

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**Aplicativos Móveis para Consulta de Informações de
Gerência Ágil de Processos utilizando LPS**

Bruno Fortes dos Santos

Florianópolis – SC

2013/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Aplicativos Móveis para Consulta de Informações de Gerência Ágil de Processos utilizando LPS

Bruno Fortes dos Santos

Trabalho de conclusão de curso
apresentado como parte dos
requisitos para obtenção do grau de
Bacharel em Sistemas de Informação

Florianópolis – SC

2013/1

Bruno Fortes dos Santos

Aplicativos Móveis para Consulta de Informações da Gerência Ágil de Processos utilizando LPS

Trabalho de conclusão de curso apresentado como parte dos requisitos para
obtenção do grau de Bacharel em Sistemas de Informação

Orientadora: Patrícia Vilain

Banca examinadora

Vitório Bruno Mazzola

Ricardo Pereira e Silva

Agradecimentos

Agradeço a Deus por ter guiado meu caminho ao longo de toda a graduação.

Agradeço minha vó Deozira, que me fez sonhar desde criança com a graduação em uma universidade federal.

Agradeço minha mãe Elza, meu pai Dirceu e meus irmãos Daniel e Samantha, vocês sempre me apoiaram e foram a base de toda essa conquista. Agradeço também aos demais familiares que me apoiaram nesse objetivo.

Agradeço minha namorada Elisa, pela paciência e compreensão que teve comigo na correria desses últimos meses da graduação, por me incentivar a concluir esse trabalho.

Agradeço a todos os amigos que fizeram parte da minha graduação, em especial os amigos da kktua que forneceram a companhia (e a cerveja) necessária para que a graduação fosse um momento inesquecível e repleto de boas lembranças.

Sumário

1. Introdução	10
1.1. Objetivos	12
1.1.1. Objetivo Geral	12
1.1.2. Objetivos Específicos	12
2. Desenvolvimento ágil de software	14
2.1. Métodos Ágeis	15
2.1.1. Extreme Programming	15
2.1.2. Scrum	23
2.2. Ferramentas de Apoio aos métodos ágeis	27
2.2.1. TinyPM	28
2.2.2. Pronto	29
2.2.3. Agilefant	31
3. Linha de Produtos de Software	33
3.1. Framework estendido	34
3.1.1. Engenharia de Domínio	36
3.1.2. Engenharia de Aplicação	37
4. Android	44
4.1. Informações Técnicas	44
4.1.1. Dalvik Virtual Machine	44
4.1.2. Informações Técnicas de Segurança	45
4.2. Android SDK.....	46
4.2.1. SDK Tools	46
4.2.2. SDK Platform	46
4.2.3. Imagens do Sistema	46
4.2.4. Google API	47
4.2.5. Android Support	47
4.3. Android Design.....	47
5. Linha de Produtos para Android para acesso às informações das ferramentas de apoio a métodos ágeis	53
5.1. Práticas ágeis para o desenvolvimento da LPS	53
5.2. Engenharia de Domínio	53
5.2.1. Lista de Requisitos	54
5.2.2. Modelo do Domínio	56

5.2.3. Projeto de Componentes	56
5.2.4. Representação da Arquitetura	57
5.2.5. Iteração 1	58
5.2.6. Iteração 2	61
5.3. Engenharia de Aplicação	65
5.3.1. Primeiro produto – Aplicativo Pronto.....	65
5.3.2. Segundo produto – Aplicativo Agilefant	70
5.3.3. Terceiro produto – Aplicativo TinyPM	74
6. Conclusão	79
7. Referências	81
8. Anexos	84

Lista de Figuras

Figura 1 - Ciclo Scrum	23
Figura 2 – Exemplo Taskboard	26
Figura 3 – Burndown Chart	27
Figura 4 - Conceitos Básicos de Linha de Produtos de Software	34
Figura 5 – Etapas do Framework estendido	36
Figura 6 - Modelo do domínio	56
Figura 7 - Representação da arquitetura	57
Figura 8 – Classes de comunicação	59
Figura 9 – Classes para manipulação dos dados	60
Figura 10 – Exemplo de teste de comunicação	60
Figura 11 - Classes produzidas na Iteração 2	61
Figura 12 - Interface gráfica produzida na Iteração 2	63
Figura 13 - Testes produzidos na Iteração 2	64
Figura 14 – Exemplo de teste de unidade	64
Figura 15 – Modelo de domínio do AppTCCPronto	67
Figura 16 – Classes do AppTCCPronto	69
Figura 17 – Abas da tela Taskboard do AppTCCPronto	69
Figura 18 – Modelo de domínio do AppTCCAgilefant	71
Figura 19 – Tela inicial do AppTCCAgilefant	73
Figura 20 – Modelo de domínio do AppTCCTinyPM	74
Figura 21 – Classes do AppTCCTinyPM	78
Figura 22 – Tela inicial do AppTCCTinyPM	79

Lista de Tabelas

Tabela 1 – Princípios para o desenvolvimento de interfaces gráficas para Android ... 47

Resumo

Nesse trabalho é apresentada uma série de três aplicativos para Android, desenvolvidos a partir de uma Linha de Produtos de Software. Esses aplicativos tem a função de apresentar as informações geradas por três ferramentas web para gerencia de projetos ágeis, cada aplicativo faz a comunicação com cada uma das ferramentas.

As metodologias ágeis são cada vez mais visadas pelas equipes de desenvolvimento de software, pois permitem a entrega contínua de partes da aplicação, agregando valores mais cedo para o sistema em desenvolvimento.

Para que as equipes ágeis atinjam seus objetivos, é necessário que todos os seus membros estejam acompanhando o andamento do projeto e das iterações. Também é importante que o cliente esteja acompanhando tudo o que a equipe está produzindo.

Diversas ferramentas web surgiram para auxiliar as equipes ágeis no planejamento e acompanhamento de suas iterações. Apesar de apresentarem características em comum, essas ferramentas também possuem pontos de variação, tanto na sua forma de implementação quanto na maneira como abordam as práticas dos métodos ágeis.

Para tornar o acompanhamento de projetos e iterações mais acessíveis a clientes e membros da equipe, esse trabalho propôs o desenvolvimento de três aplicativos para Android que acessem as informações geradas pelas ferramentas web de gerencia de projetos ágeis. Devido a existência de partes comuns e pontos de variação entre as ferramentas web, esse trabalho utilizou de uma Linha de Produtos de Software como mecanismo para desenvolvimento dos aplicativos propostos.

Palavras-chave: Métodos ágeis, Linha de Produtos de Software, Android, XP, Scrum, TinyPM, Pronto, Agilefant.

1. Introdução

Os métodos ágeis surgiram como uma alternativa aos métodos tradicionais de desenvolvimento de software e vem se popularizando cada vez mais entre as equipes de desenvolvimento. Dentre os fatores que colaboram para essa popularização destacam-se o aumento da interação com o cliente e geração mais rápida de *releases* que agregam valor ao produto. O desenvolvimento de software em pequenas *releases*, permite um acompanhamento por parte do cliente do andamento do produto, permitindo um *feedback* sobre as funcionalidades implementadas. [SOARES, 2004]

“Enquanto os métodos ágeis variam em termos de práticas e ênfases, eles compartilham algumas características, como desenvolvimento iterativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva. Desta forma existem maiores possibilidades de atender aos requisitos do cliente, que muitas vezes são mutáveis” [SOARES, 2004].

Diversas ferramentas surgiram para auxiliar as equipes de desenvolvimento na produção de softwares utilizando métodos ágeis. Com o desenvolvimento iterativo e incremental, é importante que toda a equipe possa acompanhar o andamento das iterações, avaliando durante o andamento de cada uma delas se o ritmo está sendo suficiente para produzir uma *release* ao final da iteração.

Hoje em dia, os smartphones estão sendo cada vez mais acessíveis, aumentando consideravelmente sua utilização e o surgimento de novas funcionalidades para o mesmo. O sistema operacional Android vem se popularizando e se tornando o principal sistema operacional para smartphones, tendo como característica forte o fato de ser gratuito e que suas aplicações podem ser desenvolvidas a partir de qualquer computador.

Junto a essa popularização do Android, está a necessidade de disponibilizar as informações geradas pelas ferramentas de gerência de métodos ágeis para todos os

envolvidos do projeto. Sendo assim, uma forma de disseminar essas informações é através dos dispositivos móveis dos envolvidos no projeto. Isto poderia ser feito, através de uma aplicação própria que não precisa ser carregada através do navegador e que tenha uma interface com o usuário voltada para o manuseio através do dispositivo móvel.

Uma das vantagens dos métodos ágeis é a possibilidade do cliente acompanhar o desenvolvimento do produto. Assim, a LPS proposta nesse trabalho será uma ferramenta acessível aos clientes, onde os mesmos poderão acompanhar o desenvolvimento das entregas por parte da equipe. Além do cliente, existe a possibilidade da equipe comercial, de empresas que utilizam métodos ágeis, acompanhar as próximas entregas e preparar um plano de comercialização delas, através de seu dispositivo móvel.

As ferramentas de apoio à gestão de projetos que utilizam métodos ágeis possuem características diferentes entre si, apesar dos seus objetivos em comum. A forma como as informações são organizadas e exibida aos usuários varia de ferramenta para ferramenta, além disso, algumas ferramentas costumam ser mais completas e abranger mais detalhes do projeto do que outras. Para lidar com essas diferenças e semelhanças e convergir para uma série de produtos para dispositivos móveis que satisfaça as equipes de desenvolvimento ágil, as linhas de produtos de software surgem com uma excelente alternativa de desenvolvimento.

Uma Linha de Produtos de Software (LPS) é um conjunto de softwares que compartilham entre si uma série de características que satisfazem as necessidades específicas de um determinado segmento de mercado [KRUEGER, 2006].

1.1. Objetivos

1.1.1. Objetivo Geral

O objetivo geral desse trabalho é o desenvolvimento de diferentes aplicações para dispositivos móveis na plataforma Android que proveem acesso às informações geradas por funcionalidades selecionadas de algumas ferramentas de gerenciamento de projetos para equipes ágeis. O desenvolvimento desses aplicativos deve ocorrer utilizando a abordagem de linha de produtos de software.

1.1.2. Objetivos Específicos

A seguir, são apresentados os objetivos específicos deste trabalho:

- Identificação das funções comuns entre as ferramentas, que atendam as necessidades das equipes ágeis;
- Definição das funcionalidades que serão implementadas na linha de produtos;
- Desenvolvimento de uma linha de produtos, composta de três aplicativos para acesso às informações das ferramentas de apoio a projetos ágeis selecionadas.

Nos próximos capítulos é feita uma fundamentação teórica dos conceitos abordados por esse trabalho. Essa fundamentação visa a apresentação e compreensão dos conceitos abordados, após essa etapa são descritos os passos realizados para o desenvolvimento da LPS proposta.

No capítulo 2 é abordado o conceito de métodos ágeis e descritas as principais práticas dos métodos ágeis mais conhecidos atualmente. Ainda no capítulo 2 são

descritas algumas ferramentas de apoio à gestão de projetos que utilizam métodos ágeis.

No capítulo 3 é feita a apresentação do conceito de Linhas de Produtos de Software, abordando um frameworks estendido para desenvolvimento de LPS.

Os conceitos do sistema operacional *Android* são parte importante desse trabalho, pois a LPS proposta é voltada para ele. No capítulo 4 é feita uma contextualização abordando os seus principais conceitos.

Após apresentar os conceitos anteriores, esse trabalho propõe uma LPS para dispositivos móveis *Android* que acesse as informações de ferramentas para gestão de equipes ágeis. No capítulo 5 é descrito o processo de desenvolvimento dessa LPS, desde as práticas utilizadas, passando pela sua arquitetura e seu desenvolvimento.

2. Desenvolvimento ágil de software

O desenvolvimento ágil de software surgiu a partir de algumas necessidades não contempladas pelos métodos tradicionais de desenvolvimento de software. Apesar de toda a evolução nas técnicas e ferramentas, a produção de software confiável dentro de prazo correto e que atenda os requisitos estipulados pelo cliente, ainda ocorre de forma difícil [SOARES, 2004]. O ciclo de vida clássico da engenharia de software, também chamado de modelo cascata, requer uma abordagem sistemática, iniciando com análise e avançando até a codificação, testes e manutenção. Conforme Roger S. Pressman em seu livro “Engenharia de Software” [PRESSMAN, 2001], alguns problemas que surgem quando o modelo cascata é aplicado são:

1. Os projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Alguma iteração sempre ocorre surgindo novas necessidades do cliente, isso traz problemas na aplicação do paradigma.
2. Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente. O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projetos.
3. O cliente deve ter paciência. Uma versão de trabalho do(s) programa(s) não estará disponível até um ponto tardio do cronograma do projeto. Um erro crasso, se não for detectado até que o programa de trabalho seja revisto, pode ser desastroso.

Como solução a esses problemas surgem os métodos ágeis valorizando a iteração do cliente com o desenvolvimento e a entrega contínua de partes do software que agreguem valores. O desenvolvimento de forma incremental e entrega de pequenas partes do software, permite que novos requisitos sejam incluídos ao longo do processo de desenvolvimento e que o software comece a agregar valor para o cliente antes de sua entrega total.

O reconhecimento dos métodos ágeis se dá a partir de 2001 com a publicação do “Manifesto para o desenvolvimento ágil de software” [MANIFESTO, 2012] que determina a valorização de:

Indivíduos e interação entre eles mais que processos e ferramentas;

Software em funcionamento mais que documentação abrangente;

Colaboração com o cliente mais que negociação de contratos;

Responder a mudanças mais que seguir um plano.

2.1. Métodos Ágeis

Apoiados no desenvolvimento ágil de software surgem diversos métodos ágeis. Abaixo serão descritos os métodos mais praticados, que são o *Scrum* e o *Extreme Programming*, esses métodos também costumam ser os mais abordados pelas ferramentas de gerência de métodos ágeis.

2.1.1. Extreme Programming(XP)

O Extreme Programming(XP) é um método ágil criado na década de 90 por Kent Beck, um dos principais apoiadores do Manifesto Ágil. O método XP surge com a proposta de ser utilizado por equipes de dois a dez programadores, com foco no desenvolvimento de softwares com requisitos vagos que se modificam rapidamente [BECK, 2004].

