, sendo os critérios de aceitação do software. Devem ser rodados a cada interação futura e oferecem feedback do desenvolvimento da aplicação;

- Pequenas entregas (*Small Releases*): A cada interação, é disponibilizado o software 100% funcional. Desta forma, são disponibilizadas pequenas versões para que o cliente possa obter algo de valor o mais cedo possível, minimizando os riscos;
- Posse Coletiva (*Collective Ownership*): Em um projeto XP, qualquer dupla de programadores pode melhorar o software a qualquer momento. O código tem um único dono: a equipe. Todos compartilham a responsabilidade pelas alterações;
- Integração Contínua (*Continuous Integration*): O XP mantém o projeto integrado continuamente, expondo o estado atual de desenvolvimento (viabiliza lançamentos pequenos e frequentes), oferecendo um feedback sobre todo o sistema, em qualquer etapa do processo;
- Metáfora (*Metaphor*): A equipe mantém uma visão compartilhada do funcionamento do software. Serve de base para estabelecimento dos padrões de codificação e da forma de comunicação com o cliente. Uma metáfora deve funcionar como uma forma de comunicação comum entre equipe e cliente. Em se tratando da metáfora do software, é entender o que precisa ser desenvolvido, entender a visão do produto que está sendo construindo;
- Ritmo Saudável (*Sustainable Pace*): Os projetos XP procuram obter a maior produtividade dos programadores e entregar o software na melhor qualidade possível, obtendo a satisfação do cliente. Para isso há uma grande preocupação com o cronograma. Definir um número de horas de trabalho, visando gerar um ritmo sustentável é fundamental para o maior rendimento da equipe;
- Padrões de Codificação (*Coding Standards*): Todo o código escrito segue o padrão definido pela equipe, facilitando a comunicação e refinamento do design;
- Testes (*Test-Driven Development*): Todo o desenvolvimento XP é guiado por testes. Os testes dão segurança necessária para garantir a qualidade de forma contínua, dão coragem para mudar e são a base do feedback do código para o programador;
- Programação em Pares (*Pair Programming*): Todo o desenvolvimento é feito em pares, obtendo uma melhor qualidade do design, código e testes, uma revisão constante do código e uma maior comunicação;

- Design Simples (*Simple Design*): O código está, a qualquer momento, na forma mais simples para a realização dos testes. Esta prática está presente em todas as outras etapas do XP.
- Refinamento (*Refactoring*): O design é melhorado continuamente através do refinamento. Em XP, o código é o próprio design. O refinamento é um processo formal realizado através de etapas reversíveis, melhorando a estrutura do código sem alterar sua função. O refinamento do código deve ser feito sempre após um teste que passa. Isto dá liberdade ao desenvolvedor de modificar o código fonte, pois sabe que os testes devem continuar passando. Se um teste passa a falhar, alguma coisa na lógica da aplicação foi alterada e precisa ser revista;
- Reuniões em Pé (*Stand Up Meetings*): Além das cerimônias de planejamento de release e iterações, os *Stand Up Meetings* ajudam a expor para o time como está o desenvolvimento das tarefas. Nestas reuniões, é exposto o andamento do trabalho (o que se fez ontem, o que está sendo feito, quais são os problemas), de forma que a equipe possa avaliar as dificuldades e analisar possíveis soluções.
- Spikes de Planejamento (*Spikes Solution*): Os spikes forçam que, em um momento apropriado e necessário, seja realizada uma análise de recursos tecnológicos (em geral, ferramentas, frameworks, APIs) para inclusão nas tecnologias usadas no projeto. Os spikes de planejamento ajudam o time a gerar o conhecimento necessário para que possam estimar novas funcionalidades a serem desenvolvidas no projeto.

Os princípios básicos do XP são os seguintes (Sato e Bassi, 2007):

- Feedback rápido;
- Simplicidade é o melhor negócio;
- Mudanças incrementais;
- Carregue a bandeira das mudanças / não valorize o medo;
- Alta qualidade do código.

2.6 Uso de histórias para definição do escopo

As histórias, também conhecidas como *User Stories*, são comumente utilizadas em metodologias ágeis para representar de forma breve, da perspectiva do usuário, uma funcionalidade ou característica do sistema e são suficientemente pequenas para que os desenvolvedores possam implementar um pequeno conjunto delas a cada iteração (Teles, 2005). Na Figura 3 são apresentados exemplos de histórias de usuário da ferramenta Histimate.

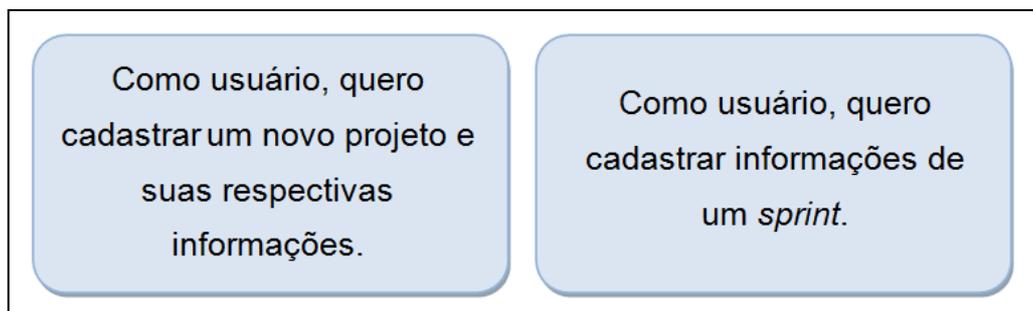


Figura 3 - Exemplos de histórias da ferramenta Histimate.

Fonte: Elaborado pelas autoras.

Nas reuniões de planejamento, ao início de cada ciclo de iteração, as histórias são estimadas pela equipe de desenvolvimento. Com base na velocidade de desenvolvimento do ciclo anterior, os desenvolvedores indicam a quantidade de histórias que poderão ser implementadas na iteração que se inicia. De posse dessas informações, são priorizadas as histórias que, em conjunto, mais possam gerar valor para o cliente quando estiverem implementadas ao final do ciclo (Teles, 2006).

A criação dos testes de aceitação também é guiada pelas histórias. Estes testes tem como objetivo validar e verificar se a história foi implementada corretamente, garantindo que o desenvolvimento está de acordo com as solicitações do cliente.

O Acrônimo INVEST foi criado por Bill Wake em 2003 para representar as características das boas histórias. Estas características são (Wake, 2003):

- Independência (*Independent*): As histórias são mais fáceis de trabalhar se forem independentes, permitindo a implementação sem ordem específica;
- Negociável (*Negotiable*): Uma história não é um contrato. Os detalhes serão criados pelo cliente e pelo desenvolvedor durante o desenvolvimento;

- Valiosa (*Valuable*): A história deve agregar valor para o cliente;
- Estimável (*Estimable*): Uma boa história deve ser estimável. Não é preciso fazer uma estimativa exata, mas o suficiente para auxiliar a priorização;
- Pequena (*Small*): As boas histórias tendem a ser pequenas, de modo a facilitar o entendimento do escopo da história;
- Testável (*Testable*): Deve ser possível definir os testes para verificar se a história foi implementada corretamente.

2.7 Estimativas em projetos de software tradicionais

Segundo Hazan (2008), em um projeto de software tradicional, as estimativas devem ser realizadas no início do processo de desenvolvimento de software. Desta forma, o artefato de entrada utilizado para o processo de estimativas é frequentemente um documento inicial de requisitos ou até mesmo um documento do próprio cliente. O responsável pelas estimativas deve analisar os requisitos para garantir a qualidade e, então, estimar o tamanho do projeto de software. No decorrer do processo de desenvolvimento, as estimativas devem ser acompanhadas conforme o refinamento dos requisitos. O projeto deve ser re-estimado se ocorrerem mudanças significativas nos requisitos funcionais ou não funcionais.

Podem-se destacar três métricas tradicionais para estimativas (Hazan, 2008):

- LOC (*Line of Code*): Consiste na contagem de linhas de código escritas;
- Pontos por Casos de Uso: Tem como propósito estimar recursos para projetos de software orientados a objeto modelados por meio de especificação de Casos de Uso;
- Pontos de Função: É uma medida de tamanho funcional de projetos de software, considerando as funcionalidades implementadas, sob o ponto de vista do usuário.

Para entender porque estimar no início dos projetos é um problema, é preciso analisar o Cone da Incerteza (Figura 4), que foi apresentado por Barry Boehm em 1981 e utilizado por Steve McConnell em 1997 no livro *Guia de Sobrevivência de Projeto de Software* (Lacey, 2012). O cone traz faixas de incerteza para diferentes etapas de um projeto no modelo cascata (*waterfall*).

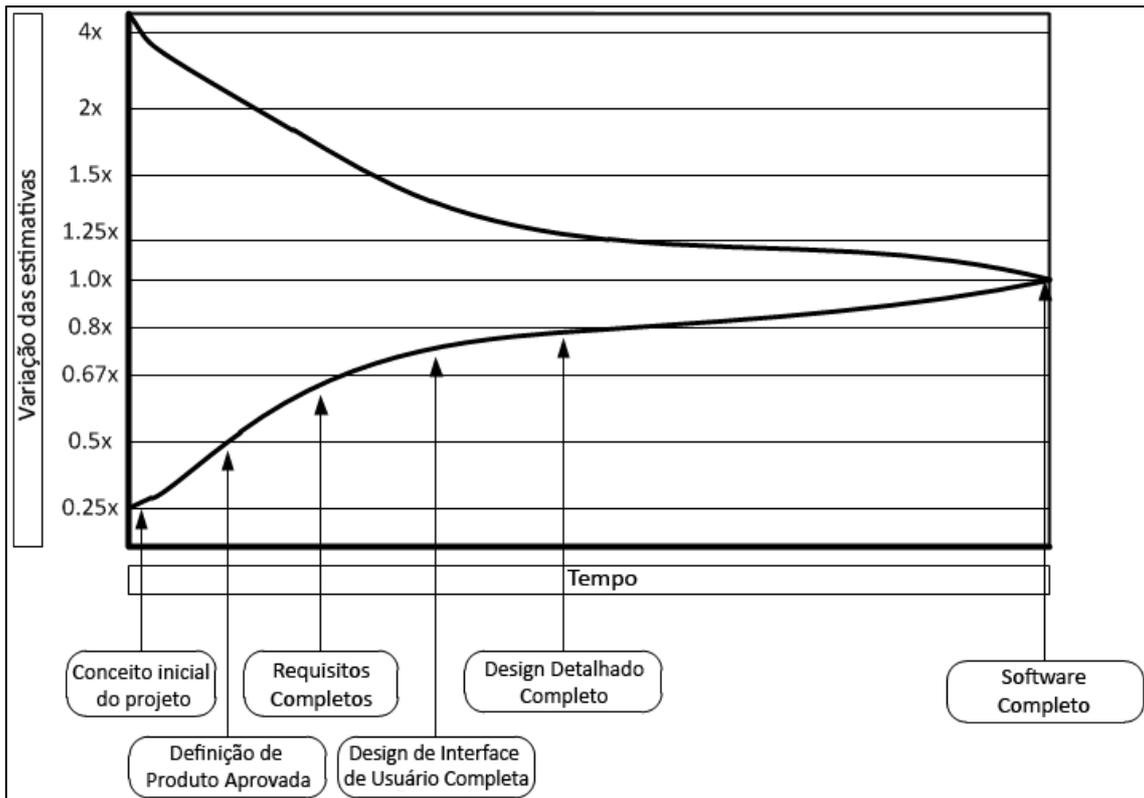


Figura 4 - Cone da Incerteza.

Fonte: Adaptado de Lacey, 2012.

O cone demonstra que tem-se o maior nível de incerteza no início de qualquer projeto (uma variação de 4x a 0,25x no intervalo) (Lacey, 2012) e conforme o projeto vai progredindo, o intervalo de incerteza diminui. Embora o início de um projeto seja a fase que apresenta mais incertezas, é também quando são solicitadas estimativas precisas.

Cohn (2005) aponta cinco razões para o planejamento tradicional ser falho:

- 1 *O planejamento é feito baseado nas tarefas necessárias e não nas funcionalidades desejadas:* As abordagens tradicionais se preocupam mais com a conclusão de tarefas que com a entrega de funcionalidades. A entrega de tarefas não agrega valor, ao contrário da entrega de funcionalidades, que traz o retorno esperado pelo cliente. Outro problema identificado causado pelo foco nas tarefas é a dependência entre elas. Ou seja, o atraso de uma tarefa acaba impactando outras e afeta o cronograma do projeto;
- 2 *A execução simultânea de tarefas gera atrasos posteriores:* Em um primeiro momento, a execução simultânea de tarefas pode ser vista como uma boa forma de

evitar atrasos e permite que, se o time encontrar dificuldades em uma das tarefas, possa alternar para outra. Entretanto, como os membros do time estão trabalhando em tarefas simultâneas, é provável que alguma destas se atrase e com a dependência entre tarefas aprontada no item anterior, este atraso pode se propagar pela cadeia de dependência entre as tarefas;

- 3 *As funcionalidades não são implementadas de acordo com sua prioridade:* Muitos dos planejamentos tradicionais são elaborados assumindo que todas as tarefas serão entregues e não priorizam as tarefas que trazem mais valor aos usuários e ao cliente. Ao final do projeto, visando manter o cronograma, o time acaba por abandonar algumas tarefas. Como não houve priorização das tarefas no planejamento, as tarefas abandonadas podem ser de maior valor para o cliente que as que serão entregues;
- 4 *As incertezas são ignoradas:* Nos métodos tradicionais de planejamento, assume-se que a análise dos requerimentos iniciais leva a uma completa e perfeita especificação. Assume-se também que os usuários que não mudarão de ideias, não refinarão suas opiniões e não terão novas necessidades durante a execução do projeto. Também é ignorada a incerteza acerca de como o produto será implementado. A melhor maneira de lidar com essas incertezas é utilizando curtas iterações, que possibilitam *feedbacks* constantes.
- 5 *As estimativas se tornam compromissos:* Associada a cada estimativa existe a probabilidade, que varia de 0% a 100%, de o trabalho ser concluído no tempo estimado. Logo, quando uma tarefa é estimada, significa que existe uma probabilidade de $x\%$ de a tarefa ser concluída em y dias. Se as estimativas forem vistas como compromissos, configura-se um problema, pois as estimativas são probabilidades e os compromissos não podem ser baseados em probabilidades, e sim em datas.

2.8 Estimativas em projetos de software ágeis

Em um projeto ágil não é incomum iniciar uma iteração com requisitos incompletos e os detalhes serem descobertos durante o ciclo. No entanto, é preciso associar uma estimativa a cada história, mesmo aquelas que não estão completamente definidas (Cohn, 2005).

As principais técnicas de estimativas ágeis são:

- *Story Points*: É uma unidade de medida para expressar o tamanho de uma história, funcionalidade ou qualquer outra atividade. Na estimativa com *Story Points* é atribuído um valor (ponto) a cada item. Não existe fórmula para definir o tamanho de uma história. A estimativa de um *Story Point* é uma junção do esforço envolvido no desenvolvimento da funcionalidade, a complexidade da implementação, o risco inerente na implementação e de outros fatores. Existem duas maneiras comuns de começar a estimar com *Story Points*. A primeira abordagem é selecionar uma história que se espera que seja a menor das histórias trabalhadas e atribuir-lhe um ponto. A segunda abordagem é selecionar uma história que parece de tamanho médio e atribuir-lhe cinco pontos. Atribuindo um valor a primeira história, cada história adicional é estimada comparando-se com a primeira ou com outras que já foram estimadas (Cohn, 2005). Dessa forma, tem uma escala de comparação na qual, por exemplo, uma história estimada em dois pontos deve ser duas vezes maior que uma história estimada em um.
- *Planning Poker*: Combina a opinião de um especialista, analogia e desagregação em uma abordagem de estimativas. Os participantes que realizam as estimativas são os integrantes do time ágil: desenvolvedores, testadores, engenheiros de banco de dados, analistas, designers de interface, entre outros. O *Product Owner* participa, mas não estima. Cada participante que irá estimar recebe um jogo de cartas, no qual cada carta corresponde a um valor de estimativa válido. O moderador - que pode ser o *Product Owner*, um analista ou qualquer um que não faça parte do time – lê a descrição de cada história e o *Product Owner* responde eventuais dúvidas que os participantes que farão as estimativas possam ter. Após a resolução de dúvidas, cada participante seleciona uma carta que representa a estimativa e as cartas não são apresentadas aos outros participantes até que todos tenham feito suas escolhas. As cartas são viradas de uma vez e em caso de divergências, a equipe deve chegar a um consenso da estimativa (Cohn, 2005).
- *Dias Ideais (Ideal Days)*: Tempo ideal é a quantidade de tempo que uma atividade demanda sem o impacto de atividades periféricas. Quando estima-se em dias ideais, é preciso assumir que: (i) a história que está sendo estimada será a única a ser

trabalhada; (ii) tudo que for necessário estará a mão quando a implementação for iniciada e que não ocorrerão interrupções. Ao estimar o número de dias ideais que uma história irá demandar para ser desenvolvida, testada e aceita, não é necessário considerar os fatores do ambiente em que o time trabalha. Se o desenvolvimento de uma tela custa um dia ideal, então custará um dia ideal independente do ambiente da equipe. Quando estes fatores são ignorados, os dias ideais podem ser vistos como outra estimativa de tamanho, assim como os *Story Points*. Então, a estimativa de tamanho expressa como o número de dias ideias pode ser convertida em uma estimativa de duração usando velocidade da mesma maneira que os *Story Points* (Cohn, 2005).

- *T-Shirt size*: Consiste no uso de tamanhos de camisas (*T-shirts*) para estimativas. Esta técnica apresenta duas grandes desvantagens: A diferença no julgamento dos tamanhos - membros da equipe podem divergir sobre a interpretação de um determinado tamanho - e elas não são aditivas, ou seja, não é possível afirmar que o trabalho estará concluído após três médias, quatro grandes e duas extra pequenas. Para contornar essas duas desvantagens, os times costumam assumir valores para cada tamanho, por exemplo, uma história média é cinco e uma grande é dez. A grande vantagem desta técnica é a facilidade de adoção e pode ser uma boa forma de se acostumar a estimativas relativas em um primeiro momento, para depois adotar as estimativas por valores (números) (Cohn, 2013).

Segundo Cohn (2005), velocidade é a medida da taxa de progresso do time. É calculada através da soma do número de *Story Points* atribuído a cada história que o time concluiu durante a iteração. Se o time concluiu três histórias estimadas em cinco pontos, sua velocidade é quinze. Se um time concluiu dez *Story Points* na última iteração, supõe-se que também conclua dez pontos nesta iteração. Como *Story Points* são estimativas de tamanho relativo, esta suposição será verdadeira tanto se o time trabalhar em duas histórias de cinco pontos como se trabalhar em cinco histórias de dois pontos. Através da soma dos *Story Points* das estimativas de todas as funcionalidades desejadas, chega-se ao tamanho estimado do projeto. Se a velocidade do time for conhecida, é possível dividir o tamanho do projeto pela velocidade para encontrar o número estimado de iterações.

Os times ágeis valorizam mais software funcionando a uma boa documentação porque isto os leva a ter uma versão estável e incrementável do produto ao fim de cada iteração. Isto torna possível receber um rápido e frequente *feedback* tanto do produto quanto do processo. O *feedback* dos usuários é repassado ao processo de desenvolvimento para assegurar que o time está sempre trabalhando nas funcionalidades mais valiosas e que irão satisfazer as expectativas do usuário. A preocupação em entregar o maior valor possível ao cliente e usuários também faz com que o time ágil se preocupe mais com a rápida resposta a mudanças que em seguir um planejamento (Cohn, 2005).

2.9 Considerações finais

Neste capítulo foi apresentada uma visão geral sobre os métodos ágeis e o Manifesto Ágil. Também foram apresentadas as metodologias Scrum, Kanban e XP, além da importância das histórias para definição do escopo e as estimativas em projetos de software tradicionais e ágeis.

Um dos princípios defendidos pelo Manifesto Ágil é a reflexão da equipe (time ágil) em como se tornar mais efetiva (Fadel e Silveira, 2010). Tal reflexão possibilita a equipe a aprender com os erros cometidos e se ajustar visando à melhoria contínua.

Segundo Cohn (2005), o planejamento ágil balanceia o esforço e investimento no planejamento, sabendo que este será revisado durante o projeto. Um plano ágil é aquele em que não há a vontade de mudança, mas sim a ansiedade em mudar. Esta mudança significa que houve algum aprendizado ou que algum erro foi evitado. É possível aprender que os usuários querem mais ou menos de uma funcionalidade ou que a usabilidade é mais importante que se acreditava no início do planejamento ou que o desenvolvimento em uma linguagem nova é mais custoso que se esperava. O impacto financeiro destas mudanças pode ser avaliado e, se for válido, pode alterar o planejamento e cronograma.

3 Especificação e funcionalidades da ferramenta

Este capítulo descreve a ferramenta, apresentando os elementos envolvidos em sua especificação e implementação, além de demonstrar suas telas e funcionalidades.

3.1 A ferramenta

Diante da importância de as organizações terem bases históricas de estimativas, apontada no Capítulo 1, este trabalho propõe o desenvolvimento de uma ferramenta que auxilie a criação e uso destas bases em organizações que utilizam métodos ágeis e consequentemente, as técnicas de estimativas comuns a estes métodos.

Espera-se que a equipe aprenda com os erros e acertos nas estimativas das histórias e tarefas dos ciclos anteriores. Porém, a falta de um registro histórico destas estimativas dificulta tanto a identificação dos erros nas estimativas, como o acompanhamento da evolução do aprendizado da equipe.

Para a criação da base histórica de estimativas, a ferramenta utilizará os dados armazenados de tamanho estimado das histórias e o esforço estimado e esforço realizado das tarefas. A base criada servirá como apoio para as estimativas dos ciclos posteriores.

Para as estimativas de histórias, a ferramenta possibilitará a utilização de métricas como *T-Shirt size* (descrita no capítulo anterior) que não são relacionadas a pontos. Entretanto, para as estimativas de esforço das tarefas, a ferramenta irá considerar apenas medidas de pontos. Segundo Sutherland (2013), as horas para implementar uma história (ou tarefa) dependem do responsável por esta implementação e essas horas podem variar com frequência. As estimativas em pontos são melhores que as estimativas em horas, pois são mais precisas e tem menos variações.

A base de estimativas será única por projeto, pois os projetos de uma organização podem apresentar características muito distintas. Uma das propostas da ferramenta é analisar histórias com tamanhos iguais, e dentro de um projeto, as estimativas são definidas considerando o mesmo caso base, ou seja, a história mais simples a ser implementada é estimada em um valor e esta relação é considerada para as estimativas de outras histórias (Conforme apresentado na Seção 2.8 deste trabalho). No contexto de uma organização, as histórias de diferentes projetos podem ter o mesmo tamanho, porém com casos bases diferentes. Logo, a criação de uma base única para a organização poderia impactar a análise das estimativas e gerar dados inconsistentes.

Dessa forma, a ferramenta apoiará:

- O armazenamento do esforço estimado para cada história e cada tarefa no planejamento dos ciclos iterativos e, também, do esforço real;
- A análise da relação entre o tamanho das histórias e o esforço das tarefas a partir dos dados históricos armazenados com o objetivo de indicar se as estimativas de histórias estão proporcionais às estimativas de tarefas e vice-versa;
- A análise da relação entre o tamanho padrão do *sprint* do time, as histórias e tarefas estimadas, indicando se o tamanho do *sprint* pode ser ampliado ou reduzido;
- Acompanhamento da evolução da implementação das tarefas, através de um quadro (*board*) com os status de tarefas definidos para o projeto;

3.2 Modelagem

Para a especificação da ferramenta, foi elaborado um documento com a definição da dinâmica de um projeto ágil, o diagrama de classes e as atividades necessárias à implementação das funcionalidades foram divididas em histórias de usuário.

3.2.1 Definição da dinâmica de um projeto ágil

Durante o planejamento da ferramenta foi elaborado um documento com as definições das variáveis presentes na dinâmica de um projeto ágil, visando identificar os requisitos da ferramenta. As definições relativas às variáveis de um projeto ágil para este trabalho são:

- Uma história é decomposta em uma ou mais histórias;
- Uma história deriva uma ou mais tarefas;
- Uma tarefa é estimada com base em esforço;
- Uma tarefa tem diferentes complexidades;
- A soma dos pontos/tamanhos define o escopo do *sprint*;
- O esforço do *sprint* está associado ao tamanho das tarefas;
- O *sprint* acontece em um espaço de tempo limitado;
- Os *sprints* geralmente tem a mesma duração de tempo;
- O esforço do *sprint* está limitado ao tempo disponível;
- O esforço do *sprint* depende do tamanho da equipe/time;
- O esforço pode ser esforço real ou esforço estimado;
- O esforço real é obtido ao final do *sprint*;
- O esforço estimado é calculado ao início do *sprint*;
- Um *sprint* é composto por n tarefas;
- Uma equipe/time é composta por uma ou mais pessoas;
- Equipe/time entra em consenso sobre o tamanho da história;

3.2.2 Diagrama de Classes

Após a identificação das variáveis e requisitos necessários no processo relatado na seção anterior, foi elaborado o diagrama de classe (Figura 5):

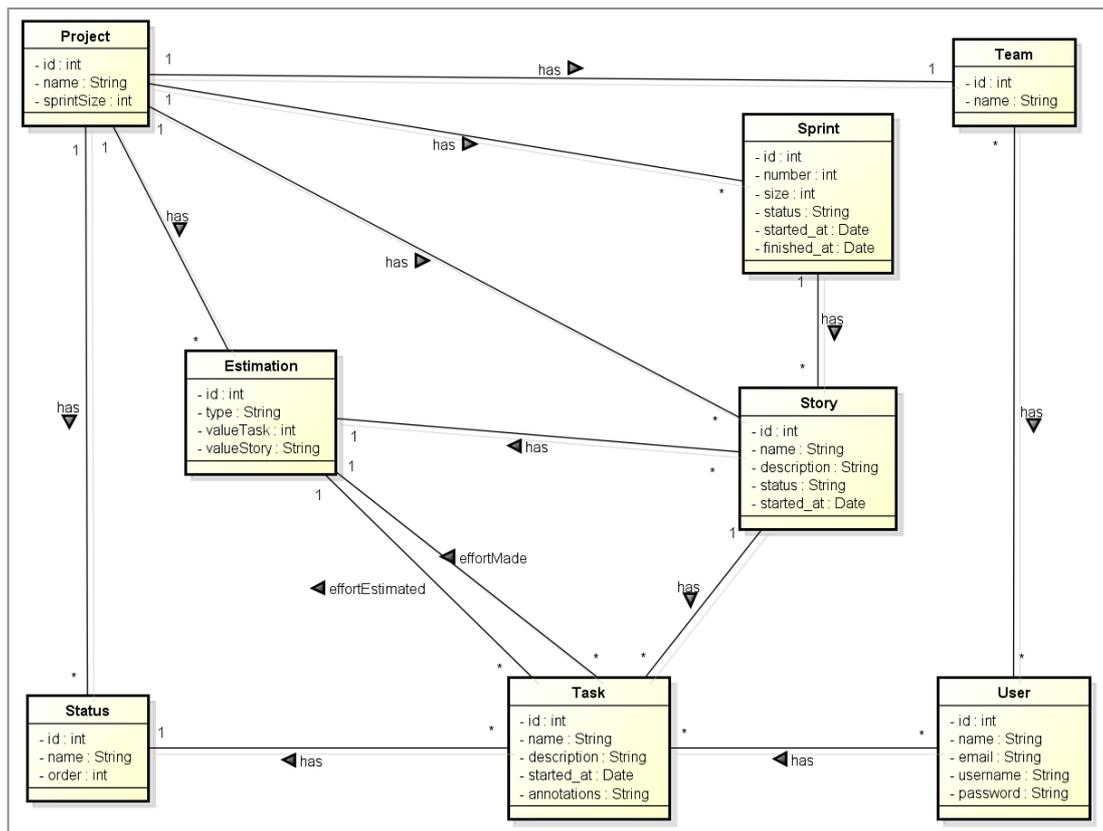


Figura 5 - Diagrama de classes da ferramenta.