O XP contribui para evitar desperdícios com funcionalidades desenvolvidas que podem ser poucas ou nunca utilizadas pelo cliente, através de iterações semanais com a priorização no desenvolvimento de funcionalidades que agreguem mais valores ao software proposto, as tarefas que agreguem menor valor são deixadas para o final

do projeto, com isso caso ocorra uma diminuição do escopo, as funcionalidades que agregam menor valor deixam de ser desenvolvidas [AG, 2010].

2.1.1.1. Valores

O XP se baseia em cinco valores fundamentais [BECK, 2004]:

2.1.1.1.1. Comunicação

O XP valoriza muito a comunicação e prioriza o diálogo presencial como forma de comunicação, devido aos inúmeros elementos que colabora para a compreensão do assunto, como gestos, expressões faciais, postura, palavras verbalizadas, tom de voz, entre outros.

2.1.1.1.2. Simplicidade

Assegurar que a equipe está focada em desenvolver primeiro o que for essencial para o projeto, evitando fazer o que poderia vir a ser necessário, mas que ainda não se provou essencial.

2.1.1.1.3. Feedback

Desenvolvedores procuram entregar novas funcionalidades o mais rápido possível para os clientes analisarem se está compatível com aquilo que pediu. Os clientes procuram se manter o mais próximo possível aos programadores, para prover informações às dúvidas que eles tenham ao longo do desenvolvimento.

2.1.1.1.4. Coragem

As equipes XP acreditam que errar é natural e quebrar o que vinha funcionando pode acontecer eventualmente. É necessário ter coragem para lidar com esses riscos.

2.1.1.1.5. Respeito

Respeito é o valor que sustenta todos os anteriores, a equipe só irá se comportar como um time se o respeito existir entre todos os membros da equipe.

2.1.1.2. Práticas

O XP se baseia em um conjunto de práticas para aplicar os seus valores e fundamentos, essas práticas estão fortemente relacionadas entre si, abaixo são descritas as principais práticas [EXTREME PROGRAMMING, 2012]:

2.1.1.2.1. Estórias de Usuário

Servem para descrever um requisito do usuário, possui uma finalidade semelhante aos casos de usos utilizados nos métodos tradicionais, porém com algumas diferenças. As estórias de usuários possuem um nível de detalhamento menor, exigem apenas um nível de detalhamento que seja suficiente para estimar o tempo necessário para desenvolvimento. Além disso, as estórias de usuário são baseadas nas necessidades do usuário, de acordo com a necessidade é atribuído um nível de importância à estória, esse nível de importância associado ao tempo estimado é utilizado na elaboração do *release planning*.

Outra característica da estória de usuário é a presença de um teste de aceitação associado a ela, esse teste de aceitação descreve os critérios para verificar se a estória foi implementada corretamente.

2.1.1.2.2. Planejamento de entrega (*Release Planning*)

No planejamento de entregas (*release planning*) os desenvolvedores definem o número de semanas necessárias para o desenvolvimento de cada estória de usuário, considerando que as semanas seriam dedicadas somente para

desenvolvimento daquela estória de usuário, sem dependências ou sem trabalhos extras para realizar, porém com a inclusão dos testes para ela. Os clientes definem quais estórias de usuário são mais importantes ou possuem maior prioridade.

As estórias de usuário são escritas ou impressas em cartões e distribuídas em uma mesa, onde desenvolvedores e clientes movem os cartões definindo quais serão implementadas na primeira versão e quais serão deixadas para as versões posteriores.

A equipe então determina a velocidade do projeto, que é um indicador de quantas estórias podem ser implementadas até certa data (tempo) ou quanto tempo um conjunto de estórias demoram para serem implementadas (escopo).

Com o tempo de desenvolvimento estimado para cada estória e a velocidade do projeto, a equipe consegue determinar quais estórias farão parte da iteração e irão compor o *release* a ser entregue ao final dela. [EXTREME PROGRAMMING, 2012]

2.1.1.2.3. Pequenas entregas (*Small Releases*)

Realizar pequenas entregas permite que logo no começo do projeto você entregue uma versão de produção pra seu cliente, com alguns requisitos testados e funcionando, já agregando valor ao produto. Através do planejamento de entrega são planejadas essas pequenas entregas e o que estará pronto em cada uma delas. [EXTREME PROGRAMMING, 2012]

2.1.1.2.4. Iterações e Planejamento de iterações

O cronograma de desenvolvimento deve ser dividido em iterações, cada iteração deve ter de 1 a 3 semanas de duração, sendo 1 o número ideal. É importante que as iterações tenham sempre o mesmo tamanho, dessa forma é possível medir o desempenho da equipe.

As tarefas de programação de cada iteração devem ser planejadas em um momento de planejamento da iteração, sempre no início de cada nova iteração. As histórias de usuários a serem implementadas são selecionadas do *release plan*, respeitando a ordem de prioridade entre elas. *Bugs* encontrados também podem ser incluídos para correção junto a iteração. [EXTREME PROGRAMMING, 2012]

2.1.1.2.5. Ambiente de trabalho aberto

Remover as barreiras entre as mesas de trabalho ou colocar os computadores reunidos em uma área central, ter um amplo espaço para as reuniões diárias em pé, incluir quadros brancos ou paredes brancas onde os cartões das histórias de usuário possam ser fixados são exemplos de atitudes que contribuem para ter um ambiente que estimula o trabalho em conjunto, muito mais eficiente que cada um trabalhar em seu espaço.

2.1.1.2.6. Ritmo sustentável

Conhecer a capacidade da equipe e manter o mesmo ritmo de desenvolvimento do início ao fim da iteração, sem causar esforço além do necessário no início da iteração ou relaxar no início da iteração e realizar hora extra no final, a fim de cumprir a entrega estipulada.

2.1.1.2.7. Reunião diária em pé

Realizar uma reunião diária em pé com toda a equipe no início da manhã, nessa reunião são apresentados os problemas, impedimentos e soluções encontradas. O fato de os participantes estarem em pé, evita longas discussões e mantém o foco da reunião.

2.1.1.2.8. Velocidade do projeto

É uma medida para indicar quanto de trabalho a equipe é capaz de entregar ao longo do tempo. Essa medida é obtida após a execução de algumas iterações, analisando a quantidade e complexidade de histórias executadas em cada uma delas.

2.1.1.2.9. Ajustar o XP

Procurar sempre melhorar o processo para que funcione da forma esperada. As regras estipuladas pelo XP devem ser seguidas, porém a equipe não deve hesitar em modificá-las caso seja necessário. Identificar o que está funcionando e o que pode ser mudado para melhorar os processos do XP.

2.1.1.2.10. Simplicidade

Procurar simplificar o projeto, focar naquilo que realmente é necessário para o usuário e buscar simplificar os problemas mais complexos. A equipe costuma tentar prever aquilo que o usuário possa vir a necessitar, porém é necessário implementar o que for essencial para resolver um problema no momento. Se o sistema é mantido de forma simples, então tudo o que for acrescentado a ele será simples também.

2.1.1.2.11. Metáforas

O XP utiliza o conceito de metáforas como um mecanismo para alinhar o pensamento dos diferentes projetistas e desenvolvedores do software. É uma forma de manter uma visão única sobre como adicionar novas funcionalidades ao software.

2.1.1.2.12. Cartões CRC

Cartões CRC (Classe, Responsabilidade e Colaboração) é uma forma para que toda a equipe possa ajudar a projetar o sistema. São identificadas as

responsabilidades de cada classe e as colaborações determinam as iterações entre essas classes.

2.1.1.2.13. Não adicionar funções antecipadamente

Como descrito anteriormente, o XP busca a simplicidade dos projetos. Mais funcionalidades implicam em mais trabalho para a equipe, levando a equipe a desprender esforço em funcionalidades que podem vir a nunca serem utilizadas. É importante focar naquilo que realmente é importante para o usuário.

2.1.1.2.14. Refatoração

Com a adição de novas funcionalidades ao sistema, o código fonte acaba sofrendo grandes modificações, o objetivo da refatoração é manter o código simples da forma como foi elaborado inicialmente. Manter o código limpo e organizado facilita a sua leitura, entendimento e manutenção.

2.1.1.2.15. Cliente sempre disponível

O cliente não somente deve auxiliar a equipe, como também deve fazer parte dela. Todas as fases do XP requerem comunicação com o cliente, a comunicação deve ocorrer face a face, por ser o meio mais eficiente de comunicação.

Pelo fato de muitos detalhes serem deixados de fora da história de usuário, existirá a necessidade de os desenvolvedores conversarem com os clientes para obterem todos os detalhes necessários.

2.1.1.2.16. Escrever os testes unitários antes

Os testes unitários servem para garantir que todas as partes do software funcionem corretamente. Escrever os testes unitários antes facilita a visualização do

que precisa ser feito durante o processo de codificação, consequentemente facilitando o processo de codificação.

2.1.1.2.17. Programação em par

Consiste em dois programadores trabalharem lado a lado em um mesmo computador ao mesmo tempo, resolvendo um mesmo problema em conjunto. A programação em par traz benefícios como a melhor inspeção do código que está sendo produzido, traz melhorias na disseminação do conhecimento entre a equipe e ajuda a manter o ritmo de programação.

2.1.1.2.18. Integração Contínua

Quando diversos programadores estão trabalhando em um mesmo projeto, corre-se o risco de vários deles estarem alterando a mesma parte do projeto ao mesmo tempo, gerando problemas de conflitos no código. Além disso, os programadores “precisam ser capazes de evoluir rapidamente sem interferir no trabalho uns dos outros” [TELES, 2004 – p. 114-116]. Para resolver esses problemas, o XP utiliza a prática de Integração Contínua, onde os diversos programadores ou pares trabalham com cópias isoladas do código fonte, porém integram suas alterações diversas vezes ao dia com o código mais recente produzido pela equipe. Quando o código é integrado, os testes unitários e de aceitação são rodados para verificar a integridade do código como um todo. [TELES, 2004 – p. 114-116]

2.1.1.2.19. Testes de Aceitação

Durante o planejamento da iteração uma história de usuário selecionada é traduzida em um Teste de Aceitação, podendo ser mais de um teste de aceitação para uma única história. Um cenário é especificado pelo cliente onde se pode verificar se a história foi corretamente implementada. Os clientes são os responsáveis por

verificarem se os testes de aceitação estão corretos e se os resultados são os esperados. Uma história só é considerada concluída quando passa no teste de aceitação [EXTREME PROGRAMMING, 2012].

2.1.2. Scrum

O Scrum é um framework que fornece práticas iterativas e incrementais para gerenciamento de equipes de desenvolvimento de software. No Scrum as iterações são chamadas de Sprint, sendo uma Sprint um ciclo onde um conjunto de atividades é executado. O fluxo de trabalho no Scrum segue de acordo com a Figura 1, apresentada abaixo.

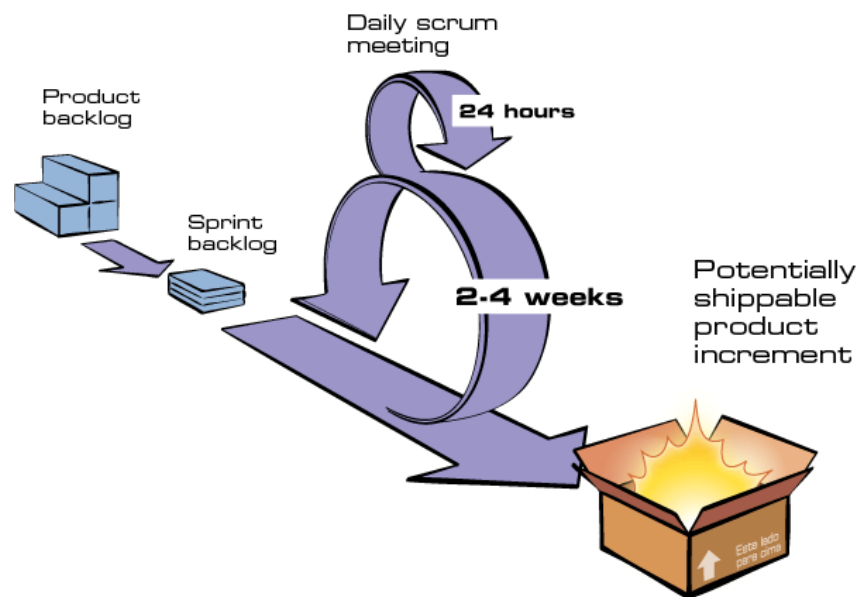


Figura 1 - Ciclo Scrum [IMPROVE, 2012]

Abaixo são apresentados mais detalhes em torno de cada uma das etapas do fluxo de trabalho do Scrum.

2.1.2.1. Product Backlog

O Product Backlog contém a lista de requisitos do sistema a ser desenvolvido pelo projeto. O Product Backlog nunca é completado e existe por todo o tempo que o produto existir. Ele é constantemente modificado para identificar o que o produto

necessita para ser apropriado, competitivo e eficiente. O Product Backlog utilizado no início do planejamento do projeto é meramente uma estimativa inicial dos requisitos. [SCHWABER, 2004]

2.1.2.2. Sprint Backlog

O Sprint Backlog define as tarefas extraídas do Product Backlog pela equipe para serem realizadas em uma Sprint, com base nas prioridades definidas pelo Product Owner e na estimativa de tempo definida pela equipe para completar as tarefas. Durante uma Sprint cabe ao Scrum Master manter o Sprint Backlog atualizado para refletir que tarefas são completadas e a estimativa de tempo da equipe para completar as que ainda não estão prontas.

2.1.2.3. Sprint

A Sprint é um timebox de 30 dias consecutivos. Sendo esse o tempo necessário para a equipe produzir uma versão do produto que pode ser entregue e que atenda a um interesse significativo do Product Owner e dos stakeholders. É o tempo máximo que os stakeholders aguardam sem perder o interesse no progresso da equipe de desenvolvimento. [SCHWABER, 2004]

Antes do início de uma Sprint é realizado um Sprint Planning Meeting onde são selecionados itens do Product Backlog e é feita a preparação de um Sprint Backlog. No decorrer da Sprint no início de cada dia deve ocorrer um Daily Scrum Meeting, onde é feita uma retrospectiva do dia anterior, definida as atividades para o dia atual e discutido sobre impedimentos que possam vir a ocorrer no dia.

Ao final da Sprint é realizada uma reunião com o Product Owner e os stakeholders chamada de Sprint Review Meeting onde são apresentadas as novas funcionalidades desenvolvidas. Também ao fim da Sprint é realizado um Sprint Retrospective Meeting, destinado à equipe de desenvolvimento, sem necessariamente

contar com o Product Owner, onde a equipe discute sobre o que ocorreu de maneira adequada na Sprint anterior e o que pode ser melhorado para as próximas Sprints.