As classes e seus respectivos atributos identificados foram:

- *Project*: Representa o projeto ágil a ser analisado. Possui os atributos id (identificador), name (nome do projeto) e sprintSize (tamanho padrão dos *sprints* do projeto). O projeto pode ter somente um time (classe *Team*) alocado, ter muitos sprints (classe *Sprint*) associados, ter diversas histórias (classe *Story*) a serem implementadas, ter muitas estimativas (classe *Estimation*) possíveis e diferentes status (classe *Status*) cadastrados para as tarefas.
- *Team*: Representa o time alocado no projeto. Possui os atributos id (identificador) e name (nome do time). O time pode estar alocado em apenas um projeto (classe *Project*) e possui muitos usuários (classe *User*) associados, que são os membros do time.
- *User*: Representa os usuários do sistema e membros do time. Possui os atributos id (identificador), name (nome do usuário), email (email do usuário), username (nome do usuário no sistema) e password (senha do usuário no sistema). O usuário pode

estar alocado em diferentes times (classe *Team*) e pode estar associado a diversas tarefas (classe *Task*).

- *Sprint*: Representa os *Sprints* do projeto. Possui os atributos id (identificador), number (número do *sprint*), size (tamanho em pontos do *sprint*), status (status pré-definidos que indicam a situação do *sprint*), started_at (data de início do *sprint*) e finished_at (data de término do *sprint*). O *sprint* pode estar associado somente a um projeto (classe *Project*) e pode possuir muitas histórias (classe *Story*).
- *Story*: Representa as histórias associadas ao projeto. Possui os atributos id (identificador), name (nome da história), description (descrição da história), status (status pré-definidos que indicam a situação da história) e started_at (data de início da implementação da história). A história pode estar associada somente a um *sprint* (classe *Sprint*), pode relacionar-se a um projeto (classe *Project*), pode possuir uma estimativa (classe *Estimation*) e diversas tarefas relacionadas (classe *Task*).
- *Task*: Representa as tarefas associadas às histórias do projeto. Possui os atributos id (identificador), name (nome da tarefa), description (descrição da tarefa), started_at (data de início da tarefa) e annotations (observações da tarefa). A tarefa pode associar-se a uma história (classe *Story*), a uma estimativa (classe *Estimation*), pode estar atribuída a um ou mais usuários (classe *User*) e pode possuir um status (classe *Status*). *Task* apresenta dois relacionamentos com a classe *Estimation*, um representa o esforço estimado, enquanto o segundo representa o esforço realizado.
- *Estimation*¹: Representa as estimativas cadastradas no projeto que podem ser utilizadas para estimar histórias ou tarefas. Possui os atributos id (identificador), type (indica o tipo da estimativa: se é de esforço, para tarefas, ou de tamanho, para histórias), valueTask (valor da estimativa, caso o tipo seja esforço) e valueStory (valor da estimativa, se o tipo for tamanho). A estimativa pode estar associada a um projeto (classe *Project*), pode ser atribuída a muitas histórias (classe *Story*) e pode ser atribuída a muitas tarefas (classe *Task*).
- *Status*: Representa os status possíveis para cada tarefa. Possui os atributos id (identificador), name (nome do status) e order (ordem de precedência do status). O

¹ O atributo 'type' define qual atributo será utilizado: valueTask, para estimativas de tarefa ou valueStory, para estimativas de história. Esta foi uma decisão de projeto visando simplificar a modelagem e a implementação. Uma alternativa possível seria a criação de classes distintas para cada tipo de estimativa.

status pode associar somente a um projeto (classe *Project*) e pode associar-se a diversas tarefas (classe *Task*).

Todas as variáveis envolvidas na especificação e consequente implementação da ferramenta foram definidas na língua inglesa, como pôde ser observado no modelo de classes e nas capturas de tela da ferramenta que serão apresentadas na Seção “Funcionalidades”. Esta escolha visa facilitar principalmente a manutenção de código, visto que a linguagem utilizada na implementação - Ruby - possui sua gramática toda escrita nesta língua, assim como a grande maioria das linguagens de programação.

3.2.3 Histórias de usuário

Com a definição dos requisitos e das classes com seus respectivos atributos, foram elaboradas as histórias de usuário para representar as atividades necessárias para a implementação da ferramenta e suas funcionalidades.

As histórias de usuário são:

1. História: Como usuário, quero me cadastrar no sistema.
2. História: Como usuário, quero logar no sistema.
3. História: Como usuário, quero cadastrar um novo projeto e suas respectivas informações.
4. História: Como usuário, quero cadastrar as estimativas de histórias e tarefas do projeto.
5. História: Como usuário, quero cadastrar os status possíveis das tarefas do projeto (que serão as colunas do board do projeto).
6. História: Como usuário, quero cadastrar equipes.
7. História: Como usuário, quero definir a equipe do projeto no momento da criação do projeto.
8. História: Como usuário, quero cadastrar uma história definindo seu nome, descrição e tamanho (dentre as opções de escala).

9. História: Como usuário, quero cadastrar um sprint e suas respectivas informações.
10. História: Como usuário, quero selecionar as histórias que farão parte do sprint.
11. História: Como usuário, na fase de detalhamento do sprint, quero escrever tarefas relacionadas a cada história selecionada para o sprint e estimar o esforço de cada uma delas.
12. História: Como usuário, quero verificar o tamanho estimado para o sprint e obter uma comparação com o tamanho padrão cadastrado para o sprint do projeto relacionado.
13. História: Como usuário, quero obter uma análise de estimativas relacionando o tamanho de uma história com a soma de esforços das tarefas relacionadas. Deverá ser feita uma comparação com base nas histórias e tarefas já concluídas anteriormente no mesmo projeto.
14. História: Como usuário, quero iniciar um sprint.
15. História: Como usuário, quero mudar status das tarefas e me tornar responsável por sua execução.
16. História: Como usuário, quero cadastrar o esforço real da tarefa que implementei quando esta estiver na última coluna do board.
17. História: Como usuário, quero visualizar o board com os status cadastrados para o projeto e as tarefas de acordo com seu status.
18. História: Como usuário, quero finalizar um sprint.
19. História: Como usuário, após finalizado o sprint, quero obter um resumo do sprint concluído. Este resumo deve conter comparações de esforço estimado/realizado das tarefas, velocidade do sprint e status final das tarefas e histórias.

3.3 Tecnologias utilizadas

Nesta seção serão apresentadas as tecnologias utilizadas no desenvolvimento da ferramenta proposta.

3.3.1 Linguagem de programação

A linguagem de programação Ruby (<https://www.ruby-lang.org/pt/>) foi selecionada para a implementação da ferramenta, devido ao seu fácil entendimento e por sua recente difusão no desenvolvimento de sistemas, em especial os web. Ruby é uma linguagem de programação dinâmica que traz uma gramática complexa e expressiva e uma biblioteca com uma rica API (*Application Programming Interface* ou Interface de Programação de Aplicativos). É inspirada em Lisp, Smalltalk e Perl, porém utiliza uma gramática de fácil compreensão para o entendimento de programadores C e Java. É uma linguagem orientada a objeto, mas também pode ser conveniente para programação funcional e procedural (Flanagan e Matsumoto, 2008). Ruby é compatível com os ambientes Windows, Linux e MacOS.

Entre as principais características do Ruby, podem-se destacar:

- Orientação a objeto: Ruby é uma linguagem completamente orientada a objeto. Desta forma, cada valor é um objeto, até tipos básicos como *strings* e números (Flanagan e Matsumoto, 2008);
- Iteradores e blocos: Iteradores são métodos que executam um bloco de código que lhes seja atribuído. Blocos são delimitados por “{ ... }” ou “do ... end” e podem receber argumentos, declarados entre “| ... |”. Após a definição do iterador, o bloco recebido é executado pelo comando “yield”, e os argumentos passados a “yield” serão atribuídos aos argumentos do bloco por atribuição múltipla. Em muitos casos, iteradores são úteis em substituição a estruturas de laço (Ruby Brasil, 2008);
- Expressões e operadores: A sintaxe de Ruby é orientada a expressões. Estruturas de controles que seriam chamadas de declarações em outras linguagens são expressões em Ruby. Muitos dos operadores presentes em Ruby são implementados por métodos e as classes podem definir (ou redefinir) estes métodos (Flanagan e Matsumoto, 2008);
- Métodos: São definidos com a palavra reservada “def”. O valor de retorno de um método pode ser uma expressão associada a um “return” ou a última expressão avaliada (Flanagan e Matsumoto, 2008);

- Atribuição: O operador de atribuição é “=” e pode ser combinado a outros operadores como “+” (incremento) e “-” (decremento). Ruby suporta atribuições múltiplas, permitindo mais de um valor e mais de uma variável nas atribuições. Este tipo de atribuição é útil em métodos que retornam mais de um valor (Flanagan e Matsumoto, 2008);
- Prefixos e sufixos: Alguns caracteres de pontuação são utilizados para definir as variáveis ou métodos em Ruby. O ponto de interrogação é utilizado para indicar predicados - métodos que retornam um valor booleano. Um sinal de exclamação ao fim do nome do método indica que é preciso ter cautela ao utilizá-lo, pois o objeto sobre o qual o método está agindo pode ser modificado. O escopo das variáveis é definido utilizando prefixos: variáveis globais tem o prefixo “\$”, enquanto variáveis de uma instância utilizam “@” e as variáveis da classe, “@@” (Flanagan e Matsumoto, 2008);
- Classes e módulos: Uma classe é uma coleção de métodos relacionados que operam no estado de um objeto. O estado de um objeto é mantido pelas variáveis de uma instância (Flanagan e Matsumoto, 2008);
- Strings: Em Ruby, as string são mutáveis, o operador “[]=” permite a alteração de caracteres de uma string, ou inserir, deletar e substituir substrings. A classe String define diversos outros métodos que permitem alterar uma string. Devido a esta mutabilidade, as strings em um programa não são objetos únicos. Se uma string for incluída em um loop, um novo objeto é criado a cada iteração do loop (Flanagan e Matsumoto, 2008).

Para gerenciar as bibliotecas utilizadas na implementação da ferramenta, foi utilizado o RubyGems, que é um sistema de gerenciamento de pacotes para projetos Ruby. As bibliotecas distribuídas pelo RubyGems são conhecidos como *gems*. Este gerenciador facilita a instalação do Ruby e gerencia automaticamente complexas dependências entre as bibliotecas (Flanagan e Matsumoto, 2008).

3.3.2 Framework web

Desde seu surgimento em 2004, Ruby On Rails rapidamente se tornou um dos mais poderosos e populares *frameworks* para a construção de aplicações web dinâmicas. É totalmente open-source e gratuito (Hartl, 2010).

Rails (<http://rubyonrails.org/>) oferece um *design* elegante e compacto e explorando a maleabilidade da linguagem Ruby, cria uma linguagem de domínio específico para desenvolver aplicações web. Como resultado, muitas tarefas comuns do desenvolvimento web - como gerar HTML, elaborar modelos de dados e rotas de URL - são facilitadas pelo Rails, tornando o código da aplicação conciso e compreensível (Hartl, 2010).

Segundo Hartl (2010), outra vantagem deste *framework* é sua rápida adaptação a novidades na tecnologia web. Um exemplo disso é que Rails foi um dos primeiros frameworks a implementar o modelo de arquitetura REST (*Representational State Transfer*), que entre outras vantagens, facilita a integração entre aplicações utilizando o padrão web para troca de dados.

Rails segue o padrão de arquitetura *model-view-controller* (MVC), que traz uma separação entre o “domínio lógico” (também conhecido como “regras de negócio”) da entrada de dados e a apresentação lógica associada a uma interface gráfica para o usuário (também conhecida como GUI – *Graphical User Interface*). No caso de aplicações web, o “domínio lógico” consiste em modelo de dados e a interface gráfica é apenas uma página web em um *browser* (Hartl, 2010).

A partir de uma interação com a aplicação Rails, o *browser* envia uma requisição, que é recebida por um servidor web e repassada a um *controller* do Rails, que se encarrega de prosseguir com a ação. Em alguns casos, o *controller* vai imediatamente renderizar uma *view*, um template que é convertido para HTML e enviado de volta ao *browser*. Em sites dinâmicos, geralmente o *controller* interage com um *model*, que é um objeto Ruby que representa um elemento do site e está encarregado da comunicação com o banco de dados. Após invocar o *model*, o *controller* então renderiza a *view* e retorna a página web para o browser como HTML (Hartl, 2010). A Figura 6 traz uma representação esquemática da arquitetura.

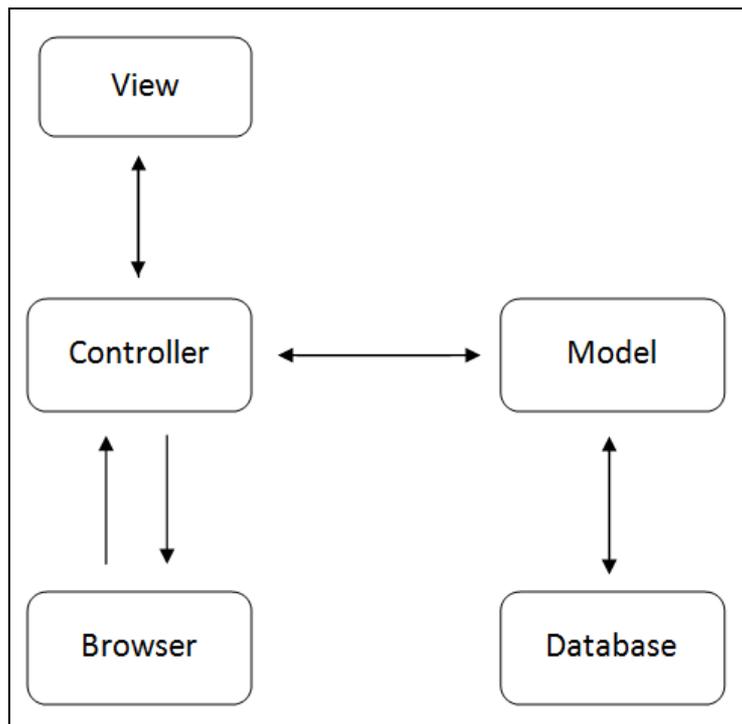


Figura 6 - Representação esquemática da arquitetura MVC.

Fonte: Adaptado de Hartl, 2010

Diante dos benefícios apresentados, optou-se pelo uso deste framework na implementação da ferramenta apresentada neste trabalho.