2.1.2.4. Visibilidade no Scrum

Uma das principais missões do Scrum é tornar visível para a equipe o seu progresso e os problemas que precisam ser resolvidos para melhorar os resultados. Visibilidade é um dos pilares mais interessantes da gestão ágil e que traz mudança forte no cotidiano da equipe [AGILE, 2010]. O Scrum se utiliza de alguns artifícios interessantes para promover a visibilidade do andamento do projeto, sendo esses artifícios o Taskboard e o BurnDow Chart, apresentados a seguir.

2.1.2.4.1. Taskboard

O *Taskboard* é um quadro normalmente dividido em três colunas, a primeira coluna com as atividades previstas para a Sprint, a segunda coluna com as atividades em desenvolvimento e a terceira coluna com as atividades concluídas. As atividades são escritas em *post-it* e movidas entre as colunas nas reuniões diárias da equipe. Recomenda-se a utilização de cores diferentes nos *post-it* para cada tipo de atividade (desenvolvimento, teste, planejamento). Recomenda-se também que no título de cada coluna seja colocada a capacidade da equipe para aquela coluna, rejeitando atividades que façam com que o total de esforço das atividades na coluna seja maior que a capacidade da equipe. No rodapé de cada coluna do *Taskboard* pode-se colocar os critérios/condições que uma atividade deve atender para entrar naquela coluna. A Figura 2 mostra o exemplo de um *taskboard*.

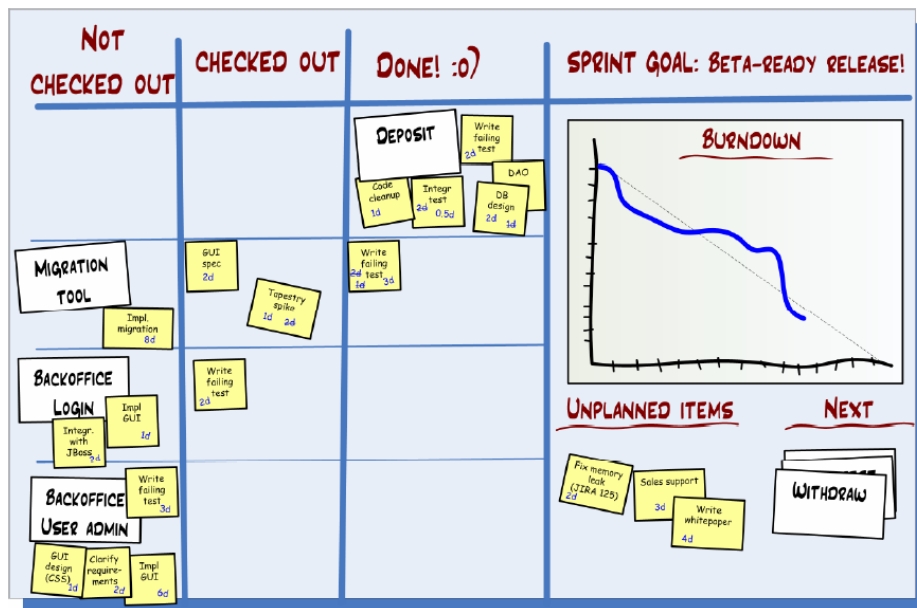


Figura 2 – Exemplo Taskboard [KNIBERG, 2007]

2.1.2.4.2. Burndown Chart

O *Burndown Chart* é uma forma de visualizar a relação entre o trabalho restante em qualquer ponto do tempo e o progresso da equipe em reduzir esse trabalho. Através do cruzamento dos pontos dos eixos horizontal e vertical do gráfico é possível ter uma probabilidade de quanto trabalho estará concluído em um dado momento do tempo. A Figura 3 é um exemplo de *Burndown chart*:

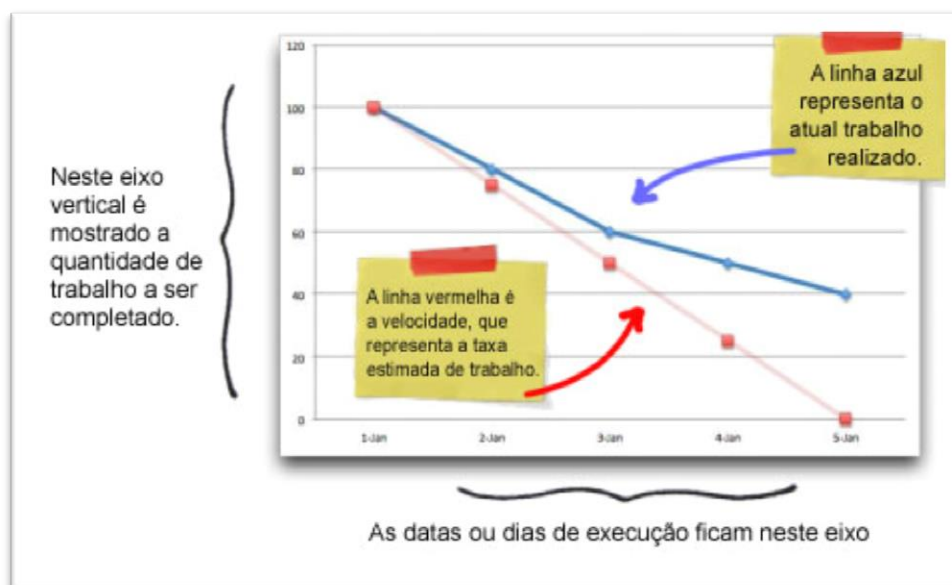


Figura 3 – Burndown Chart [CRUZ, 2012]

2.2. Ferramentas de Apoio aos métodos ágeis

Para apoiar as equipes na adoção de métodos ágeis e aplicação de suas práticas da forma mais eficiente, várias ferramentas de apoio surgiram. Cada uma com características diferentes e apoio a métodos ágeis específicos.

Para o desenvolvimento desse trabalho foram selecionadas três dessas ferramentas. Como critérios para seleção foram definidos: (1) a existência de uma versão *Free* da ferramenta (independente da quantidade de usuários liberados); (2) possibilidade de instalação da ferramenta na máquina local ou em servidor específico (critério importante na facilitação no processo de desenvolvimento e teste da ferramenta); e (3) possibilidade de acessar os dados gerados pelas funcionalidades selecionadas na ferramenta, via API disponibilizada por ela ou através de acesso direto ao seu banco de dados.

Foram identificadas as funcionalidades que atendessem as principais práticas dos métodos ágeis, como gestão de histórias de usuários, tarefas, iterações e quadro de acompanhamento das iterações.

Abaixo segue as ferramentas selecionadas, suas características e funcionalidades selecionadas para o desenvolvimento desse projeto:

2.2.1. tinyPM

O tinyPM é uma ferramenta avançada, leve e inteligente para colaboração com equipes que utilizam métodos ágeis. Ela inclui *backlog*, *taskboard*, estórias de usuário e REST API. O tinyPM é desenvolvido e mantido pela empresa Agilers Sp. z o.o. da Polônia¹.

O tinyPM roda sobre o servidor *Apache Tomcat* e banco de dados *MySQL*. Possui uma versão *free* para até três projetos e dois usuários. Para projetos acadêmicos e *open source* é possível solicitar uma licença para adicionar novos usuários e projetos. Neste trabalho será utilizada a versão para até dois usuários, pois ela já provê os requisitos necessários.

As funcionalidades selecionadas no tinyPM para a LPS proposta nesse projeto foram:

Taskboard

Demonstra as atividades selecionadas para uma iteração, organizadas em três colunas: *Pending* (com as atividades pendentes), *In Progress* (com as atividades sendo realizadas) e *Completed* (com as atividades finalizadas).

Permite que as atividades sejam arrastadas entre as colunas, atualizando os status delas quando trocadas de coluna.

¹ A ferramenta tinyPM pode ser baixada em <http://www.tinypm.com/>

Backlog Iteration

Lista das histórias de usuários selecionadas para a iteração atual. É possível acessá-lo através do menu *Backlog*. Clicando sobre as histórias é possível editar seu conteúdo e incluir e alterar as atividades relacionadas a ela.

Estórias de Usuário

Funcionalidade para controle de todas as histórias de usuário cadastradas para o projeto, não existe uma tela específica para elas, é possível acessá-las, editá-las e incluir novas histórias através do menu *Backlog*.

Como citado na descrição dos métodos Scrum e XP, o planejamento de iterações e *sprints* se dão sobre as histórias de usuários definidas no projeto, por esse motivo é fundamental conhecê-las.

Task (Tarefas)

São as tarefas ou atividades do projeto. Elas estão associadas às histórias e ditam as atividades a serem realizadas para a conclusão de cada uma das histórias.

2.2.2. Pronto

O Pronto é um projeto para gestão de projetos ágeis². Ele surgiu em 2008 como trabalho de conclusão do curso de Sistemas de Informação da Faculdade de Informática e Administração Paulista, desenvolvido por Luiz Faias Jr e André Faria Gomes. O Pronto é um projeto open-source que roda sobre o servidor *Apache Tomcat* com banco de dados em PostgreSQL. Possui sua documentação e código fonte escrito em português. Provê suporte a todos os requisitos necessários na gestão de projetos utilizando Scrum. O Pronto foi selecionado para esse trabalho por já ser uma ferramenta conhecida do autor. Além de atender os requisitos e ser open-source.

² A ferramenta Pronto pode ser baixada em <https://github.com/bluesoft/pronto-agile>

Para a LPS proposta nesse trabalho foram selecionadas as seguintes funcionalidades:

Product Backlog

Lista as histórias de usuários cadastradas para o produto em desenvolvimento. É possível priorizar histórias e fazer uma estimativa de tempo do Backlog com base no esforço estipulado para elas

Backlog da Sprint atual

Lista as histórias de usuários selecionadas para Sprint sendo executada no momento, é possível estimar o tempo total da Sprint com base no esforço estimado para as histórias e definir a prioridade delas.

Estórias

O Pronto possui cadastro e lista das histórias para o projeto sendo gerenciado. As histórias cadastradas podem ser organizadas por categorias definidas pelo usuário e classificadas como “Defeito” quando for o caso. Na tela de planejamento da Sprint é feito o cadastro das Estórias que serão executadas nela.

Quadro Kanban

O quadro Kanban do Pronto possui a função do Taskboard do Scrum citado anteriormente na seção 2.1.2.4.1. Ele lista as histórias organizadas em colunas com as fases do desenvolvimento: “*To Do*”, *Doing*, *To teste*, *Testing* e *Done*. As histórias podem ser arrastadas de uma coluna para outra causando a alteração de seu status.

2.2.3. Agilefant

Agilefant é uma ferramenta web *open-source* (distribuída sob a licença MIT) para gestão de *backlog*³. Ela procura ser o mais simples possível, porém visa atender desde pequenas equipes até equipes com larga escala de desenvolvimento. Roda sobre o servidor *Apache Tomcat*, com banco de dados em *MySQL*. A ferramenta Agilefant foi encontrada através da pesquisa pelos termos “free open source tool agile” no *Google*.

Para a LPS proposta nesse projeto, foram selecionadas as seguintes funcionalidades do Agilefant:

Products

Cadastro do produto sendo desenvolvido.

Projects

O Agilefant entende que o desenvolvimento de um produto é composto por um ou vários projetos, então para um produto são feitos os cadastros dos projetos através da funcionalidade *Projects*.

Iterations

Permite cadastrar as iterações dentro do projeto, antes de iniciar uma iteração ela deve ser cadastrada para que as histórias a serem desenvolvidas nela possam ser associadas a ela.

³ A ferramenta Agilefant pode ser baixada em <http://www.agilefant.org/>

Stories

Cadastro das estórias de usuário do projeto. No cadastro da estória é possível informar o esforço dela, com esse esforço é possível fazer o planejamento das iterações com as estórias que farão parte dela.

3. Linha de Produtos de Software

Uma Linha de Produtos de Software (LPS) é um conjunto de softwares que compartilham em comum, uma série de características que satisfazem as necessidades específicas de um determinado segmento de mercado, e que são desenvolvidos a partir de um conjunto base de componentes previamente planejados para esse segmento.

Os avanços recentes no campo das LPSs têm demonstrado que a aplicação desses conceitos pode render grandes melhorias no tempo para o produto chegar ao mercado, escalabilidade do portfólio e os custos com engenharia de software [KRUEGER, 2011].

O que difere a LPS do reuso oportunista de software, é que o desenvolvimento de componentes para a LPS ocorre de forma previamente planejada. Ao invés de se colocar todos os componentes dentro de uma biblioteca com a esperança de surgirem oportunidades para reutilizá-los, na LPS os artefatos de software só são criados quando é previsto que serão utilizados em um ou mais produtos da LPS [KRUEGER, 2011].

Abaixo seguem os quatro conceitos envolvidos que melhor podem descrever uma LPS, após é apresentada a Figura 4 que demonstra como esses conceitos estão envolvidos [KRUEGER, 2006]:

Componentes de Software (Software asset inputs):

Uma coleção de componentes de softwares (como requisitos, código fonte de componentes, testes, arquitetura e documentação) que podem ser configurados e compostos de diferentes formas para criar todos os produtos da LPS. Para acomodar as variações entre os produtos, os componentes podem ser opcionais ou possuírem pontos de variação que podem ser configurados de diferentes formas para prover diferentes comportamentos.

Modelo de *feature* e decisões de produtos (Product Decisions)

O modelo de *feature* descreve as características opcionais e variáveis para o produto de uma LP. Cada produto da LP é único e definido pelas suas decisões de produto (escolha das características opcionais e variáveis do modelo de decisão).

Mecanismo de produção e processo (Production)

São os meios para composição e configuração de produtos a partir dos componentes de software. Decisões de produtos são utilizados durante o processo para determinar quais componentes de software utilizar e como configurar os pontos de variação com esses componentes.

Softwares produzidos (Software Product Outputs)

A coleção de todos os produtos que podem ser produzidos para a LPS. O escopo da LPS é determinado pelo conjunto de softwares que podem ser produzidos a partir dos componentes de software e do modelo de decisão.

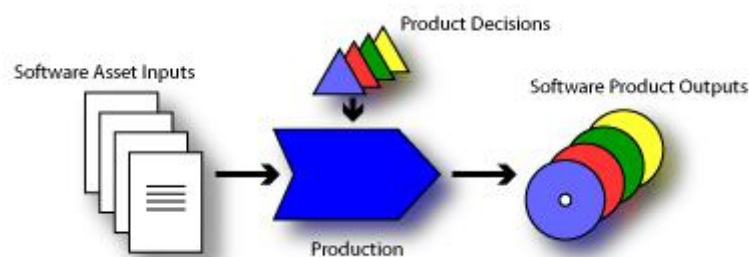


Figura 4 - Conceitos Básicos de Linha de Produtos de Software [KRUEGER, 2006]

O desenvolvimento de uma LPS consiste em duas atividades principais, que são engenharia de domínio e engenharia de aplicação. A engenharia de domínio trata do levantamento de requisitos e elaboração dos componentes da LPS, já a engenharia

de aplicação consiste no desenvolvimento de aplicações, através da seleção de componentes obtidos no processo de engenharia de domínio.

3.1. Framework estendido

Nesta seção, será descrito um framework de seleção de práticas ágeis estendido para LPS [SOUZA, 2012]. Essa extensão será seguida no desenvolvimento da LPS proposta nesse trabalho. Segundo [SOUZA, 2012], existem pontos conflitantes entre os conceitos que envolvem o desenvolvimento ágil e o desenvolvimento de LPS. Enquanto o desenvolvimento ágil busca uma redução nas etapas de documentação e diagramação, a LPS exige essa etapa para que o domínio seja modelado de forma eficiente.

O framework de métodos ágeis proposto inicialmente em [FAGUNDES, 2005] foi elaborado a partir de uma análise comparativa de diferentes métodos ágeis e reunia as melhores práticas dos métodos ágeis, porém, não era adequado ao desenvolvimento de LPS. O trabalho realizado em [SOUZA, 2012] acrescentou práticas utilizadas na LPS ao framework elaborado em [FAGUNDES, 2005], essa extensão “permitiu a especificação de um framework que mostra as etapas e práticas para o desenvolvimento de toda uma linha de produtos de software” [SOUZA, 2012 – p. 57].

O processo de desenvolvimento da LPS é dividido em dois ciclos, denominados “Engenharia de Domínio” e “Engenharia de Aplicação”, e cada ciclo fica subdividido em atividades menores. Durante o processo de engenharia de domínio, são coletadas as informações e requisitos referentes ao domínio da LPS que se pretende desenvolver, são definidos quais requisitos são comuns a todos os produtos da LPS e quais fazem parte dos pontos de variabilidade. Durante o processo de engenharia da aplicação, um produto específico é desenvolvido a partir da LPS para

desenvolvimento, ou seja, o produto é obtido através da utilização dos artefatos produzidos na engenharia de domínio.

A Figura 5 ilustra cada uma das etapas do framework estendido, e em seguida é feita uma descrição breve de cada uma das atividades [SOUZA, 2013]:

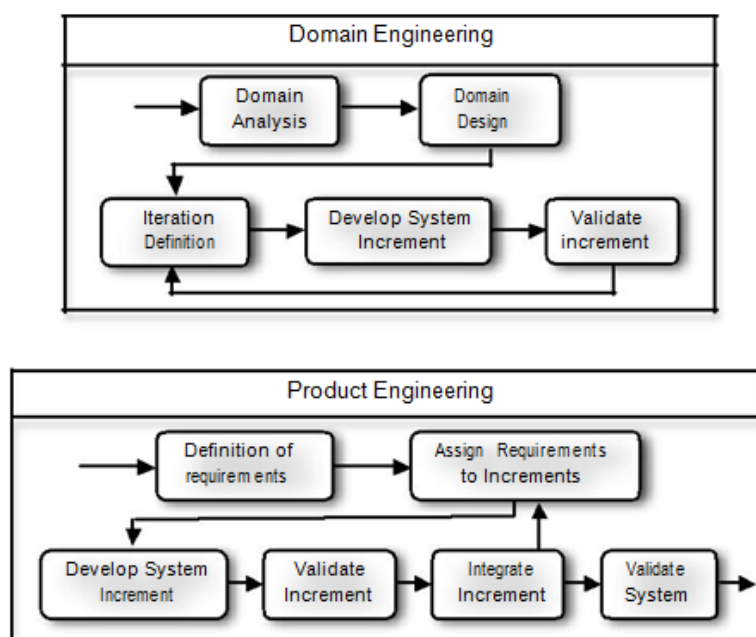


Figura 5 – Etapas do Framework estendido [SOUZA, 2013]

3.1.1. Engenharia de Domínio

A engenharia de domínio consiste no processo de coleta das informações e requisitos referentes ao domínio da LPS que se pretende desenvolver. É nessa etapa que são definidos os requisitos comuns aos produtos da LPS e os pontos de variabilidade. A engenharia de domínio acontece através da realização das atividades descritas abaixo:

3.1.1.1. Atividade análise de domínio

Realização de uma análise geral das aplicações que pertencem à LPS e as características de cada uma delas, composto pelas seguintes práticas:

Lista de requisitos de cada aplicação

Mapeamento das aplicações que farão parte da LPS, com a lista de requisitos de cada uma delas.

Modelo de *features*

Um modelo que destaca os requisitos identificados para as aplicações da LPS, o modelo deve identificar a hierarquia entre os requisitos. Também deve identificar quais requisitos são obrigatórios e quais são pontos de variação da LPS.

Documentação

Breve descrição das características e do modelo gerado. Essa documentação também pode apresentar uma visão geral da LPS, com estimativas de custos, riscos e expectativas para cada produto.

3.1.1.2. Atividade projeto de domínio

Especificação da arquitetura reutilizável do sistema.

Projeto de componentes

Definição de um componente para cada requisito, ou então um requisito pode ser quebrado em vários componentes, ou ainda vários requisitos podem gerar um único componente [SOUZA, 2012].

Representação da arquitetura do domínio

Representar a arquitetura desenvolvida no passo anterior baseada nos componentes especificados. Um diagrama de pacotes pode ser utilizado para descrevê-la, por exemplo [SOUZA, 2013].

3.1.1.3. Atividade definição da iteração

Definição de cada uma das iterações.

Planejamento da iteração

Planejar como a iteração será desenvolvida, distribuição dos componentes ou requisitos entre os membros da equipe de desenvolvimento. Estimar o tempo necessário para desenvolvimento da iteração.

3.1.1.4. Atividade Desenvolvimento de incrementos do sistema

Em cada iteração, o desenvolvimento das funcionalidades identificadas da LPS, e de novas que surgirem, deve seguir os seguintes passos:

Reunião Diária

Reuniões curtas e diárias entre os membros da equipe com o objetivo de debater os progressos ou dificuldades.

Estórias de usuário

Breve descrição de como usar as novas funcionalidades, fornecido pelo usuário.

Casos de uso

Cada requisito pode possuir um caso de uso correspondente.

Projeto da iteração

Projeto para descrever a nova funcionalidade.

Escrita dos testes de unidade

Escrever testes de unidade antes da codificação.

Escrita dos testes de aceitação

Testes de aceitação também devem ser escritos antes da codificação.

Desenvolvimento da iteração

Codificação dos requisitos selecionados para a iteração.

Desenvolvimento de código coletivo

Quando o código fonte é escrito por vários programadores é necessários que todos conheçam e entendam todo o código que está sendo produzido, não somente a parte que cada programador escreveu.

Programação em pares

A programação em pares sugere que um programador escreva o código, enquanto o outro observa e sugere modificações.

Refatoração

Refatorar o código toda vez que a equipe julgar necessário.

Programação lado-a-lado

Colocar os programadores para trabalharem lado-a-lado no ambiente de trabalho é uma forma de melhorar o relacionamento entre os membros da equipe.

Integração contínua

Sempre que finalizada uma tarefa, o código deve ser integrado com a versão mais recente.

3.1.1.5. Atividade validação do incremento

Inspeção do código

Os programadores devem verificar os códigos uns dos outros, a fim de encontrar problemas que possam ter sido inseridos no código a partir do novo incremento.

3.1.2. Engenharia de Aplicação

A engenharia de aplicação é o processo de desenvolvimento de cada um dos produtos que compõe a LPS. Nessa etapa um produto em específico é selecionado para o desenvolvimento, o produto é produzido através da utilização dos artefatos da engenharia de domínio. Abaixo segue uma descrição das atividades que compõe a engenharia de aplicação:

3.1.2.1. Atividade definição dos requisitos

Lista de requisitos

Definir uma lista de requisitos para o produto que será desenvolvido.

Seleção dos componentes

Selecionar, do núcleo de artefatos, os componentes que serão utilizados para atender os requisitos definidos para o produto.

Modelo geral

Se existir algum novo requisito para esse produto, pode ser feita uma especialização do modelo de *features* existente com a inclusão desse novo requisito.

3.1.2.2. Atividade atribuir requisitos nas iterações

Planejamento das iterações

Planejar como as iterações serão desenvolvidas, distribuição dos módulos ou requisitos entre os membros da equipe de desenvolvimento. Estimar o tempo necessário para desenvolvimento das iterações.

3.1.2.3. Atividade desenvolvimento de incrementos do sistema

Reuniões diárias

Reuniões curtas e diárias, onde a equipe apresenta seu progresso e suas dificuldades.

Estórias do usuário

Breve descrição de como usar as novas funcionalidades, fornecido pelo usuário.

Casos de uso

Cada requisito deve possuir um caso de uso correspondente.

Projeto da iteração

Projeto de sistemas com diagramas UML, de acordo com os requisitos da iteração em desenvolvimento.

Desenvolvimento da iteração

Codificação dos requisitos selecionados para a iteração.

Escrita dos testes de unidade

Escrever testes de unidade antes da codificação.

Escrita dos testes de aceitação

Testes de aceitação também devem ser escritos antes da codificação.

Desenvolvimento coletivo do código

“Toda a equipe fica responsável pelo desenvolvimento, portanto é necessário que todos entendam e conheçam o que está sendo feito. As partes envolvidas são os programadores.” [SOUZA, 2012].

Programação em pares

A programação em pares sugere que um programador escreva o código, enquanto o outro observa e sugere modificações.

Refatoração

Refatorar o código toda vez que a equipe julgar necessário.

Programação lado-a-lado

Colocar os programadores para trabalharem lado-a-lado no ambiente de trabalho, é uma forma de melhorar o relacionamento entre os membros da equipe.

3.1.2.4. Atividade validação de incremento

Inspeção do código

Verificar o código fonte, a fim de encontrar possível erros inseridos durante o processo de codificação.

3.1.2.5. Atividade integração do incremento

Reunião de revisão da integração

Ao final de cada iteração, realizar uma reunião para discutir toda a etapa de integração que foi concluída.

3.1.2.6. Atividade sistema final

Geração de uma breve documentação

Uma breve descrição do produto, para ser entregue ao cliente. Necessário caso uma documentação não tenha sido gerada ao longo do processo de desenvolvimento.

Entrega do sistema ao cliente

Entrega do produto ao cliente, após a conclusão de todas as etapas.

4. Android

Segundo o site oficial do projeto Android [ANDROID, 2013], o “Android é um software open-source criado para telefones móveis e outros dispositivos”. O projeto é liderado pela *Google*, que fica encarregada pelo desenvolvimento e manutenção do Android.

O Android teve sua origem através de um grupo de empresas chamado *Open Handset Alliance* (OHA), liderado pela *Google*. O Projeto Android tem como objetivo principal “criar um produto real e de sucesso, que melhore a experiência de usuários finais com dispositivos móveis” [ANDROID, 2013]. Para alcançar tal objetivo, o projeto Android quer assegurar a existência de uma plataforma aberta onde operadoras, fabricante de equipamentos e desenvolvedores possam transformar suas ideias inovadoras em realidade.

O Android é intencionalmente *open-source*, dessa forma as organizações que compartilham necessidades em comum, podem agrupar recursos para contribuir na implementação de um único produto compartilhado que cada colaborador possa adaptar e customizar [ANDROID, 2013]. Apesar de receber varias contribuições, o Android é visto como um produto único e integral, e não uma distribuição, especificação ou coleção de partes reutilizáveis.

Nesse trabalho está sendo proposta uma LPS para a plataforma Android, para tanto será abordado somente os aspectos relevantes para o desenvolvimento de aplicativos, desde a SDK até os padrões de interface determinados pelo Projeto Android.

4.1. Informações Técnicas

4.1.1. Dalvik Virtual Machine

Dalvik é a máquina virtual originalmente criada especificamente para o projeto Android. Ao desenvolver as aplicações para Android, é utilizada “a linguagem Java e

todos os seus recursos normalmente, mas depois que o *bytecode* (.class) é compilado e convertido para o formato .dex (*Dalvik Executable*), que representa a aplicação Android compilada” [LECHETA, 2010 - p. 24]

Da mesma forma que os programas escritos em Java são compilados para rodar sobre a máquina virtual Java (JVM), as aplicações Android são compiladas para rodar sobre a máquina virtual Dalvik.

4.1.2. Informações Técnicas de Segurança

Aplicativos Android fazem uso avançado de software e hardware, bem como dos dados locais e do servidor. Para proteger esses dados, a plataforma precisa fornecer um ambiente que assegure a segurança dos usuários, dos dados, dos aplicativos, da rede e do dispositivo.

O Android foi planejado com múltiplas camadas de segurança que fornece a flexibilidade exigida por uma plataforma aberta. O planejamento de segurança do Android, leva em consideração que os invasores podem utilizar técnicas comuns de ataque, como engenharia social (convencer o usuário a instalar *malwares*) e ataques a aplicativos de terceiros [ANDROID, 2013].

O Android é baseado no sistema operacional *Linux*, dessa forma a plataforma Android provê a segurança do *kernel do Linux*, bem como uma segurança na comunicação entre processos (IPC – *inter-process communication*) facilitada para permitir a comunicação entre aplicações rodando em diferentes processos.

O Android toma vantagem da proteção do Linux baseada em usuário, como um meio para identificar e isolar os recursos das aplicações. O sistema Android assimila um ID único de usuário (UID) para cada aplicação e roda como um processo separado. Essa abordagem é diferente de outros sistemas operacionais, onde várias aplicações rodam com a mesma permissão de usuário.

Dessa forma o *kernel* reforça a segurança entre as aplicações em nível de processos. Por padrão, as aplicações não podem interagir entre elas e tem acesso limitado ao sistema operacional.

4.2. Android SDK

O Android SDK (*Software Development Kit*) prove a API e as ferramentas necessárias para construir, testar e depurar aplicativos para Android. O Android SDK é composto de pacotes modulares que podem ser baixados separadamente através do *Android SDK Manager*.