3.3.3 Framework *front-end*

Bootstrap (<http://getbootstrap.com/>) é um *framework* de código aberto para a construção de interfaces web utilizando design responsivo. Utiliza JavaScript (incluindo JQuery), CSS e HTML, além de suportar CSS3 e HTML5. *Bootstrap* apresenta quatro categorias de funcionalidades: *scaffolding*, componentes, plugins JavaScript e CSS base. *Scaffolding* é construído com base em um grid responsivo de 12 colunas, *layouts* e componentes. Os componentes incluem estilos básicos para funcionalidades comuns de interface de usuário. A base CSS inclui estilos para elementos HTML básicos e um conjunto de ícones chamados *Glyphicons* (Patton, 2013).

Web Design Responsivo (ou *Responsive Web Design*) consiste em apresentar a informação de forma acessível e confortável para diversos meios de acesso. Para auxiliar na detecção das características e resolução dos diversos aparelhos pelos quais uma informação pode ser acessada, são utilizadas as *Media Queries* (Eis, 2011). *Media*

Queries é a utilização de *Media Types* - que definem para qual tipo de media o CSS será direcionado - com uma ou mais expressões envolvendo características de uma media para definir formatações para diversos dispositivos. O browser (ou a aplicação) lê as expressões definidas na query, e caso o dispositivo se encaixe nestas requisições, o CSS será aplicado (Eis, 2009).

Outras características do Web Design Responsivo são *layouts* e imagens fluídas e menus adaptáveis. Estas características são estratégias de adaptação aos diferentes tamanhos da janela do navegador, de modo que os elementos possam se expandir ou contrair de acordo com as variações de tamanho. O *layout* fluído consiste na definição de larguras percentuais dos componentes com base no tamanho da tela do navegador. Imagens fluídas sugere a redução ou alteração das imagens por outras menores de modo que se adaptem ao navegador. Menus adaptáveis podem utilizar alternativas como a transformação do menu em um *selectbox* (ou *combobox*), retirada de alguns itens que podem não ser tão relevantes ou reformatação de design (Eis, 2011).

3.3.4 Controle de versão

Controle de versão é um sistema que grava as mudanças de um arquivo ou conjunto de arquivos ao longo do tempo, permitindo a recuperação de versões específicas. Um sistema de controle de versão permite, entre outras funcionalidades: reverter arquivos ou o projeto de volta a um estado anterior, rever mudanças realizadas com o tempo e ver o responsável pela última alteração que pode estar ocasionando algum problema (Chacon, 2009).

O Git (<http://www.git-scm.com/>) é um sistema de controle de versão e desde sua origem, em 2005, tem evoluído e amadurecendo no sentido de facilitar o uso e ainda assim, manter as qualidades iniciais. A principal diferença entre o Git e os outros sistemas de controle de versão é a maneira como tratam os dados. Conceitualmente, a maioria dos outros sistemas armazena informação como uma lista composta por arquivos e mudanças feitas nestes arquivos ao longo do tempo. O Git, a cada submissão de código ou alteração no estado de um projeto armazenado, registra uma “fotografia” do estado de todos os arquivos e armazena uma referência. Para ser eficiente, se os arquivos não foram alterados, o Git não armazena o arquivo novamente, apenas relacionada a uma versão anterior idêntica que já está armazenada (Chacon, 2009).

O Bitbucket (<https://bitbucket.org/>) é um site de hospedagem para sistemas de controle de arquivos e foi utilizado, junto ao Git, para armazenar o código gerado pela implementação desta ferramenta.

3.3.5 Testes

Para o desenvolvimento de testes da ferramenta foram utilizadas a técnica TDD e as ferramentas RSpec e Coverage que serão apresentadas a seguir.

O TDD (*test-driven development* ou desenvolvimento orientado a testes) é uma abordagem na qual o teste é escrito antes do código da aplicação e tem como objetivo garantir que a funcionalidade adicionada está sendo testada. Nesta abordagem, primeiro escreve-se um teste que irá falhar, para depois corrigi-lo e fazê-lo passar. Uma maneira de proceder com o TDD é um ciclo conhecido como “Vermelho, Verde, Refatorar”. O primeiro passo se refere ao teste falho, enquanto o segundo passo se refere ao teste passar. Uma vez que o teste foi passado, pode-se refatorar o código, eliminando duplicações, se necessário, sem alterar a função (Hartl, 2010).

O RSpec (<http://rspec.info/>) é um *framework* de testes escrito em Ruby, que permite que a descrição da aplicação em uma DSL (*Domain Specific Language*) simples e elegante. Possui suporte nativo para o Ruby on Rails através da extensão *rspec-rails* (Vieira, 2011).

Segundo Vieira (2011), o RSpec possui uma nomenclatura específica quando se refere às asserções: *expectativas*. Quando exemplos são descritos, são definidas *expectativas* de como o código deve se comportar. Esta extensão permite criar *expectativas* em diversas partes do Ruby on Rails como modelos, *controllers*, *views* e *helpers*, de forma isolada.

O RSpec permite criar *mocks*, objetos que simulam o comportamento de outros objetos e que podem ter seu comportamento alterado de forma controlada. Normalmente, são usados em alguns casos específicos, como quando suas *expectativas* dependem de objetos complexos, quando um objeto não fornece resultados determinísticos ou quando um objeto possui estados difíceis de reproduzir (Vieira, 2011).

Para analisar a cobertura de código, ou seja, a proporção de código que está coberta pelos testes, foi utilizada a biblioteca simplecov (<https://github.com/colszowka/simplecov>). A cobertura do código da ferramenta Histimate indicada pela simplecov foi de 98%.

3.3.6 Banco de dados

PostgreSQL (<http://www.postgresql.org/>) é um banco de dados relacional que teve sua origem na Universidade da Califórnia, no fim da década de 70. Nesta época, foi iniciado o desenvolvimento do banco de dados relacional Ingres, antecessor do PostgreSQL. Em 1986, o Ingres ganhou funcionalidades de orientação a objeto e esta nova versão recebeu o nome de Postgres. Em 1996, o nome foi novamente alterado, para PostgreSQL, que trouxe novas funcionalidades como melhorias na performance e controle de concorrência multiversão (Douglas e Douglas, 2005).

O PostgreSQL é um produto *open source*, ou seja, a distribuição da ferramenta e de seu código fonte é livre e acessível a todos os interessados. Esta ferramenta é um dos mais avançados servidores de banco de dados e suas principais funcionalidades são (Douglas e Douglas, 2005):

- Objeto-relacional: Toda tabela define uma classe. PostgreSQL implementa herança entre as tabelas (classes) e funções e operadores são polimórficos.
- Compatibilidade com normas: A sintaxe deste banco implementa boa parte das funcionalidades estabelecidas nas revisões SQL92 e SQL99 da linguagem SQL. As diferenças de sintaxe são relacionadas a funcionalidades que são exclusivas do PostgreSQL.
- Código aberto: Um time de desenvolvedores internacionais mantém o PostgreSQL. Isto garante que o banco possa ser utilizado em várias linguagens, e não apenas em inglês.
- Processamento de transação: O modelo de transação utilizado é baseado no controle de concorrência multiversão (MVCC), que oferece uma melhor performance que outros controles que coordenam múltiplos usuários com o bloqueio a nível de tabelas, páginas ou registros.

- Integridade referencial: Este banco implementa uma completa integridade referencial, suportando relacionamentos de chaves estrangeiras e primárias e *triggers*.
- Múltiplas linguagens procedurais: *Triggers* e outros procedimentos podem ser escritos em diversas linguagens procedurais. A maior parte dos códigos *server-side* é escrita em PL/pgSQL, uma linguagem procedural semelhante ao Oracle PL/SQL, mas também é possível o desenvolvimento em Tcl, Perl e até bash, o interpretador de comandos Linux/Unix.
- Múltiplas APIs: PostgreSQL suporta o desenvolvimento de aplicações clientes em diversas linguagens, como C, C++, ODBC, Perl, PHP e Python.
- Tipos de dados únicos: Uma variedade de tipos de dados está disponível no PostgreSQL. Além dos tipos usuais, como numérico e texto, também são disponibilizados tipos geométricos, booleanos e tipos de dados específicos para endereços de rede.
- Extensibilidade: É uma das mais importantes funcionalidades. É possível implementar novas funcionalidades no PostgreSQL, como novos tipos de dados, funções e operadores e até novas linguagens procedurais e clientes.

Ruby apresenta a interface Pg que permite a conexão com o PostgreSQL, e que foi utilizada na implementação desta ferramenta.

3.4 Funcionalidades

Esta seção descreve funcionalidades e apresenta algumas telas da ferramenta.

3.4.1 Acesso ao sistema

Ao acessar a ferramenta, é apresentada a tela de login (Figura 7). Se o usuário já for cadastrado, basta fornecer o login e senha e clicar em “Login” para acessar a ferramenta. Caso contrário, é possível ser direcionado a tela de cadastro (Figura 8), clicando na opção “Cadastrar”. Esta tela também apresenta a opção “Esqueci minha

senha”, para que usuários cadastrados possam recuperar suas senhas em caso de esquecimento.

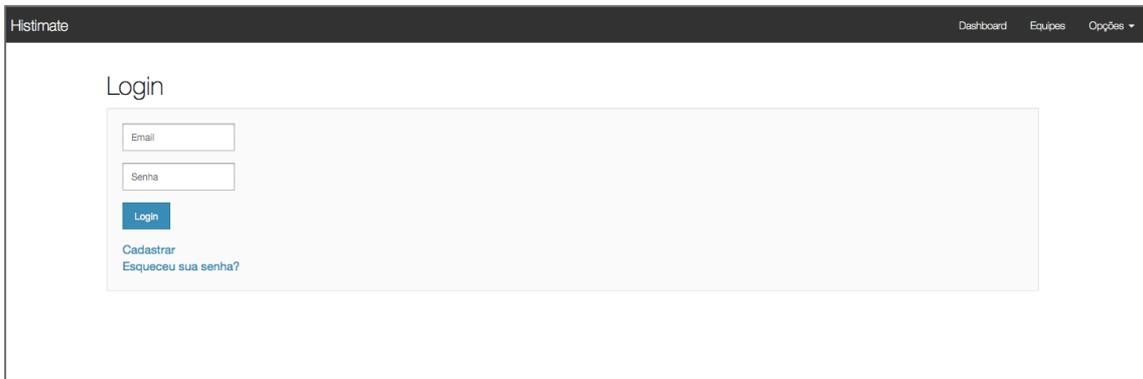


Figura 7 - Tela de login.

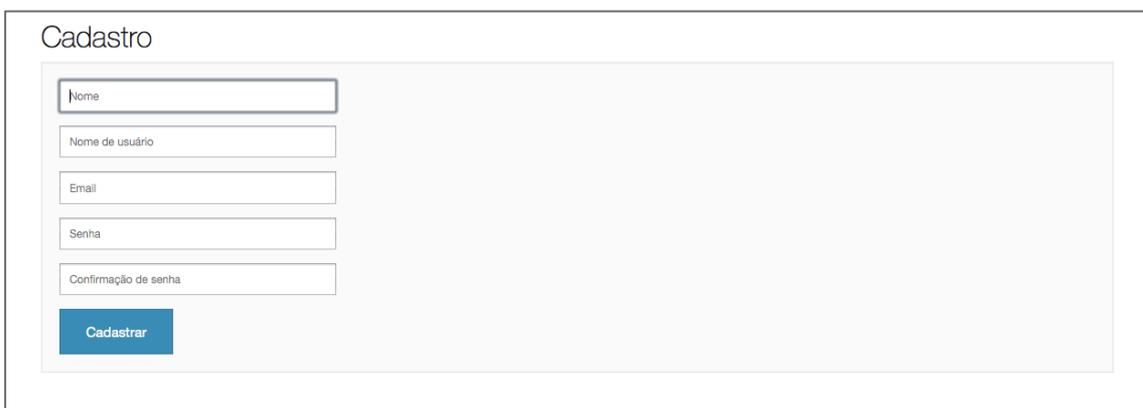


Figura 8 - Tela de cadastro.

Uma vez logado, o usuário pode editar seu cadastro (Figura 9), acessando o menu superior (Nome do usuário > Editar cadastro).

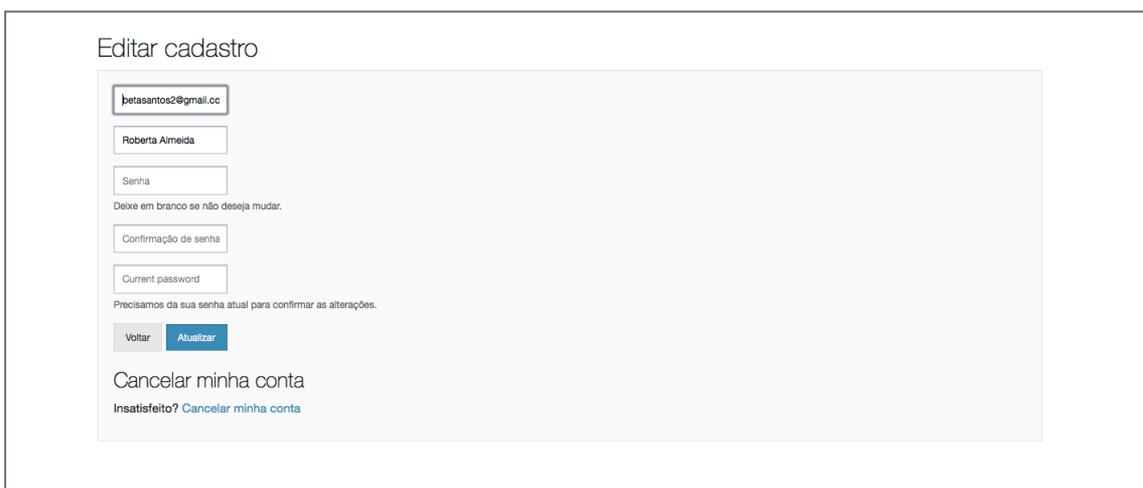


Figura 9 - Tela de edição de cadastro.

3.4.2 Criação do projeto

Após o login, o usuário é redirecionado a tela de *Dashboard* (Figura 10) que apresenta os projetos já cadastrados e a opção para cadastrar um novo projeto. Ao clicar na opção de criar um novo projeto, uma tela (Figura 11) é apresentada ao usuário, com as informações nome e tamanho do *sprint*, além das opções: selecionar uma equipe já cadastrada ou cadastrar uma nova equipe que será alocada no projeto, cadastrar os status das tarefas do projeto que serão expostos no *board* e cadastrar as estimativas que serão utilizadas para histórias e tarefas.