Existem diferentes pacotes disponíveis para o Android SDK, abaixo serão descritos os principais pacotes [ANDROID, 2013]:

4.2.1. SDK Tools

Contém ferramentas para depuração e testes, além de outros utilitários necessários para desenvolver um aplicativo Android.

4.2.2. SDK Platform

Existe uma plataforma SDK disponível para cada versão do Android. Ela inclui um arquivo *android.jar* com as bibliotecas Androids compatíveis. Para construir um aplicativo Android é necessário especificar qual plataforma SDK deverá ser utilizada.

4.2.3. Imagens do Sistema

Cada versão da plataforma oferece uma ou mais imagens do sistema. O emulador Android requer uma imagem de sistema para operar. Os aplicativos desenvolvidos devem ser testados na última versão da plataforma, utilizar o emulador com a última versão da imagem do sistema é a forma recomendada para realizar esse teste.

4.2.4. Google API

É uma extensão para a SDK que prove uma plataforma para o desenvolvimento de aplicativos que utilizam a API Google e uma imagem de sistema para o emulador, assim é possível testar o aplicativo utilizando a API Google.

4.2.5. Android Support

Uma biblioteca estática que pode ser incluída no código da aplicação para aproveitar todos os recursos da API, que não estão disponíveis na plataforma padrão.

4.3. Android Design

Para auxiliar no design de aplicativos Android, a Google lançou um guia de estilos para os desenvolvedores [DESIGN, 2013]. O objetivo desse guia é tornar mais bonita, iterativa e uniforme a interface dos aplicativos desenvolvidos.

O design do Android é focado em três grandes objetivos, que devem ser levados em consideração pelos desenvolvedores no momento de criar novos aplicativos para a plataforma. Esses objetivos são encantar o usuário, simplificar a vida do usuário e ser impressionante. A Tabela 1 aponta os principais aspectos a serem analisados dentro de cada um desses objetivos:

Tabela 1 – Princípios para o desenvolvimento de interfaces gráficas para Android

Encantar o	Encantar de forma surpreendente	Uma interface bonita, uma animação cuidadosamente colocada ou um efeito sonoro no tempo certo é uma forma de tornar mais agradável a experiência do usuário.
-------------------	---------------------------------	--

	Objetos reais ao invés de botões e menus	Permitir que as pessoas manipulem e toquem diretamente os objetos na aplicação. Isso reduz o esforço cognitivo em realizar uma tarefa e torna ela emocionalmente mais satisfatória.
	Permitir customizar	As pessoas adoram adicionar toques pessoais que a fazem se sentirem mais confortáveis e no controle das ferramentas. Fornecer padrões bonitos para as telas, mas considerar customizações opcionais que não dificultam as tarefas primárias.
	Conhecer o usuário	Aprender as preferências do usuário ao longo do tempo, ao invés de pedir que ele tome as mesmas decisões repetidas vezes. Colocar escolhas anteriores de fácil acesso.
Simplificar a vida do usuário	Ser breve	Usar frases curtas e palavras simples. Os usuários costumam ignorar as frases que são longas.
	Imagens são melhores que palavras	Considerar a utilização de imagens para transmitir ideias, elas prendem mais a atenção das pessoas e podem ser muito mais eficientes que palavras.

	Ajudar o usuário a se decidir	O aplicativo deve considerar o seu melhor palpite, baseado nas experiências do usuário e agir segundo ele. Muitas escolhas e decisões desagradam o usuário. Apenas no caso de estar errado, habilitar a opção de 'desfazer' para o usuário.
	Simplificar as opções	Mostrar apenas o que o usuário necessita, no momento em que ele necessitar. Os usuários ficam sobrecarregados quando veem muitas opções ao mesmo tempo, informações e tarefas devem ser quebradas em partes menores. É recomendado que sejam escondidas as opções que não são essenciais no momento e instruir o usuário pelo caminho que ele deve seguir.
	Permitir que o usuário saiba onde está	Permitir que o usuário saiba o caminho de volta, fazer com que as telas da aplicação se pareçam diferentes entre si e usem transições para mostrar a relação entre elas. Fornecer feedback das tarefas em progresso.

	Nunca perder o conteúdo do usuário	É importante salvar aquilo que usuário levou tempo para criar e permitir que isso seja acessado de qualquer lugar. Guardar configurações, toques pessoais e criações em telefones, <i>tablets</i> e computadores. Isso torna muito mais fácil as operações de <i>upgrade</i> .
	Se forem iguais, devem ter a mesma função	Auxiliar o usuário a distinguir funções diferentes fazendo com que elas sejam diferentes visualmente. Devem-se evitar telas que sejam parecidas e que executem funções diferentes para uma mesma entrada.
	Só interromper se for importante	O usuário pretende permanecer focado, e a menos que seja crítico e urgente, uma interrupção pode ser frustrante. Como um bom assistente pessoal, o aplicativo deve proteger o usuário de notificações sem importância.
Ser impressionante	Truques que funcionam em qualquer lugar	O usuário sente-se bem quando consegue executar as coisas por si mesmo. Deve-se desenvolver a aplicação para que seja fácil de aprender trazendo padrões visuais e memória muscular de outros aplicativos Android. O gesto de arrastar pode ser um bom atalho para navegação, por exemplo.

	Instruções de forma clara	O aplicativo deve ser gentil na forma como leva o usuário a realizar as correções. O usuário quer se sentir inteligente enquanto usa o aplicativo. Se algo ocorrer errado, devem ser fornecidas informações claras para recuperação, mas poupando o usuário dos detalhes técnicos. Se for possível consertar o problema sem a interferência do usuário, será bem melhor.
	Encorajar o usuário	As tarefas complexas devem ser quebradas em pequenos passos que possam ser compreendidos facilmente. Dar <i>feedback</i> das ações, mesmo que sejam uma sutil iluminação na tela.
	Realizar o “serviço pesado”	Fazer com que os usuários novatos se sintam experientes, permitindo que façam coisas que não acreditavam serem capazes de realizar. Um atalho que combina múltiplos efeitos em fotos pode fazer um amador em fotografias se sentir maravilhado, por exemplo.

	Tornar rápido o que for mais importante	Nem todas as ações são iguais. Decidir quais ações são as mais importantes na aplicação e torná-las fáceis e rápidas de utilizar, como o botão de tirar fotos em uma câmera ou o botão de <i>pause</i> em um tocador de músicas.
--	--	---

5. Linha de Produtos para Android para acesso às informações das ferramentas de apoio a métodos ágeis

Neste capítulo será descrito o desenvolvimento da LPS proposta neste trabalho, as práticas selecionadas, as tecnologias envolvidas e a arquitetura da LPS.

5.1. Práticas ágeis para o desenvolvimento da LPS

No estudo realizado em [SOUZA, 2012] é proposto um framework para seleção de práticas ágeis que atividades e práticas para o desenvolvimento de uma LPS. O framework sugere várias práticas a serem adotadas na definição de um processo para a Engenharia de Domínio e Engenharia de Aplicação.

Abaixo estão listadas as práticas sugeridas pelo framework e assinaladas com “X” as práticas que serão utilizadas no desenvolvimento da LPS proposta neste trabalho.

Engenharia de Domínio	Lista de requisitos de cada aplicação (obrigatório)	X
	Modelo de features (obrigatório)	X
	Documentação	
	Projeto da arquitetura do sistema (obrigatório)	X
	Representação da arquitetura de domínio (obrigatório)	X
	Planejamento da iteração (obrigatório)	X
	Duração da iteração (obrigatório)	X
	Distribuição dos módulos para os responsáveis	
	Reunião diária	
	Projeto da iteração	
	Desenvolvimento da iteração (obrigatório)	X
	Escrita dos testes de unidades	X
	Escrita dos testes de aceitação	
	Desenvolvimento coletivo do código	
	Programação em pares	
	Refatoração	
	Programação lado-a-lado	
	Inspeção do código	X

Engenharia de Aplicação	Lista de requisitos (Obrigatório)	X
	Seleção dos componentes (Obrigatório)	X
	Modelo geral	
	Documentação Inicial	
	Planejamento da iteração (Obrigatório)	X
	Duração de iterações (Obrigatório)	X
	Distribuição de requisitos para os responsáveis (Obrigatório)	X
	Reuniões diárias	
	Estórias do usuário	
	Casos de uso	
	Projeto da iteração	
	Desenvolvimento da iteração (Obrigatório)	X
	Escrita dos testes de unidade	
	Escrita dos testes de aceitação	
	Desenvolvimento coletivo do código	
	Programação em pares	
	Refatoração	
	Programação lado-a-lado	
	Inspeção de código	X
	Reunião de revisão da integração	X
	Geração de uma breve documentação	
	Entrega do sistema ao cliente	X

5.2. Engenharia de Domínio

5.2.1. Lista de Requisitos

Abaixo são listados os requisitos selecionados para a LPS proposta nesse trabalho:

- Product backlog (comum) – Exibe o *Backlog* de um produto, uma lista com o título das estórias de usuário selecionadas para o *Product Backlog*.
- Detalhamento de estória do backlog (comum) – Detalhe de uma estória de usuário. Assim, ao selecionar uma estória do backlog, será possível visualizar os seus detalhes, os detalhes da estória são título, descrição e solicitador.

- Pesquisa de histórias do backlog (comum) – Permite encontrar uma história no Product Backlog, pesquisando por parte do texto de sua descrição ou título.
- Andamento da iteração atual – Permite acompanhar o andamento dos tickets dentro do taskboard, classificado de acordo com o seu status (*Todo, Doing, Done*);
- Pesquisa de tickets do taskboard – Possibilidade de pesquisar um ticket do taskboard, através de parte do seu texto ou título.
- Detalhamento de ticket – Permite selecionar um ticket do taskboard para visualizar seus detalhes. Os detalhes são título, descrição, status e esforço estimado.
- Pesquisa de história do taskboard – Possibilidade de pesquisar uma história do taskboard, por parte de seu texto ou título.
- Detalhamento de história – Possibilidade de selecionar uma história do Taskboard para exibir seus detalhes (que são título, descrição e solicitador).
- Detalhes da iteração atual – Possibilidade de visualizar os detalhes da iteração atual, os detalhes são título, objetivo, data de início e data fim.
- Backlog da iteração atual – Lista das histórias selecionadas para a iteração atual.
- Possibilidade de comunicação com os sistemas *Pronto*, *TinyPM* e *AgileFant*.
- Interface Gráfica para dispositivo Android.

5.2.2. Modelo do Domínio

Na Figura 6, é apresentado o modelo do domínio elaborado para a LPS proposta nesse trabalho:

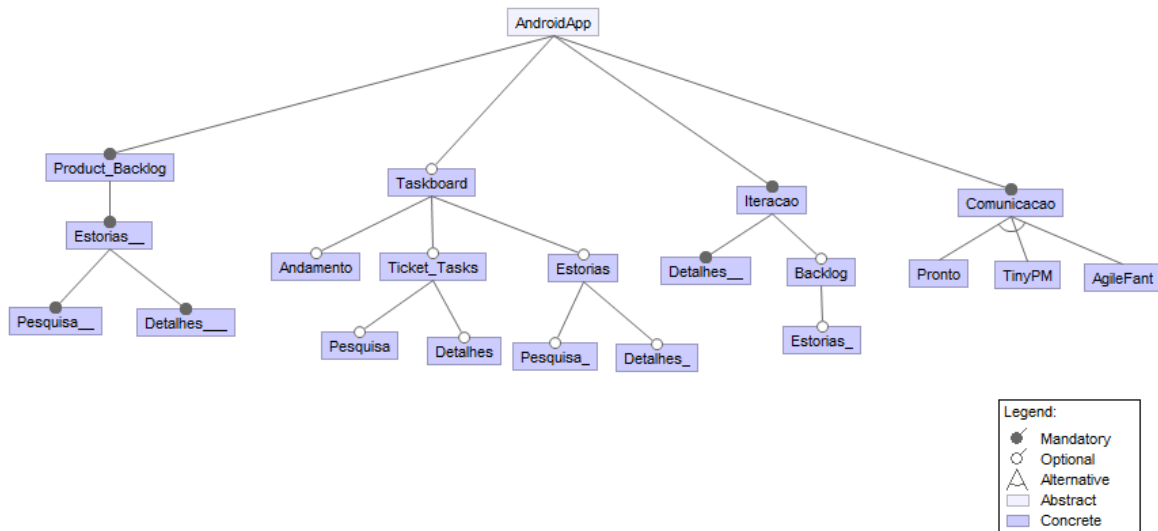


Figura 6 - Modelo do domínio

5.2.3. Projeto de Componentes

Componentes para os pontos comuns da LPS:

- Interface Gráfica;
- Product Backlog;
- Pesquisa de estórias do Product Backlog;
- Detalhes de estórias do Product Backlog;
- Detalhes da Iteração Atual.

Componentes que farão parte dos pontos de variação da Linha de Produtos:

- Taskboard;
- Andamento dos Tickets;
- Pesquisa dos Tickets do Taskboard;
- Detalhe dos Tickets do Taskboard;

- Backlog da Iteração;
- Comunicação.