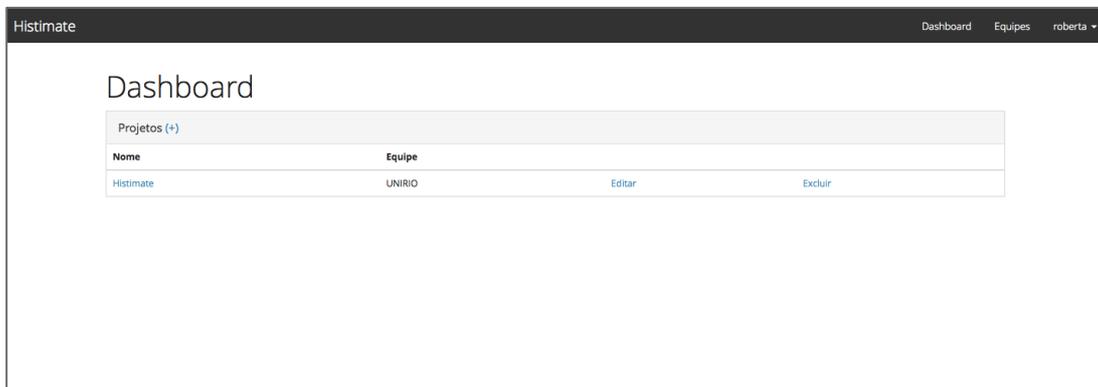


Figura 10 - Tela de *Dashboard*.

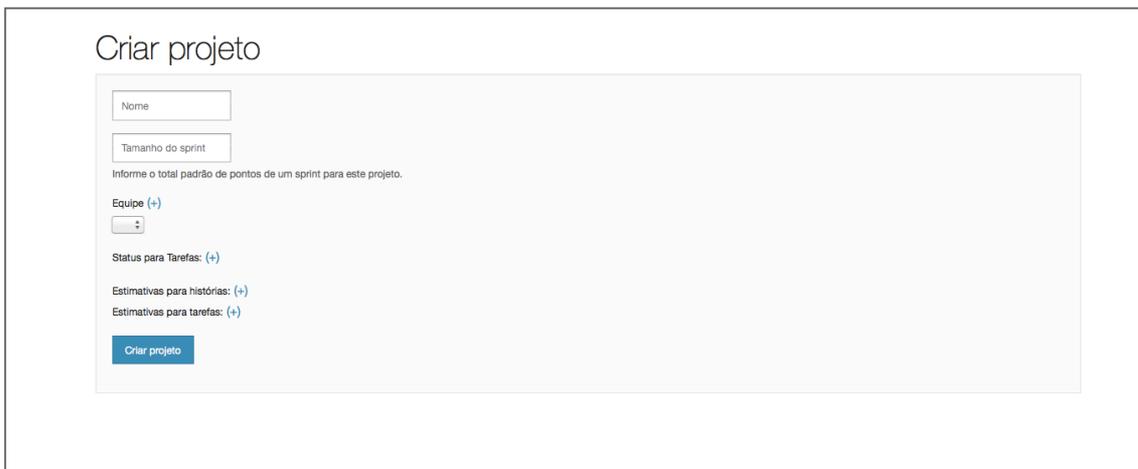


Figura 11 - Tela de criação de projeto.

Ao clicar na opção “Criar nova equipe”, uma janela modal é exibida (Figura 12) com um campo para o nome da equipe e a listagem de usuários já cadastrados no

sistema. Para escolher os que irão compor a equipe, é preciso selecioná-los. Após salvar estas informações, a janela modal é fechada e o usuário retorna à tela de cadastro do projeto, que irá apresentar a equipe criada já selecionada.

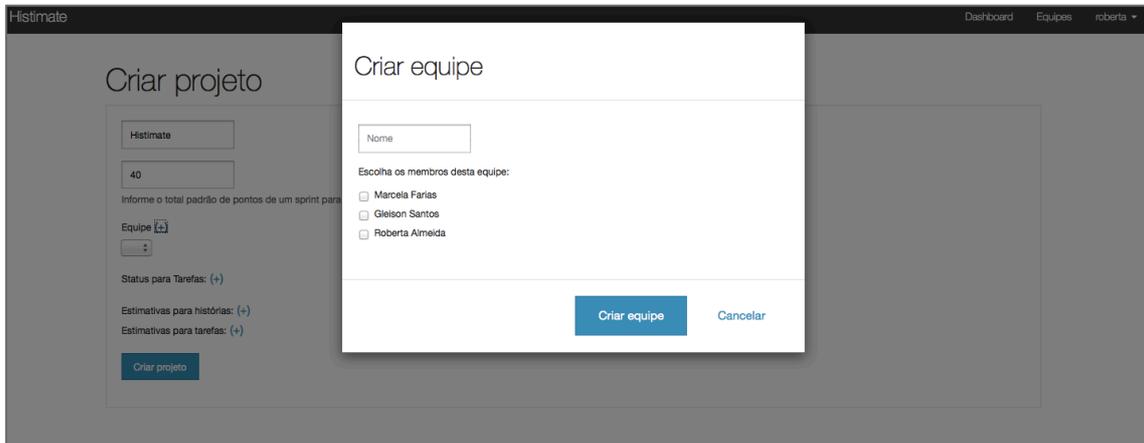


Figura 12 - Janela modal de cadastro de equipe.

Ao clicar na opção “Adicionar novos status”, uma janela modal é exibida (Figura 13), apresentando os campos nome e ordem, que irá definir a posição em que o status será exibido no *board* de tarefas. É possível criar quantos status forem necessários. Ao salvar as informações, a janela modal é fechada e o usuário retorna à tela de cadastro do projeto.

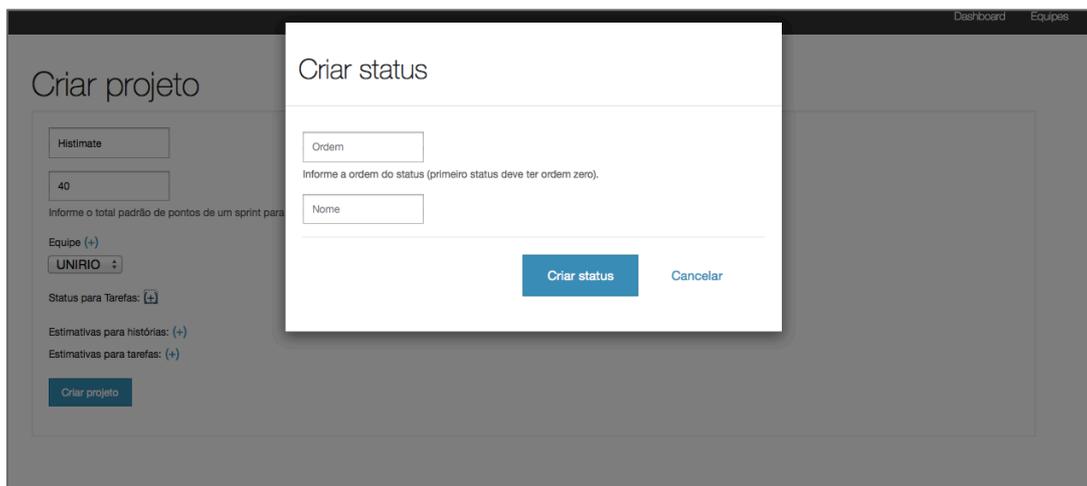


Figura 13 - Janela modal de criação de status.

Ao clicar na opção “Adicionar novas estimativas para história”, uma janela modal é exibida (Figura 14) com o tipo de estimativa - tamanho - selecionado e um campo para o valor da estimativa. Ao salvar as informações, a janela modal é fechada e o usuário retorna à tela de cadastro do projeto.

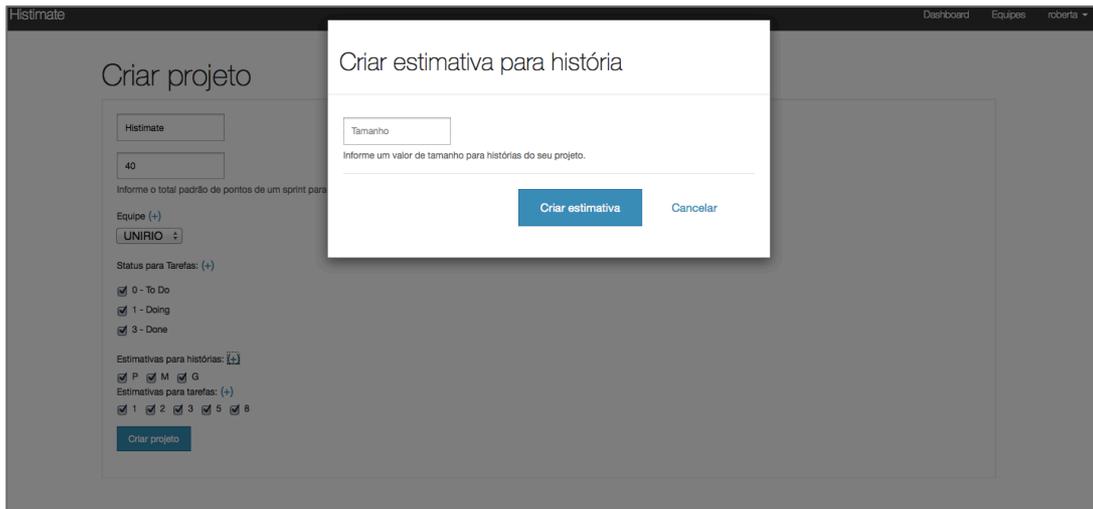


Figura 14 - Janela modal de criação de estimativas de histórias.

Ao clicar na opção “Adicionar novas estimativas para tarefas”, uma janela modal é exibida (Figura 15) com o tipo de estimativa - esforço - selecionado e um campo para o valor da estimativa de esforço. Ao salvar as informações, a janela modal é fechada e o usuário retorna à tela de cadastro do projeto (Figura 16).

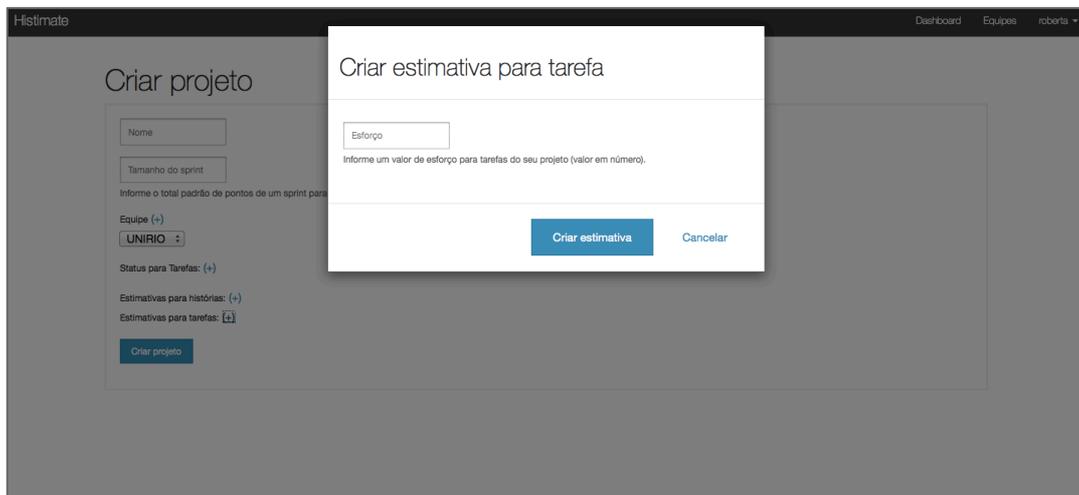


Figura 15 - Janela modal de criação de estimativa de tarefa.

Criar projeto

Histimate

40

Informe o total padrão de pontos de um sprint para este projeto.

Equipe (+)

UNIRIO

Status para Tarefas: (+)

0 - To Do
 1 - Doing
 3 - Done

Estimativas para histórias: (+)

P M G

Estimativas para tarefas: (+)

1 2 3 5 8

[Criar projeto](#)

Figura 16 - Tela de cadastro do projeto com informações preenchidas.

Após salvar as informações e confirmar a criação do projeto, o usuário é redirecionado para uma tela (Figuras 17 e 18) que apresenta o *backlog* do projeto, trazendo as histórias já cadastradas para aquele projeto, com as opções de editar e excluir uma história. As opções para adicionar histórias no *backlog* e ver *sprints* do projeto também são apresentadas nesta tela.

Projeto: Histimate

Equipe do projeto: UNIRIO [Adicionar história ao backlog](#)

Backlog		
Nome	Descrição	Sprint
Como usuário, quero me cadastrar no sistema.	Informações do cadastro: nome, nome de usuário, email e senha.	Editar Excluir
Como usuário, quero logar no sistema.	Informações para login: email e senha.	Editar Excluir
Como usuário, quero cadastrar um novo projeto e suas respectivas informações.	Informações de cadastro: nome, tamanho do sprint, equipe, status para tarefas, estimativas para tarefas e histórias.	Editar Excluir
Como usuário, quero cadastrar as estimativas de histórias e tarefas do projeto.		Editar Excluir
Como usuário, quero cadastrar os status possíveis das tarefas do projeto.		Editar Excluir
Como usuário, quero cadastrar equipes.		Editar Excluir
Como usuário, quero definir a equipe de um projeto.		Editar Excluir
Como usuário, quero cadastrar uma história definindo seu nome, descrição e tamanho.	As opções de tamanho disponíveis são as cadastradas no projeto previamente.	Editar Excluir
Como usuário, quero cadastrar informações de um sprint.		Editar Excluir
Como usuário, quero selecionar as histórias que farão parte do sprint.		Editar Excluir
Como usuário, quero escrever tarefas relacionadas a cada história selecionada para o sprint e estimar o esforço de cada uma.		Editar Excluir

Figura 17 - Tela de *backlog* do projeto.