5.2.4. Representação da Arquitetura

A Figura 7 apresenta a representação da arquitetura para desenvolvimento da LPS proposta nesse trabalho:

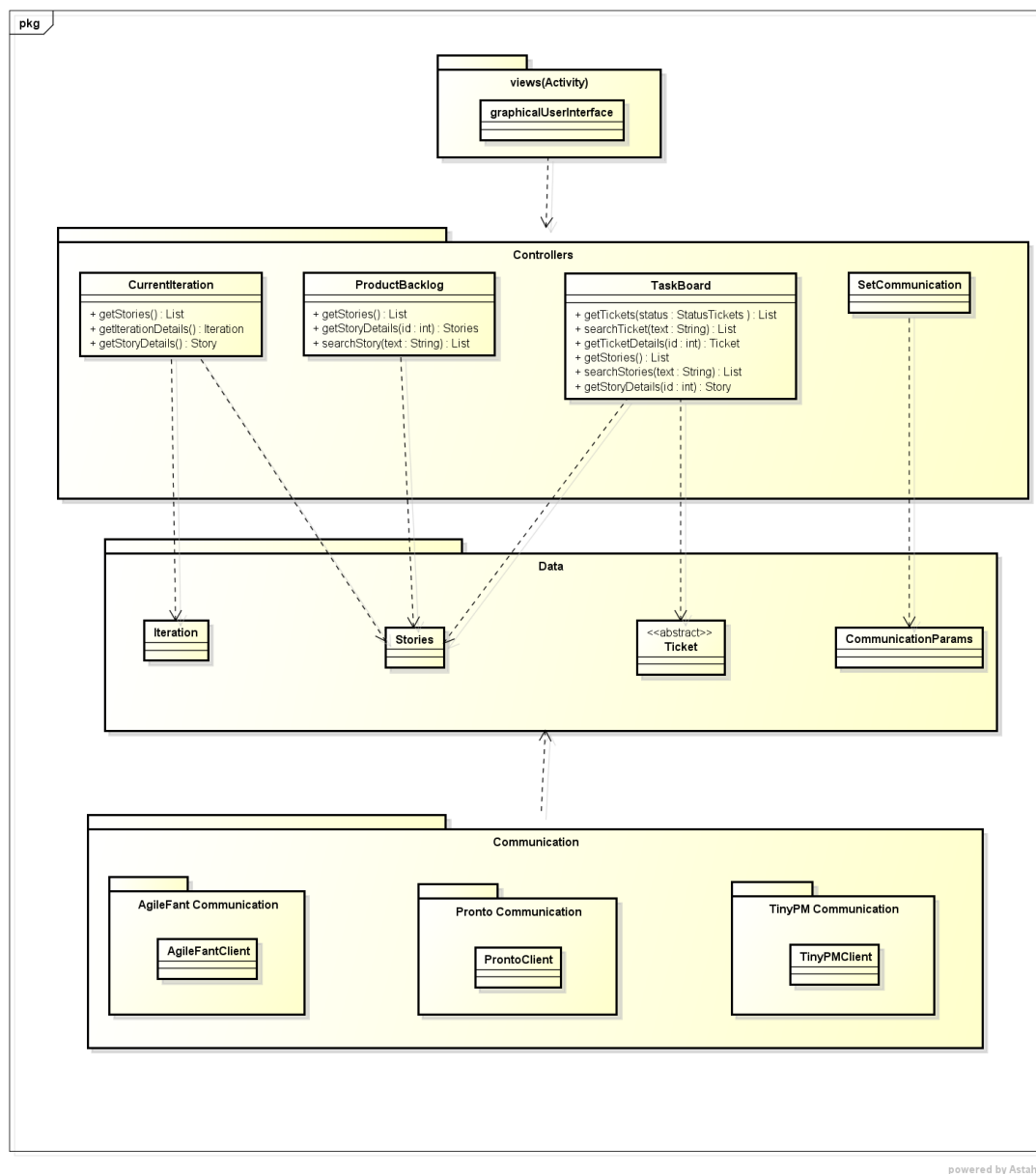


Figura 7 - Representação da arquitetura

5.2.5. Iteração 1

Planejamento da Iteração

O objetivo dessa iteração é desenvolver uma classe para comunicação com a API de cada uma das ferramentas *TinyPM*, *Pronto* e *AgileFant*. Cada classe obterá um json com os dados requisitados e armazenará esses dados em um banco de dados SQLite local. Os dados retornados serão as informações relacionadas à iteração atual, ao *taskboard* e ao *product backlog*.

Posteriormente em outra iteração, serão desenvolvidos os controllers que acessarão o banco de dados SQLite e fornecerão os dados para a interface gráfica.

Duração da iteração

Essa iteração teve a duração de uma semana.

Desenvolvimento da iteração

Para essa iteração foram desenvolvidas as classes de comunicação apresentadas na Figura 8:



Figura 8 – Classes de comunicação

As classes de comunicação apresentam os métodos `makeRequest` e `readStream`, eles são responsáveis pela requisição dos dados para a ferramenta web e a transformação dos dados. Essas classes também contêm os métodos para requisição dos dados, de acordo com os dados suportados por cada linha de produtos, que obtêm os dados transformados e fazem a gravação deles no banco SQLite local.

Para manipulação dos dados no banco SQLite local, foram desenvolvidas as classes apresentadas na Figura 9. A estrutura é composta por uma classe denominada `ApptccSQLiteOpenHelper`, que faz a criação das tabelas na primeira vez que o banco for requisitado. Além dela, existe uma classe para manipulação de cada uma das tabelas, essas classes são responsáveis pela persistência e consulta dos dados.

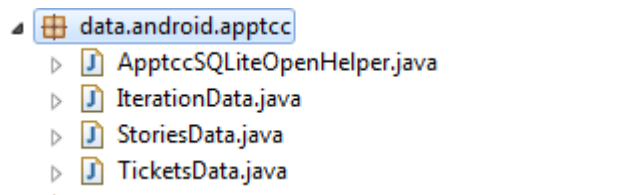


Figura 9 – Classes para manipulação dos dados

Escrita dos testes de unidades

Foram escritos, antes da codificação, testes de unidade para verificar cada uma das classes de comunicação. O teste consiste em instanciar a classe de comunicação e chamar os métodos para requisição dos dados, na sequência são instanciadas as classes de manipulação dos dados e realizada uma consulta. Essa sequência de testes garante que a comunicação está sendo realizada e que os dados estão sendo persistidos no banco local SQLite.

A Figura 10 mostra um método de teste da classe *TinyPM*. Este teste é bem semelhante com os testes das demais classes.

```
public class TestTinyPMCommunication extends TestCase {  
    TinyPM tiny = new TinyPM();  
    public void testRequestIteration() {  
        tiny.requestIterationData();  
        IterationData iteration = new IterationData();  
        assertEquals(new Iteration(), iteration.getCurrentIterationData());  
    }  
}
```

Figura 10 – Exemplo de teste de comunicação

Inspeção do código

Foi realizado o processo de inspeção do código, a fim de identificar variáveis declaradas e não utilizadas, importações de classes não utilizadas e possíveis erros de gramática em comentários.

5.2.6. Iteração 2

Planejamento da Iteração

Nessa iteração foram desenvolvidas as camadas *controller* e *view*. Foi realizada a Codificação dos objetos *CurrentIterations*, *ProductBacklog* e *Taskboard*. Desenvolvimento de interface gráfica que respeite o guia de estilos estipulado pela Google, descritos no capítulo 4.3 desse trabalho.

Duração da iteração

Essa iteração teve a duração de duas semanas.

Desenvolvimento da iteração

Foram desenvolvidas as classes apresentadas na Figura 11:

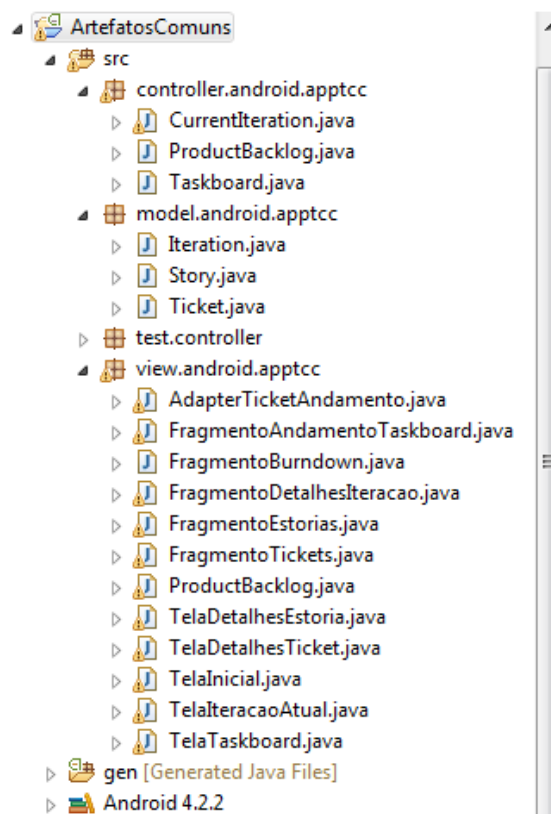


Figura 11 - Classes produzidas na Iteração 2

Foram desenvolvidos os controladores para a LPS, responsáveis por acessar os dados nas tabelas temporárias da aplicação e fornecê-los à interface gráfica. Cada controlador representa uma *feature* do modelo de *features*, cada método do controlador representa uma ação que pode ser realizada com a *feature* relacionada. Dessa forma, garante-se que o modelo de *features* será atendido pela LPS desenvolvida.

Também foi desenvolvida a interface gráfica para comunicação com o usuário seguindo o guia de estilos determinado pela *Google*. Para a tela inicial foram desenvolvidos botões que dão acesso direto às principais funções do aplicativo. Seguindo o princípio de simplificar as opções, essa organização também permite ao usuário intuitivamente encontrar as opções que procura, mesmo que acessando o aplicativo pela primeira vez. Ao entrar em uma das funcionalidades listadas na tela inicial, o usuário encontra a navegação por abas com as opções relativas à funcionalidade selecionada. A navegação em abas permite que o usuário saiba onde está e se localize mais facilmente no aplicativo.

Na Figura 12 está a mostrada a tela inicial e a tela com as informações do *taskboard*, como exemplo:

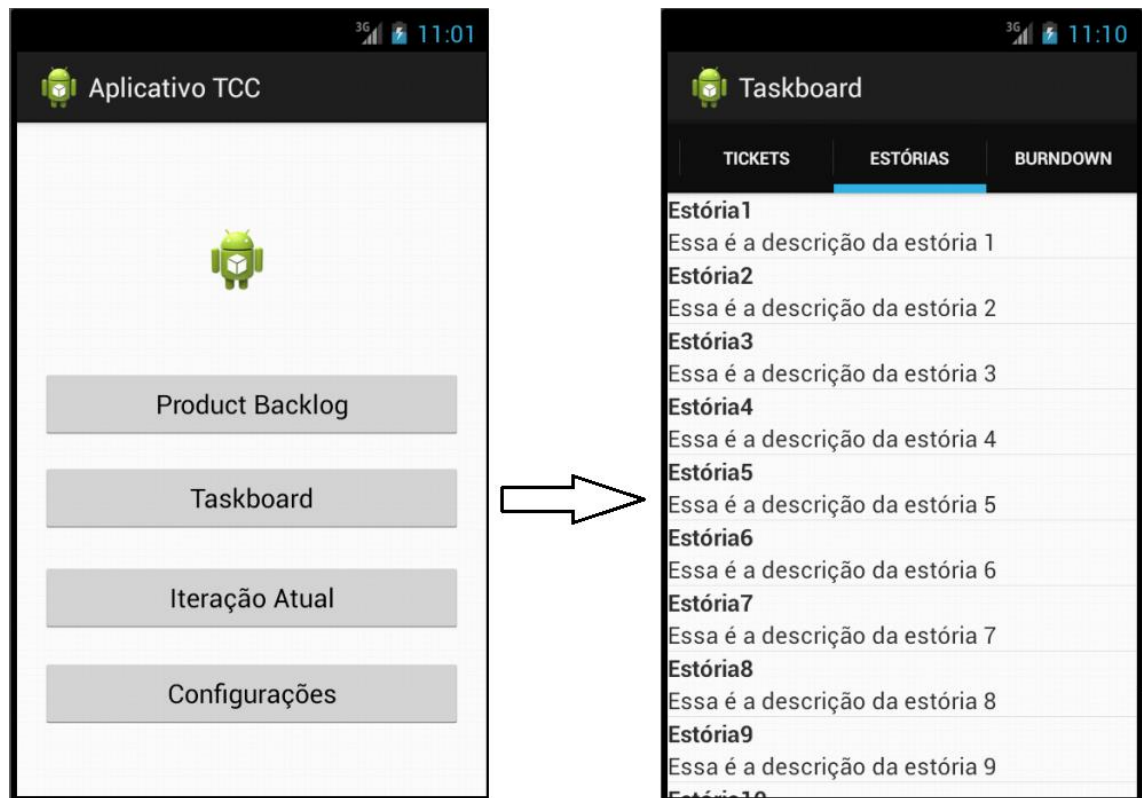


Figura 12 - Interface gráfica produzida na Iteração 2

Escrita dos testes de unidades

Foram desenvolvidos testes de unidades para os métodos das classes controladoras. Antes da implementação de cada controlador, foi elaborado um teste unitário que verificasse os métodos previstos na arquitetura do domínio para aquele controlador. Na Figura 13 estão relacionados as classes de testes unitários, com os respectivos métodos de testes.

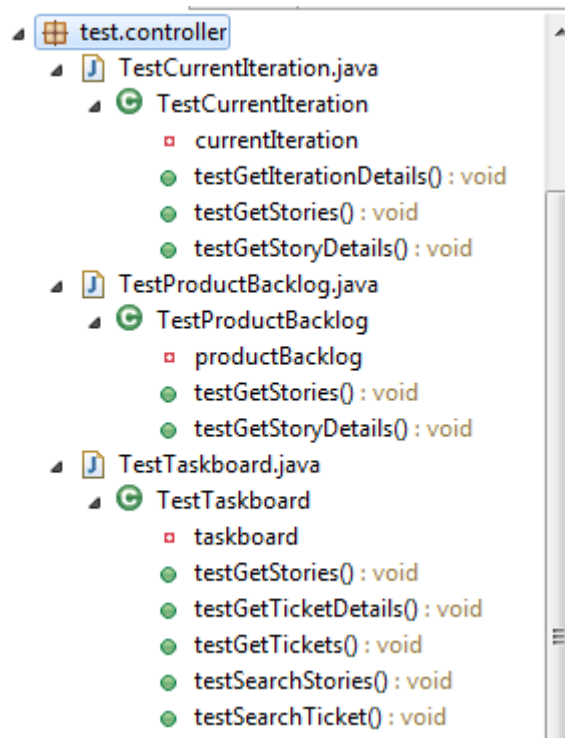


Figura 13 - Testes produzidos na Iteração 2

A Figura 14 mostra o exemplo de um teste escrito para testar o controlador CurrrentIteration. O teste apresentado verifica se todos os objetos retornados pelo método getStories são do tipo Story:

```
public class TestCurrentIteration extends TestCase {
    private CurrentIteration currentIteration = new CurrentIteration();

    public void testGetStories() {
        // Testar se todos os resultados são Stories
        boolean result = true;
        List<Story> stories = currentIteration.getStories();

        for (Story s : stories) {
            if (!s.getClass().equals(new Story().getClass())) {
                result = false;
            }
        }
        assertTrue(result);
    }
}
```

Figura 14 – Exemplo de teste de unidade

Inspeção do código

Foi realizado o processo de inspeção do código, a fim de identificar variáveis declaradas e não utilizadas, importações de classes não utilizadas e possíveis erros de gramática nos comentários do código fonte. Foram removidos comentários gerados pela IDE Eclipse que não fossem de conteúdo relevante. Foi revisada também a indentação do código fonte.

5.3. Engenharia de Aplicação

Nesta etapa serão desenvolvidos três produtos da LPS: AppTCCPronto, o AppTCCAgilefant e o AppTCCTinyPM. Cada produto será responsável pela comunicação com cada uma das ferramentas web para gerenciamento de projetos ágeis, selecionadas para esse trabalho.

Abaixo, serão descritos os processos para desenvolvimento de cada um desses produtos.

5.3.1. Primeiro produto – AppTCCPronto

O primeiro produto a ser desenvolvido, foi o aplicativo Android para comunicação com a ferramenta web Pronto, chamado de AppTCCPronto. Abaixo segue a lista de requisitos e os componentes selecionados para o desenvolvimento do produto.

Lista de requisitos

Abaixo estão listados os requisitos selecionados para esse produto:

- Product backlog – Exibe o *Backlog* de um produto, uma lista com o título das histórias de usuário selecionadas para o *Product Backlog*.

- Detalhamento de estória do backlog – Detalhe de uma estória de usuário. Assim, ao selecionar uma estória do backlog, será possível visualizar o seu título, descrição e solicitador.
- Pesquisa de estórias do backlog – Permite encontrar uma estória no Product Backlog, pesquisando por parte do texto de sua descrição ou título.
- Andamento da iteração atual – Permite acompanhar o andamento dos tickets dentro do taskboard, classificado de acordo com o seu status (*Todo, Doing, Done*);
- Pesquisa de tickets do taskboard – Possibilidade de pesquisar um ticket do taskboard, através de parte do seu texto ou título.
- Detalhamento de ticket – Permite selecionar um ticket do taskboard para visualizar seus detalhes (título, descrição, status e esforço estimado).
- Pesquisa de estória do taskboard – Possibilidade de pesquisar uma estória do taskboard, por parte de seu texto ou título.
- Detalhamento de estória – Possibilidade de selecionar uma estória do *taskboard* para exibir seus detalhes que são título, descrição e solicitador.
- Detalhes da iteração atual – Possibilidade de visualizar os detalhes da iteração atual: título, objetivo, data de início e data fim.
- Backlog da iteração atual – Lista das estórias selecionadas para a iteração atual.
- Comunicação com o sistema *Pronto*.
- Interface Gráfica para dispositivo Android.

Seleção dos componentes

Os seguintes componentes foram selecionados para o desenvolvimento desse produto:

- Interface Gráfica;

- Product Backlog;
- Pesquisa de histórias do Product Backlog;
- Detalhes de histórias do Product Backlog;
- Detalhes da Iteração Atual;
- Taskboard;
- Andamento dos Tickets;
- Pesquisa dos Tickets do Taskboard;
- Detalhe dos Tickets do Taskboard;
- Backlog da Iteração;
- Comunicação com o sistema *Pronto*.

Modelo de domínio

Na Figura 15 é apresentado o modelo de domínio do produto desenvolvido nessa etapa:

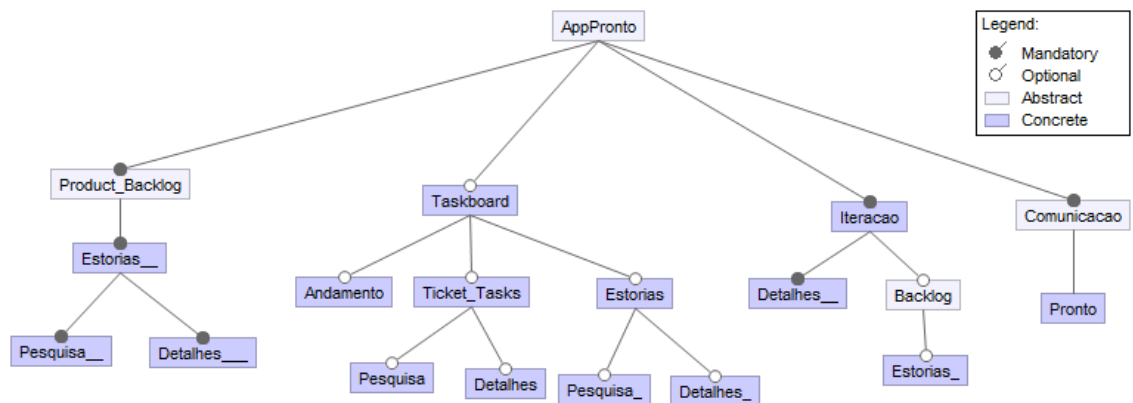


Figura 15 – Modelo de domínio do Aplicativo AppTCCPronto

5.3.1.1. Iteração 1

Durante essa iteração serão selecionados os componentes para desenvolvimento do aplicativo para comunicação com a ferramenta web Pronto.

Planejamento das iterações

O desenvolvimento desse produto foi realizado em uma única iteração. Nessa iteração foram agrupados os componentes selecionados que atendem os requisitos do produto.

Foi desenvolvida uma interface gráfica para o aplicativo, com base na interface geral elaborada para a LPS, ocultando as telas que não tem relação com os requisitos selecionados para esse produto.

Duração da iteração

Essa iteração teve a duração de uma semana.

Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento dos requisitos foi o autor desse trabalho.

Desenvolvimento da iteração

O desenvolvimento dessa iteração produziu as classes listadas na Figura 16. O principal ponto de variação desse produto é a inexistência do *burndown*, uma vez que a ferramenta não fornece controle de esforço estimado para os tickets.

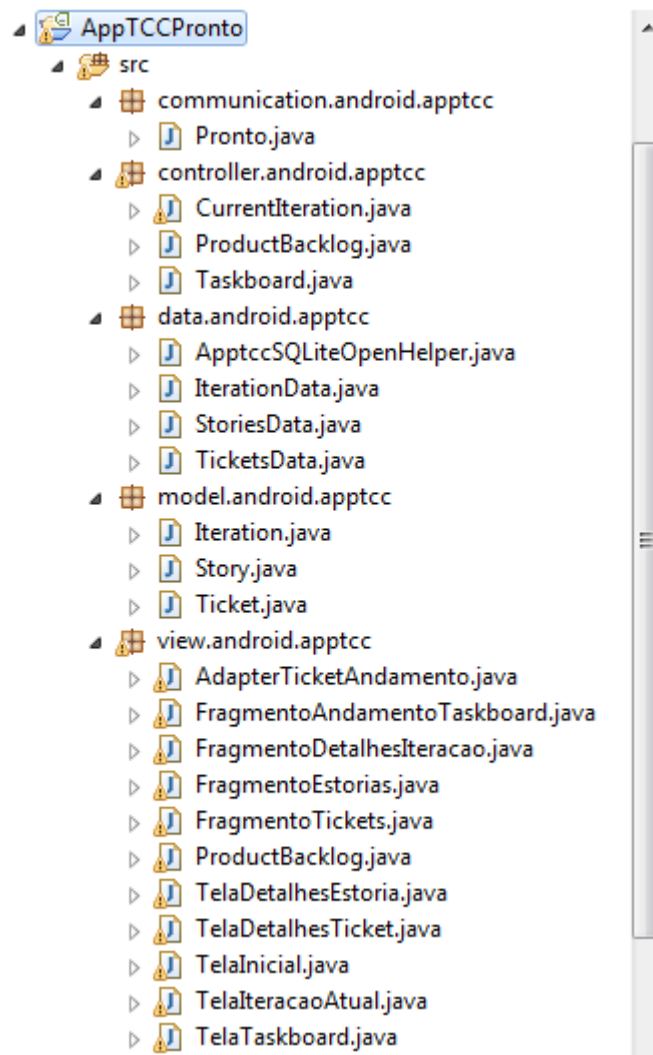


Figura 16 – Classes do Aplicativo AppTCCPronto

Na Figura 17, destaque para as abas da tela Taskboard sem a presença da opção *Burndown*, principal ponto de variação desse produto da LPS:

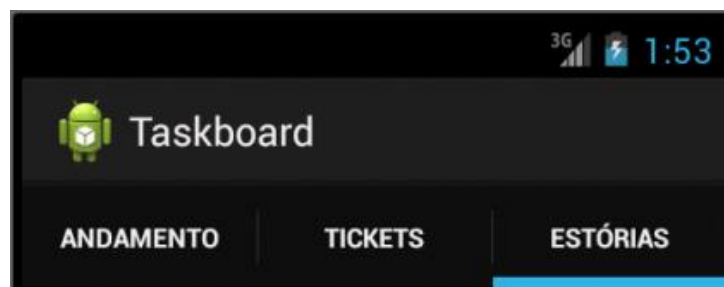


Figura 17 – Abas da tela Taskboard do Aplicativo AppTCCPronto

Inspeção de código

Foi realizada a inspeção do código para correções e remoção de partes que não fossem relevantes para o funcionamento do produto. Foi verificada a existência de erros de grafia em comentários.

5.3.2. Segundo produto – *AppTCCAgilefant*

O segundo produto é um aplicativo Android para comunicação com a ferramenta web *Agilefant*, chamado de *AppTCCAgilefant*. Abaixo seguem a lista de requisitos e os componentes selecionados para o desenvolvimento do produto.

Lista de requisitos

Abaixo estão listados os requisitos e componentes selecionados para o desenvolvimento desse produto.

- Product backlog – Exibe o *Backlog* de um produto, uma lista com o título das histórias de usuário selecionadas para o *Product Backlog*.
- Detalhamento de história do backlog – Detalhe de uma história de usuário. Assim, ao selecionar uma história do backlog, será possível visualizar os seus detalhes: título, descrição e solicitador.
- Pesquisa de histórias do backlog – Permite encontrar uma história no Product Backlog, pesquisando por parte do texto de sua descrição ou título.
- Detalhes da iteração atual – Possibilidade de visualizar os detalhes da iteração atual, os detalhes são título, objetivo, data de início e data fim.
- Comunicação com o sistema *AgileFant*.
- Interface Gráfica para dispositivo Android.

Seleção dos componentes

Os seguintes componentes foram selecionados para o desenvolvimento desse produto:

- Interface Gráfica;
- Product Backlog;
- Pesquisa de histórias do Product Backlog;
- Detalhes de histórias do Product Backlog;
- Detalhes da Iteração Atual;
- Comunicação.

Modelo de domínio

Na Figura 18 é apresentado o modelo de domínio do Aplicativo AppTCCAgilefant, esse é o mais simples dos produtos da LP, a principal variação é a inexistência do *taskboard* que não é suportado pela ferramenta web Agilefant.

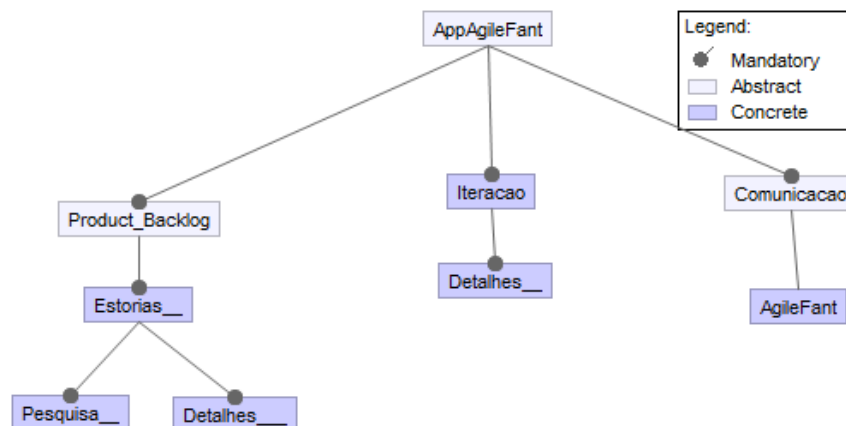


Figura 18 – Modelo de domínio do Aplicativo AppTCCAgilefant

5.3.2.1. Iteração 1

Planejamento da iteração

O desenvolvimento desse produto foi realizado em uma única iteração. Nessa iteração foram agrupados os componentes selecionados que atendem os requisitos do produto.

Foi desenvolvida uma interface gráfica para o aplicativo, com base na interface geral elaborada para a LPS, ocultando as telas que não tem relação com os requisitos selecionados para esse produto.

Duração da iteração

Essa iteração teve a duração de uma semana.

Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento dos requisitos foi o autor desse trabalho.

Desenvolvimento da iteração

O desenvolvimento dessa iteração resultou nas classes listadas na Figura 19. Destaque para inexistência dos componentes relacionados ao taskboard, sendo esse o principal ponto de variação desse produto.

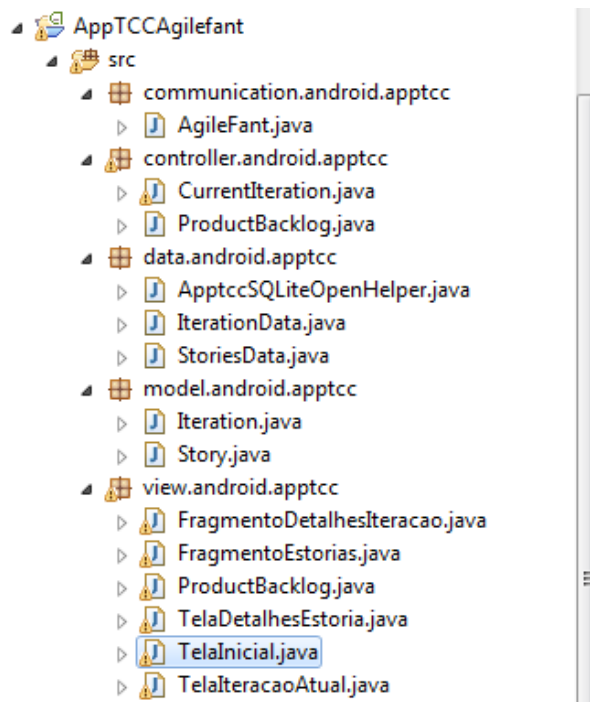


Figura 19 – Classes do Aplicativo AppTCCAgilefant

Na Figura 20 está representada a tela inicial do Aplicativo AppTCCAgilefant. A tela inicial provê acesso às opções de Product Backlog e Iteração atual, que são suportadas pela ferramenta web Agilefant.