Histimate		Dashboard	Equipes	roberta
Como usuário, quero me cadastrar no sistema.	Informações do cadastro: nome, nome de usuário, email e senha.	Editar	Excluir	
Como usuário, quero logar no sistema.	Informações para login: email e senha.	Editar	Excluir	
Como usuário, quero cadastrar um novo projeto e suas respectivas informações.	Informações de cadastro: nome, tamanho do sprint, equipe, status para tarefas, estimativas para tarefas e histórias.	Editar	Excluir	
Como usuário, quero cadastrar as estimativas de histórias e tarefas do projeto.		Editar	Excluir	
Como usuário, quero cadastrar os status possíveis das tarefas do projeto.		Editar	Excluir	
Como usuário, quero cadastrar equipes.		Editar	Excluir	
Como usuário, quero definir a equipe de um projeto.		Editar	Excluir	
Como usuário, quero cadastrar uma história definindo seu nome, descrição e tamanho.	As opções de tamanho disponíveis são as cadastradas no projeto previamente.	Editar	Excluir	
Como usuário, quero cadastrar informações de um sprint.		Editar	Excluir	
Como usuário, quero selecionar as histórias que farão parte do sprint.		Editar	Excluir	
Como usuário, quero escrever tarefas relacionadas a cada história selecionada para o sprint e estimar o esforço de cada uma.		Editar	Excluir	
Como usuário, quero mudar status da tarefa para "em desenvolvimento" e me tornar responsável por sua execução.		Editar	Excluir	
Como usuário, quero cadastrar o esforço real da tarefa que implementei.		Editar	Excluir	
Como usuário, quero visualizar o board com os status cadastrados para o projeto e as tarefas de acordo com seu status.		Editar	Excluir	
Como usuário, quero iniciar um sprint.		Editar	Excluir	

Ver sprints

Figura 18 – Continuação da tela de *backlog* do projeto.

Se a opção “Adicionar histórias ao *backlog*” for clicada, uma janela modal é apresentada (Figura 19) com os campos nome, descrição e o status da história no momento de sua criação é “Pendente” por padrão. As opções de status da história são: “Pendente”, “Em andamento”, “Concluída” e estas opções são alteradas de acordo com a situação da história no fluxo do projeto. Ao finalizar a criação da história, a janela modal é fechada e o usuário retorna a tela que *backlog* que já apresentará a história que acabou de ser criada.

Figura 19 - Janela modal de criação de história.

Ao clicar na opção “Visualizar sprints” na tela de *backlog* do projeto, uma nova tela (Figura 20) é apresentada, com a listagem de *sprints* cadastrados para o projeto, com as opções editar e excluir para cada *sprint* cadastrado. Um link para criar um novo *sprint* também é disponibilizado nesta tela.

Controlar sprints

Projeto: Histimate Criar novo sprint

Sprints				
#	Data de início	Data fim	Status	Total de pontos estimados
1	04 Dec 00:00	04 Dec 00:00	Concluído	31
2			Em planejamento	0

Figura 20 - Tela de visualização de *sprints*.

Ao clicar em “Criar novo *sprint*”, uma nova tela (Figura 21) é apresentada com o campo para inserir o número do *sprint*. No momento da criação, o status é “Em Planejamento” por padrão. As opções de status do *sprint* são: “Em planejamento”, “Em detalhamento”, “Em execução” e “Concluído” e estas opções são alteradas de acordo com a situação do *sprint* no fluxo do projeto.

Criar sprint para projeto Histimate

Número do sprint

Voltar
Criar sprint

Figura 21 - Tela de criação do *sprint*.

Ao clicar em um número de *sprint* na tela de visualizações, é exibida uma tela com as informações do status atual do *sprint*.

3.4.3 Planejamento e detalhamento do *sprint*

Quando o *sprint* está em planejamento, a tela que pode ser observada na Figura 22 é apresentada. É possível selecionar histórias que ainda não estejam associadas a outros *sprints* para compor o *sprint* que está sendo planejado. É possível também atribuir uma estimativa a cada história através da opção “Estimar”, que ao ser clicada, apresenta uma janela modal (Figura 23) com as opções de estimativas de história cadastradas previamente no projeto. Após a seleção das histórias que irão compor o *sprint*, o planejamento pode ser encerrado ao clicar em “Finalizar planejamento”.

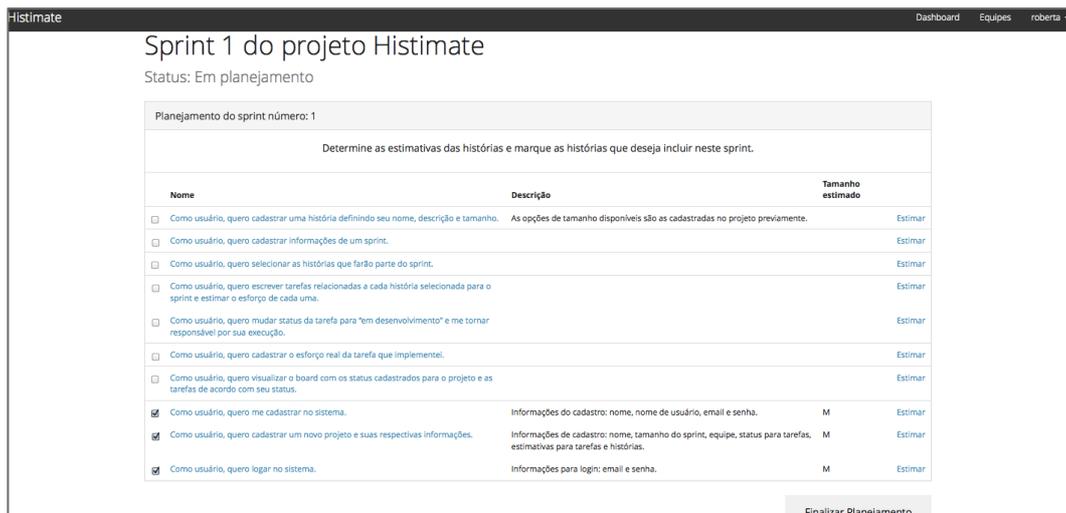


Figura 22 - Tela de *sprint* com status em planejamento.

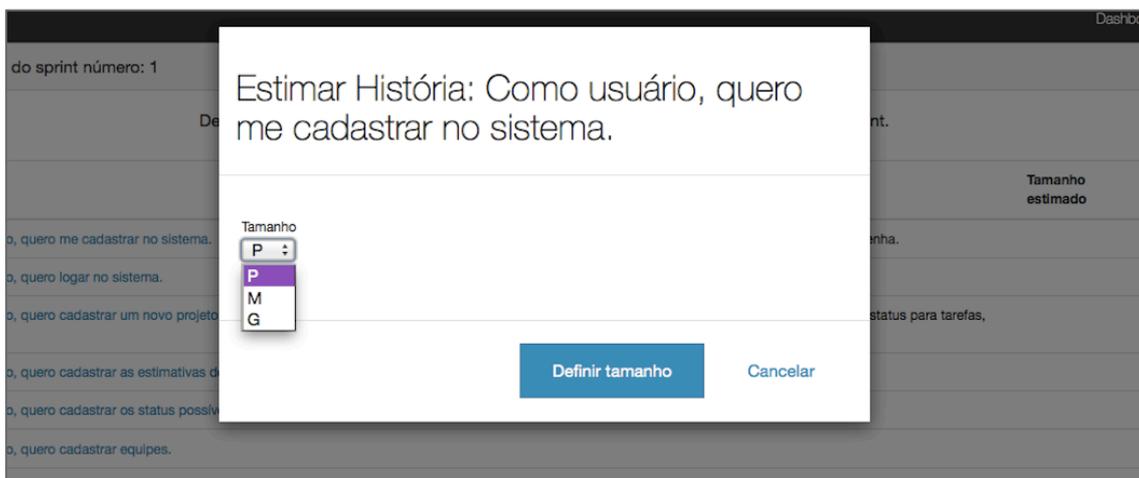


Figura 23 - Janela modal de estimativa de história.

Após a finalização do planejamento, o status do *sprint* passa para “Em detalhamento” (Figura 24). Nesta fase é possível adicionar tarefas e verificar estimativas para cada história adicionada na etapa anterior.

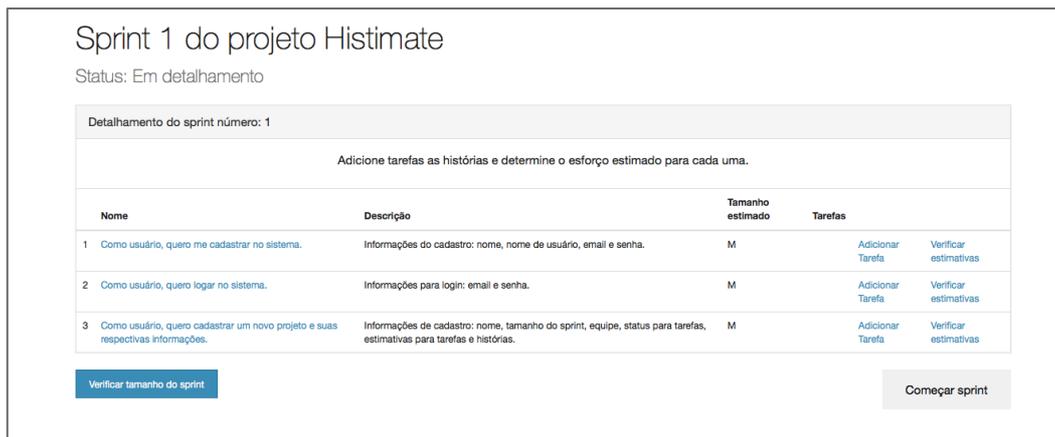


Figura 24 - Tela de *sprint* com status em detalhamento.

Ao clicar em “Adicionar tarefa”, uma janela modal é apresentada (Figura 25) com os campos nome, descrição e uma lista de opções com as estimativas de esforço definidas no início do projeto. Ao confirmar a criação da tarefa, a janela modal é fechada e o usuário retorna à tela de detalhamento do *sprint*.

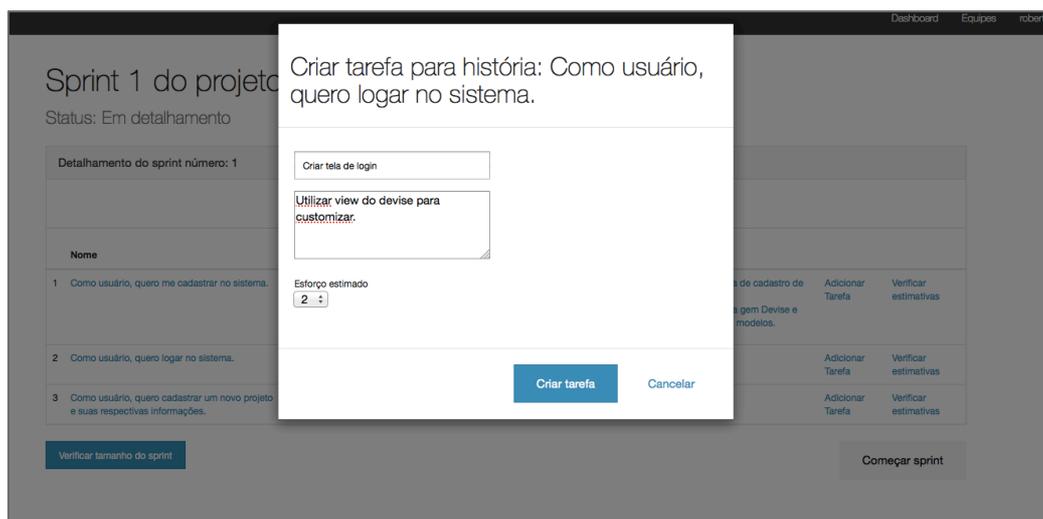


Figura 25 - Janela modal de criação de tarefas.

Ao clicar em “Verificar estimativas” de uma história, o sistema verifica na base de dados se existem histórias de mesmo tamanho da que está sendo verificada. Caso a base ainda não tenha histórias cadastradas com o mesmo tamanho da história que está sendo verificada, a mensagem exibida na Figura 26 é exibida.

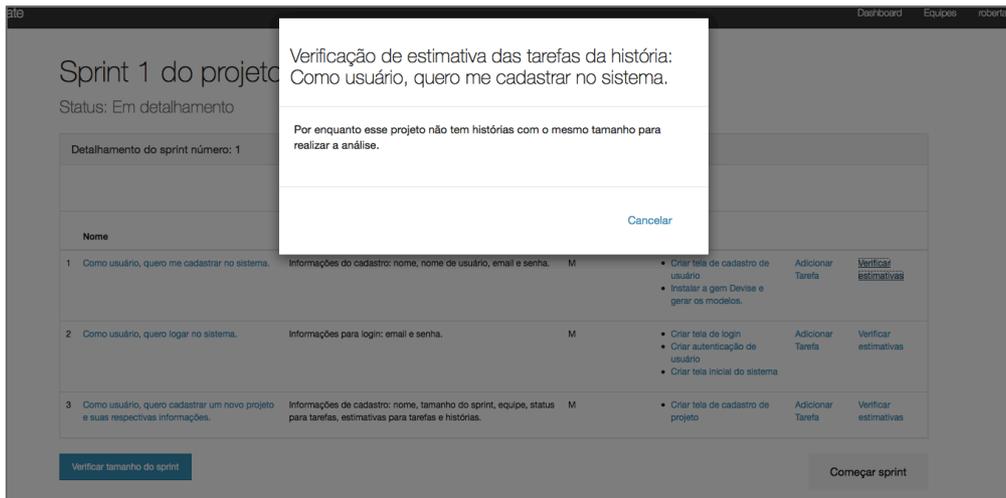


Figura 26 - Mensagem informativa de ausência de base histórica de histórias com mesmo tamanho.

Caso existam histórias com o mesmo tamanho, o sistema calcula a média de esforço realizado das tarefas destas histórias e a compara a soma dos esforços estimados das tarefas da história que está sendo verificada, conforme Figura 27.

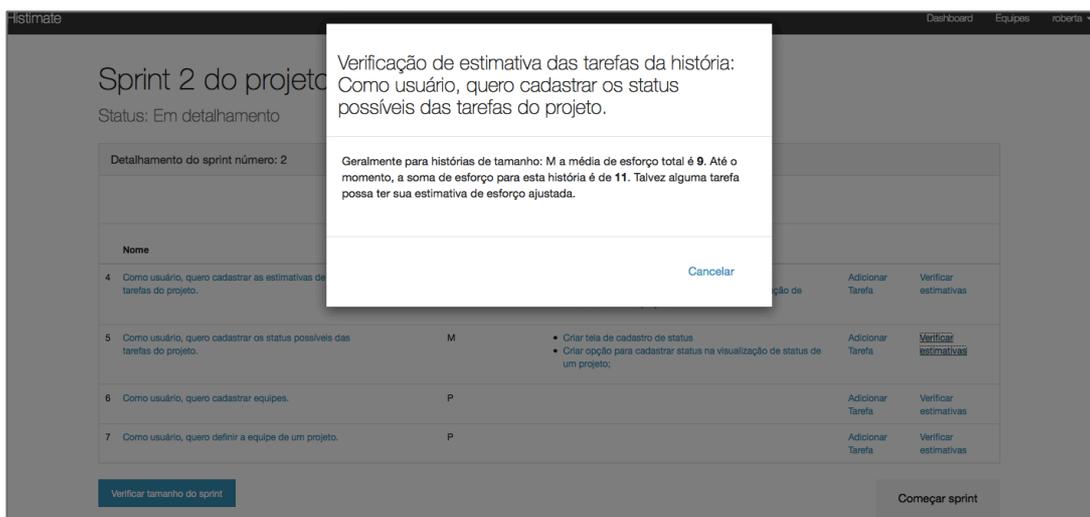


Figura 27 - Mensagem com resultado da análise das estimativas das tarefas de uma história.

Outra opção presente na tela de detalhamento de *sprint* é “Verificar o tamanho do *sprint*” que verifica se a soma dos esforços estimados está de acordo com o tamanho padrão do *sprint* definido no projeto. Ao clicar nesta opção, uma janela modal (Figura 28) com o resultado da análise é apresentada.

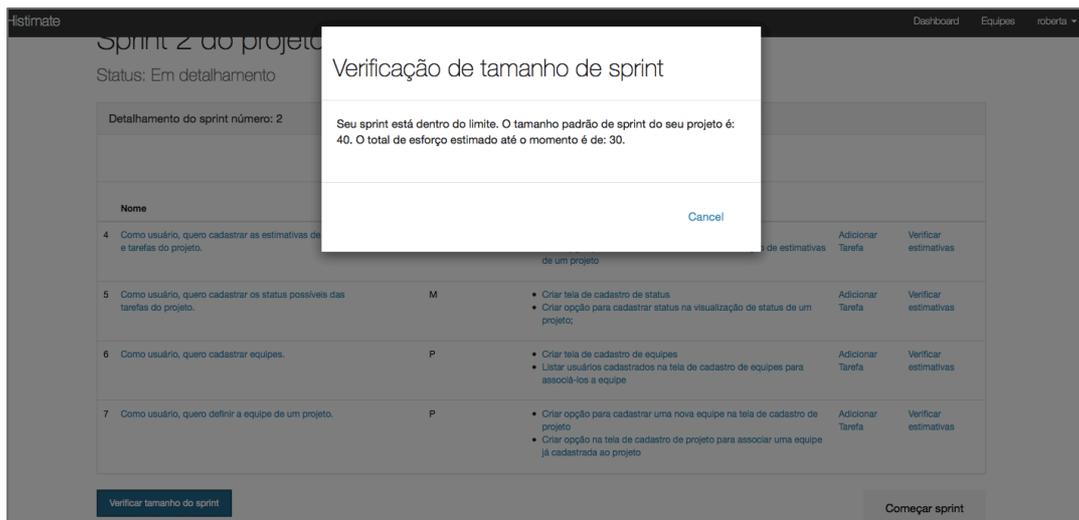


Figura 28 - Mensagem de resultado da análise de verificação do tamanho do *sprint*.

Após o cadastro e estimativa de esforço de todas as tarefas, o *sprint* pode ser iniciado através do clique no botão “Começar *sprint*”. Neste momento, os status das histórias são alterados para “Em andamento”.

3.4.4 Execução do *sprint*

Com a mudança de status do *sprint* para “Em execução” a tela de visualização do *sprint* também é alterada e passa a ser um quadro (*board*) dividido com os status cadastrados para o projeto e com as tarefas inicialmente dispostas no status de ordem 0 (Figura 29).

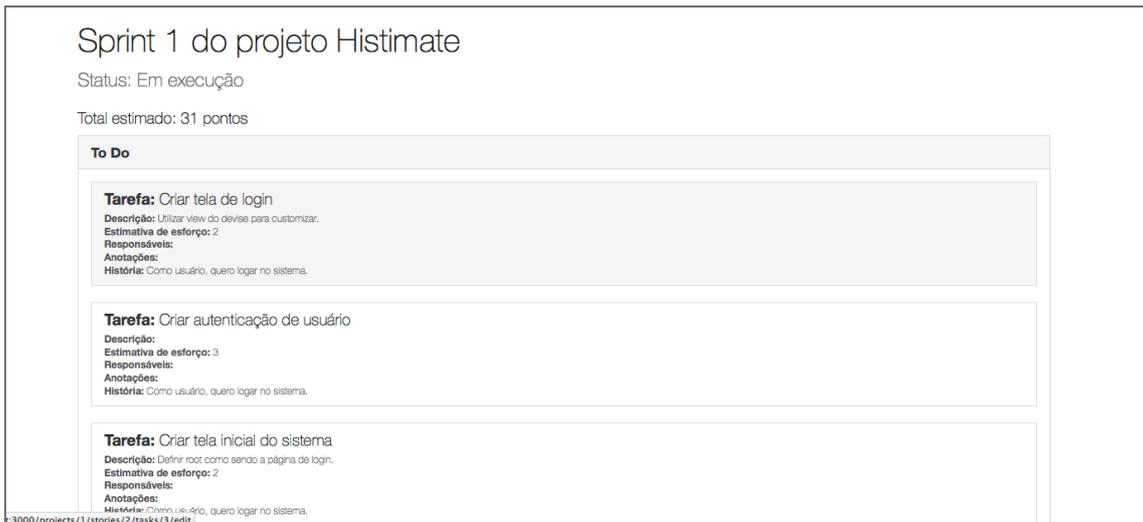


Figura 29 - Tela de *sprint* com status em execução.

Ao clicar em cada tarefa apresentada no *board*, uma janela modal (Figura 30) é exibida e traz a possibilidade de edição do nome, descrição e observações da tarefa. Além disso, é possível alterar o status da tarefa, fazendo com que esta mude de coluna na disposição do *board* e atribuir um ou mais responsáveis por esta tarefa.

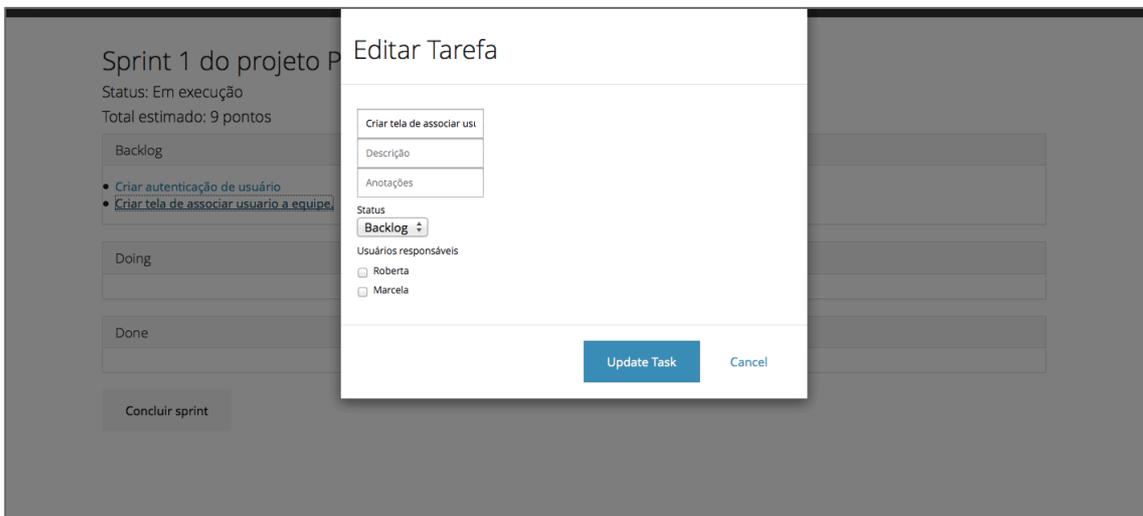


Figura 30 - Janela modal de edição de tarefa.

Quando a tarefa estiver no último status do *board* (Figura 31) e for aberta para edição (Figura 32), mais um campo será exibido, para que seja inserido o registro do esforço realizado.

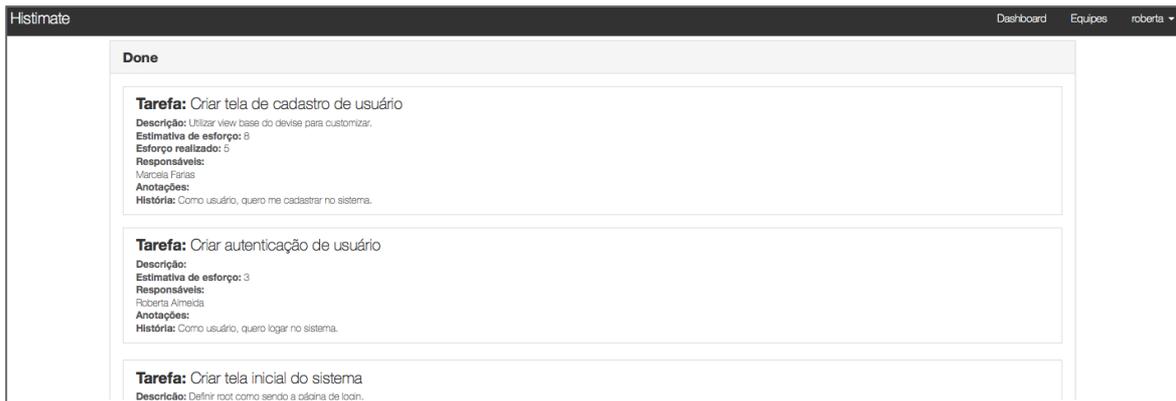


Figura 31 - Apresentação de tarefas no último status do *board*.

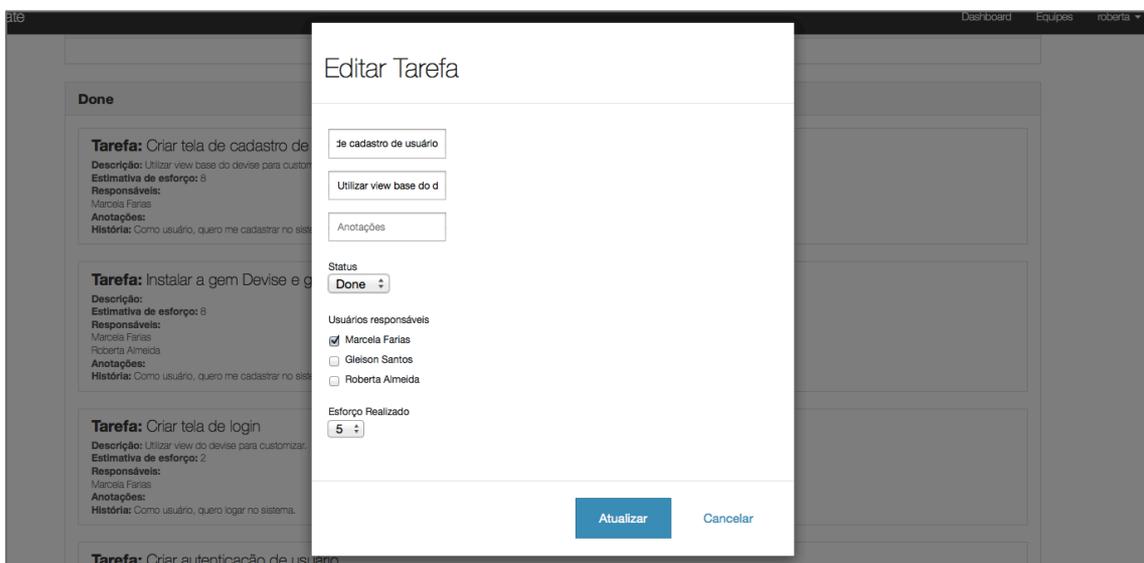


Figura 32 - Tela de edição de tarefa concluída.

3.4.5 Conclusão do *sprint*

Na tela de visualização de um *sprint* em execução, é apresentado um botão “Concluir *sprint*”. Após o clique neste botão, é apresentada uma janela modal (Figura 33) de confirmação da finalização do *sprint*. Caso a finalização seja confirmada, o status do *sprint* é alterado para “Concluído” e a tela de visualização (Figura 34) também é alterada, apresentando um resumo do *sprint*. Neste momento, apenas as histórias em que todas as tarefas foram concluídas terão o status alterado para concluída. Caso alguma tarefa da história não tenha sido finalizada, o status da história não será alterado.

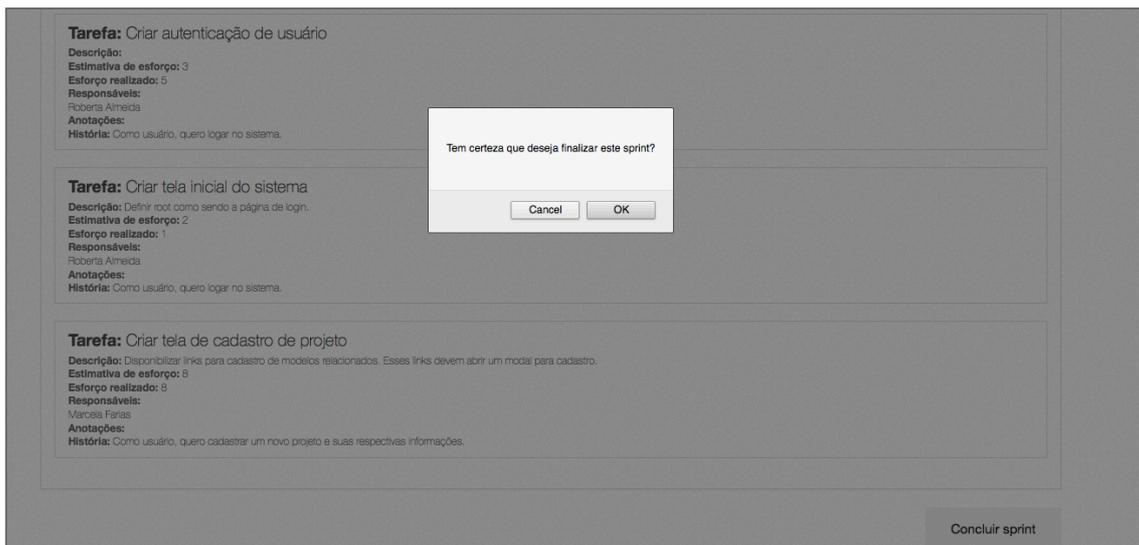


Figura 33 - Janela modal de confirmação da finalização do sprint.

Sprint 1 do projeto Histimate

Status: Concluído

Sprint 1 finalizado do projeto Histimate

Esforço estimado do sprint: 31
 Esforço realizado do sprint: 27

Nome	Descrição	Status	Tamanho estimado	Tarefas	Total de esforço estimado	Total de esforço realizado
1	Como usuário, quero me cadastrar no sistema.	Concluído	M	<ul style="list-style-type: none"> - Nome: Criar tela de cadastro de usuário - Descrição: Utilizar view base do devise para customizar. - Status: Done - Nome: Instalar a gem Devise e gerar os modelos. - Descrição: - Status: Done 	16	10
2	Como usuário, quero logar no sistema.	Concluído	M	<ul style="list-style-type: none"> - Nome: Criar tela de login - Descrição: Utilizar view do devise para customizar. - Status: Done 	7	9

Figura 34 - Tela de visualização de *sprint* com status concluído.

A cada ciclo iterativo, espera-se que os totais de esforço estimado e realizado apresentados na tela de *sprint* concluído (Figura 34) sejam cada vez mais próximos. Isso indicará o aprendizado da equipe, que poderá observar nestes dados se as estimativas definidas no planejamento estão de acordo com o esforço realizado ao fim do *sprint*. Desta forma, é possível observar que o estabelecimento da base histórica de estimativas efetivamente auxilia o aprendizado da equipe e conseqüentemente, leva a uma maior precisão das estimativas, como proposto no Capítulo 1.

3.5 Comparação com ferramentas similares

Existem ferramentas que auxiliam as organizações no gerenciamento de projetos ágeis e que possuem algumas características comuns à ferramenta proposta neste trabalho. Entre estas ferramentas, foram selecionadas as ferramentas: Scrumwise (<http://www.scrumwise.com/>), ScrumHalf (<http://www.myscrumhalf.com/pt/>), Kanbanery (<https://kanbanery.com/>), Trello (<https://trello.com/>) e JIRA (<https://www.atlassian.com/software/jira>), para uma breve comparação de funcionalidades, apresentada na Tabela 1.

Tabela 1 - Tabela comparativa de ferramentas similares

	Scrumwise	ScrumHalf	Kanbanery	Trello	JIRA	Histimate
Board de tarefas	Sim	Sim	Sim	Sim	Sim	Sim
Backlog	Sim	Sim	Sim	Sim	Sim	Sim
Customização de status e estimativas	Não	Não	Não	Parcial	Parcial	Sim
Apoio a estimativas	Não	Não	Não	Não	Não	Sim
Definição de tarefas e histórias	Sim	Sim	Sim	Não	Sim	Sim

Como pode ser observado, o diferencial da ferramenta Histimate é a possibilidade de customizar os status das tarefas, que permite sua utilização em projetos que adotam diferentes metodologias ágeis e com diferentes escalas de estimativas, e o apoio à definição de estimativas.

3.6 Considerações finais

Este capítulo apresentou uma breve introdução da ferramenta e as justificativas para algumas definições realizadas, como a estimativa em pontos das tarefas e uma base histórica única por projeto.

Foram apresentados também os elementos utilizados em sua especificação, as variáveis do fluxo de um projeto ágil que foram definidas na fase de planejamento deste trabalho, o modelo de classes e as histórias de usuário criadas para a implementação da ferramenta, bem como suas respectivas tarefas. As tecnologias utilizadas na implementação foram apresentadas em seguida, com definições sobre cada uma e suas respectivas vantagens. Além disso, foram expostas as funcionalidades, com descrições detalhadas do fluxo de utilização da ferramenta e a apresentação de telas relacionadas a cada fluxo descrito.

Por fim, uma comparação com ferramentas similares foi apresentada, buscando expor as principais diferenças entre a Histimate e outras ferramentas que também são utilizadas no apoio a metodologias ágeis. O diferencial da Histimate é possibilitar o apoio ao processo de estimativas e a customização de status e estimativas. Ao contrário da Histimate, as ferramentas similares apresentadas não apresentam a visão crítica em relação ao histórico de estimativas. Além disso, a ferramenta possibilita a definição de tarefas e histórias, a exposição de tarefas em um quadro (*board*) e a visualização do *backlog* de histórias do projeto.

4 Conclusão

4.1 Considerações finais

Este trabalho apresentou a ferramenta Histimate, que possibilita a construção de uma base histórica de estimativas. Para auxiliar o entendimento das necessidades envolvidas no contexto da ferramenta, foram apresentados os conceitos de Métodos Ágeis e o Manifesto Ágil, as três metodologias mais utilizadas: Scrum, Kanban e XP, além do uso de histórias para definição do escopo de um projeto e as definições de estimativas em projetos tradicionais e ágeis, buscando esclarecer as diferenças entre as duas abordagens. Além da possibilidade de definição de histórias e tarefas, apresentação das tarefas em um quadro (*board*) e do *backlog* do projeto, esta ferramenta tem como diferencial a possibilidade de customização de estimativas de histórias e tarefas e o apoio a estimativas.

4.2 Principais contribuições

Diante da dificuldade encontrada por muitas organizações em estimar as atividades necessárias para a implementação de um projeto, é importante que busquem mecanismos para incentivar o aprendizado da equipe, buscando diminuir inconsistências nas estimativas que acabam por impactar no cronograma do projeto e podem gerar custos adicionais.

Como principal contribuição deste trabalho, foi desenvolvida uma ferramenta que apoia o processo de estimativas em organizações que fazem uso de metodologias ágeis, criando as bases que irão apoiar os processos de estimativas de ciclos iterativos de um projeto.

Ao apoiar este processo, a ferramenta busca facilitar a identificação de estimativas imprecisas que podem impactar no cronograma de um projeto e acabar por gerar custos que não estavam previstos inicialmente. Outro benefício é o aprendizado da

equipe, pois o registro histórico de erros e acertos nas estimativas contribui para que as possíveis melhorias sejam identificadas com mais facilidade de modo que as estimativas definidas nas reuniões de planejamento estejam mais próximas da realidade a cada ciclo iterativo.

Além do objetivo principal citado acima, a ferramenta também pode auxiliar organizações que estejam iniciando a adoção de metodologias ágeis, guiando as equipes através das etapas do processo que são definidas na ferramenta.

As possibilidades de definição de status e estimativas fazem com que a ferramenta não seja específica a apenas uma metodologia ágil e permitem uma maior customização para o usuário adequar estas informações a realidade de sua organização.

4.3 Trabalhos futuros

Durante o desenvolvimento deste trabalho foram identificados diversos pontos de melhoria visando agregar mais valor ao usuário da ferramenta, com funcionalidades que apoiariam não só estimativas, mas também outros processos utilizados em métodos ágeis. Os pontos identificados são:

- Relatórios relacionados ao usuário: Busca acompanhar a contribuição de cada usuário para o projeto. Este relatório contabiliza as tarefas as quais um usuário está envolvido, gerando um percentual de participação deste usuário no projeto;
- Estatísticas do projeto: Disponibilização de informações consolidadas sobre o projeto, como informações sobre os ciclos iterativos e outros fatores envolvidos. Geração de gráficos e relatórios de acompanhamento.
- Usabilidade: Melhorias na interface com o usuário, como a possibilidade de arrastar e soltar tarefas no *board*.
- Informações relativas ao usuário: criação de tela principal com resumo de projetos, tarefas associadas, entre outras informações relativas ao usuário;
- Detalhamento de estimativas: Armazenar histórico das estimativas por história e de tarefas ao longo do *sprint*;
- Utilização de estatísticas: Estabelecer semelhanças entre as histórias utilizando análises estatísticas;

- Outras funcionalidades: Integração com outras aplicações, como sistemas controladores de versão, de modo a possibilitar a associação de *check ins* (integrações de código) com as histórias e tarefas cadastradas na ferramenta Histimate.

Referências Bibliográficas

- Steffen, J. B. (2012) “O que são essas tais de metodologias Ágeis?”, https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/mas_o_que_s_c3_a3o_essas_tais_de_metodologias__c3_a1geis?lang=en. Acessado em 03 de Abril de 2013.
- AgilCoop. (200-?) “Casos de sucesso: Descrições de casos reais de aplicação de métodos ágeis e XP nos quais o projeto foi considerado um sucesso”, http://ccsl.ime.usp.br/agilcoop/casos_de_sucesso. Acessado em 03 de Abril de 2013.
- Bardusco, D. (2008) “Scrum Na Globo.Com - Estudo de caso”. Apresentação feita no evento Scrum & CMMi em 23/05/08 no Recife, <http://www.slideshare.net/bardusco/scrum-na-globocom-estudo-de-caso>. Acessado em 03 de Abril de 2013.
- Rebelo, P. (2013) “Escalando o Agile na Spotify: exemplo de sucesso de Lean Startup, Scrum e Kanban”, <http://www.infoq.com/br/articles/spotify-escalando-agile>. Acessado em 03 de Abril de 2013.
- Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. e Thomas, D. (2001) “Manifesto for Agile Software Development”, <http://agilemanifesto.org>. Acessado em 31 de março de 2013.
- Fadel, A. C. e Silveira, H. M. (2010) “Metodologias ágeis no contexto de desenvolvimento de software: XP, Scrum e Lean”. Monografia do Curso de Mestrado FT-027 - Gestão de Projetos e Qualidade da Faculdade de Tecnologia – UNICAMP.

- Sommerville, Ian. (2011), Engenharia de Software, Pearson Education - BR, 9ª Edição.
- Schwaber, K. e Beedle, M. (2002), Agile Software Development with Scrum, Prentice Hall, 1ª edição.
- Schwaber, K. e Sutherland, J. (2013) “Official Scrum Guide”, <http://www.scrum.org/Scrum-Guides>. Acessado em 03 de Abril de 2013.
- Santos, R. (2011) “Kanban para Desenvolvimento de Software”, <http://www.slideshare.net/Ridlo/kanban-para-desenvolvimento-de-software>. Acessado em 03 de abril de 2013.
- Shore, J. e Warden, S. (2007), The Art of Agile Development, O'Reilly Media, 1ª edição.
- Hastie, S. (2009) “No Easy Road To Agile”, <http://www.infoq.com/br/news/2009/07/no-easy-road-to-agile>. Acessado em 16 de Setembro de 2013.
- Boeg, J. (2012) “Kanban em 10 passos”, <http://www.infoq.com/br/minibooks/priming-kanban-jesper-boeg>. Acessado em 16 de Setembro de 2013.
- Teles, V. M. (2006) “Extreme Programming”, <http://desenvolvimentoagil.com.br/xp>. Acessado em 13 de Outubro de 2013.
- Teles, V. M. (2005) “Um Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming”, Dissertação de Mestrado em Informática da Universidade Federal do Rio de Janeiro – UFRJ.
- Wildt, D. F. e Lacerda, G.S. (2010) “Conhecendo o eXtreme Programming (XP)”, <http://www.slideshare.net/dwildt/conhecendo-o-extreme-programming>. Acessado em 13 de Outubro de 2013.
- Sato, D. e Bassi, D. (2007) “Métodos Ágeis de Desenvolvimento de Software e a Programação eXtrema (XP)”, <http://ccsl.ime.usp.br/agilcoop/files/AES-20070425.pdf>. Acessado em 14 de Outubro de 2013.

- Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks”, <http://xp123.com/articles/invest-in-good-stories-and-smart-tasks>. Acessado em 14 de Outubro de 2013.
- Freedman, R. (2010) “Comparing traditional and agile project management estimation techniques”, <http://www.techrepublic.com/blog/tech-decision-maker/comparing-traditional-and-agile-project-management-estimation-techniques/>. Acessado em 15 de Novembro de 2013.
- Hazan, C. (2008) Análise de Pontos de Função - Uma aplicação nas estimativas de tamanho de Projetos de Software. Engenharia de Software Magazine, 2ª edição.
- Lacey, M. (2012) “Estimando”, <http://msdn.microsoft.com/pt-br/library/hh765979.aspx>. Acessado em 15 de Novembro de 2013.
- Cohn, M. (2005) Agile Estimating and Planning, Prentice Hall, 1ª edição.
- Cohn, M. (2013) “Estimating with Tee Shirt Sizes”, <http://www.mountaingoatsoftware.com/blog/estimating-with-tee-shirt-sizes>. Acessado em 19 de novembro de 2013.
- Takeuchi, H., Nonaka, I. (1986) "The New Product Development Game", Harvard Business Review, <http://hbr.org/1986/01/the-new-new-product-development-game/>. Acessado em 21 de Novembro de 2013.
- Flanagan, D. e Matsumoto, Y. (2008) The Ruby Programming Language, O'Reilly Media, 1ª edição.
- Hartl, M. (2010) Ruby on Rails 3 Tutorial, Addison-Wesley Professional, 1ª edição.
- Douglas, K. e Douglas, S. (2005) PostgreSQL, Sams Publishing, 2ª edição.
- Chacon, S. (2009) Pro Git (Expert's Voice in Software Development), Apress, 1ª edição.
- Sutherland, J. (2013) “Story Points: Why are they better than hours?”, <http://scrum.jeffsutherland.com/2010/04/story-points-why-are-they-better-than.html>. Acessado em 01 de Dezembro de 2013.

Eis, D. (2011) “Introdução ao Responsive Web Design”, http://tableless.com.br/introducao-ao-responsive-web-design/#.Up_oqtJwpuQ.

Acessado em 01 de Dezembro de 2013.

Patton, T. (2013) “Put Bootstrap's responsive design features to good use”, <http://www.techrepublic.com/blog/web-designer/put-bootstraps-responsive-design-features-to-good-use/>.

Acessado em 02 de Dezembro de 2013.

Eis, D (2009) “Introdução sobre Media Queries”, <http://tableless.com.br/introducao-sobre-media-queries/#.UqNSzNJwpuQ>. Acessado em 07 de Dezembro de 2013.