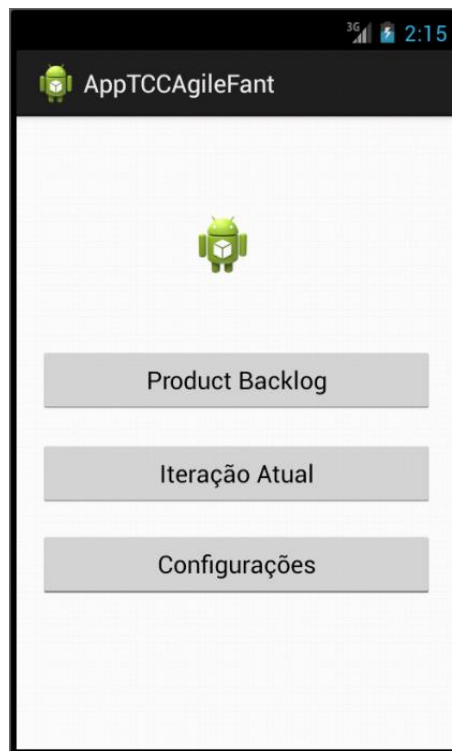


Figura 20 – Tela inicial do Apicativo AppTCCAgilefant

Inspeção de código

Foi realizada a inspeção do código para correções e remoção de partes que não fossem relevantes para o funcionamento do produto. Foi verificada a existência de erros de grafia em comentários.

5.3.3. Terceiro produto – *AppTCCTinyPM*

O terceiro produto da LPS é um aplicativo Android para comunicação com a ferramenta web TinyPM, chamado de AppTCCTinyPM. Esse é o mais completo dos produtos da LPS e utiliza todos os componentes dela.

Abaixo seguem a lista de requisitos e os componentes selecionados para o desenvolvimento do produto.

Lista de requisitos

Abaixo são listados os requisitos selecionados para a linha de produtos proposta nesse trabalho:

- Product backlog – Exibe o *Backlog* de um produto, uma lista com o título das histórias de usuário selecionadas para o *Product Backlog*.
- Detalhamento de história do backlog – Detalhe de uma história de usuário. Assim, ao selecionar uma história do backlog, será possível visualizar os seus detalhes: título, descrição e solicitador.
- Pesquisa de histórias do backlog – Permite encontrar uma história no Product Backlog, pesquisando por parte do texto de sua descrição ou título.
- Andamento da iteração atual – Permite acompanhar o andamento dos tickets dentro do taskboard, classificado de acordo com o seu status (*Todo*, *Doing*, *Done*);
- Pesquisa de tickets do taskboard – Possibilidade de pesquisar um ticket do taskboard, através de parte do seu texto ou título.
- Detalhamento de ticket – Permite selecionar um ticket do taskboard para visualizar seus detalhes: título, descrição, status e esforço estimado.
- Pesquisa de história do taskboard – Possibilidade de pesquisar uma história do taskboard, por parte de seu texto ou título.
- Detalhamento de história – Possibilidade de selecionar uma história do Taskboard para exibir seus detalhes que são título, descrição e solicitador.
- Detalhes da iteração atual – Possibilidade de visualizar os detalhes da iteração atual, os detalhes são título, objetivo, data de início e data fim.
- Backlog da iteração atual – Lista das histórias selecionadas para a iteração atual.
- Comunicação com o sistema *TinyPM*.
- Interface Gráfica para dispositivo Android.

Seleção dos componentes

Os seguintes componentes foram selecionados para o desenvolvimento desse produto:

- Interface Gráfica;
- Product Backlog;
- Pesquisa de histórias do Product Backlog;
- Detalhes de histórias do Product Backlog;
- Detalhes da Iteração Atual;
- Taskboard;
- Andamento dos Tickets;
- Pesquisa dos Tickets do Taskboard;
- Detalhe dos Tickets do Taskboard;
- Backlog da Iteração;
- Comunicação.

Modelo de domínio

Na Figura 21 é apresentado o modelo de domínio do Aplicativo AppTCCTinyPM. Esse é o mais completo dos produtos da LPS, por utilizar todos os componentes disponíveis para ela.

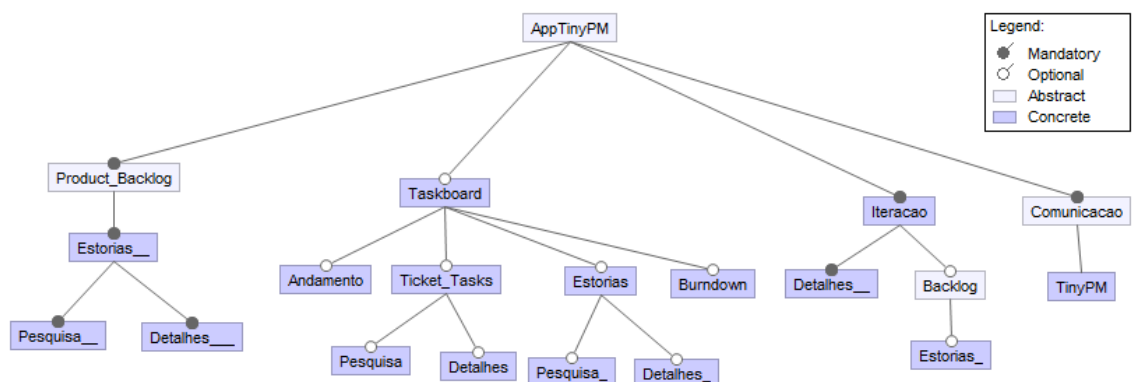


Figura 21 – Modelo de domínio do aplicativo AppTCCTinyPM

5.3.3.1. Iteração 1

Planejamento da iteração

O desenvolvimento desse produto foi realizado em uma única iteração. Nessa iteração foram agrupados os componentes selecionados que atendem os requisitos do produto.

Foi desenvolvida uma interface gráfica para o aplicativo, com base na interface geral elaborada para a LPS, ocultando as telas que não tem relação com os requisitos selecionados para esse produto.

Duração da iteração

Essa iteração teve a duração de uma semana.

Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento dos requisitos será o autor desse trabalho.

Desenvolvimento da iteração

Na Figura 21 são listadas as classes produzidas no desenvolvimento dessa iteração.

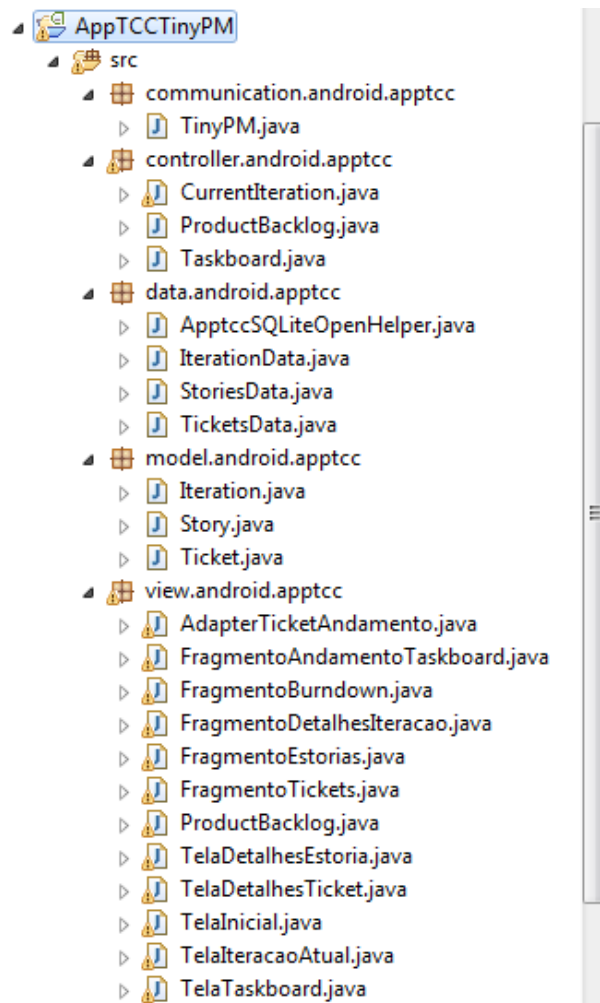


Figura 21 – Classes do Aplicativo AppTCCTinyPM

Na Figura 22 é apresentada a interface gráfica do Aplicativo AppTCCTinyPM, ela sofre pouca alteração em relação a interface gráfica padrão produzida na engenharia de domínio, isso se deve ao fato do Aplicativo AppTCCTinyPM suportar todos os componentes da LPS.

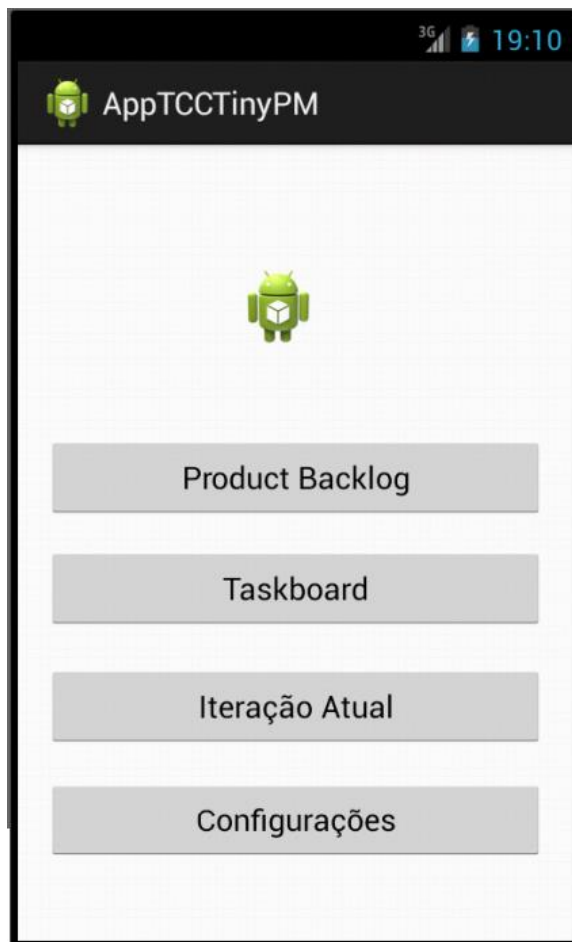


Figura 22 – Tela inicial do Aplicativo AppTCCTinyPM

Inspeção de código

Foi realizada a inspeção do código para correções e remoção de partes que não fossem relevantes para o funcionamento do produto. Foi verificada a existência de erros de grafia em comentários.

6. Conclusão

Esse trabalho resultou no desenvolvimento de uma LPS composta de três aplicativos para Android, onde cada aplicativo é responsável por acessar os dados gerados por três diferentes ferramentas web para gerência de equipes ágeis selecionadas. As ferramentas web selecionadas foram *Pronto*, *tinyPM* e *Agilefant*, por serem ferramentas de código aberto ou por possuírem uma versão gratuita que pudesse ser baixada e instalada em um servidor próprio.

A comunicação entre os aplicativos e as ferramentas web ocorreram com sucesso, permitindo que as informações geradas pelas ferramentas fossem apresentadas nos aplicativos desenvolvidos.

Apesar de semelhantes, as três ferramentas web possuem funcionalidades diferentes entre si, desse modo os aplicativos Android que se comunicam com elas compartilham funcionalidades comuns e pontos de variação entre si. Para atender o desenvolvimento desses aplicativos com pontos em comuns e de variação, foi utilizada uma LPS como mecanismo de desenvolvimento, dessa forma os pontos em comum puderam ser reaproveitados.

Uma experiência que vale ressaltar, foi que durante a modelagem da representação da arquitetura, percebeu-se que para cada *feature* do modelo de *features* poderia existir um controlador correspondente na representação da arquitetura, e para cada ação (opção) da *feature* poderia existir um método correspondente dentro do controlador. Dessa forma, a representação da arquitetura espelha o modelo de *features*, indicando como ele será desenvolvido e garantido que todo ele será atendido pela LPS.

O framework estendido para o desenvolvimento da LPS proposto em [SOUZA, 2013] foi de grande auxílio no desenvolvimento da LPS proposta nesse trabalho, facilitando que os objetivos do trabalho fossem atingidos. O desenvolvimento da LPS proposto através de iterações permitiu que pequenas entregas fossem

realizadas e validadas ao longo do processo de elaboração da mesma. A LPS desenvolvida nesse trabalho reforça a eficiência do framework estendido, servindo como um exemplo de sua aplicação e validação das práticas propostas.

Como trabalhos futuros relacionados a essa pesquisa, fica recomendado:

- Uma extensão dos aplicativos desenvolvidos nesse trabalho, com a possibilidade de executar edição dos dados fornecidos pelas ferramentas web para gerência de equipes ágeis, atualmente somente a consulta é realizada.
- Uma extensão da LPS, com o desenvolvimento de novos aplicativos que acessem os dados de outras ferramentas web para gerência de equipes ágeis, além das ferramentas aqui abordadas.
- Realizar um estudo de caso dos aplicativos desenvolvidos nesse trabalho, com a utilização deles por equipes que utilizem em seu dia-a-dia as ferramentas web selecionadas nesse trabalho.
- Desenvolvimento de uma LPS semelhante que rode sobre outras plataformas para dispositivos móveis, além da plataforma Android.

7. Referências

[SOARES, 2004] SOARES, Michel dos Santos. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. Disponível em: <<http://revistas.facecla.com.br/index.php/reinfo/article/view/146/38>> Acesso em 03 abr. 2012

[PRESSMAN, 2001] PRESSMAN, Roger S. "Software Engineering: A Practitioner's Approach". New York (NY): McGraw-Hill, 2001.

[MANIFESTO, 2012] Manifesto para o desenvolvimento ágil de software. Disponível em <<http://manifestoagil.com.br/>> Acesso em 03 abr. 2012

[BECK, 2004] BECK, Kent. "Programação Extrema Explicada", Bookman, (2004).

[EXTREME PROGRAMMING, 2012] Extreme Programming: A Gentle Introduction. Disponível em <<http://www.extremeprogramming.org> > Acesso em 17 abr. 2012

[IMPROVE, 2012] Scrum: metodologia ágil para gestão e planejamento de projetos - Improve It. Disponível em <<http://improveit.com.br/scrum>> Acesso em 19 abr. 2012

[SCHWABER, 2004] SCHWABER, Ken. "Agile project management with Scrum". Washington: Microsoft Press, 2004.

[AGILE, 2010] ACADEMIA AGILE. Gestão de projetos com Scrum. 2010.

[KNIBERG, 2007] KNIBERG, Henrik. "Scrum and XP fromm the Trenches – How we do Scrum". InfoQ.Com, 2007

[CRUZ, 2012] Artefatos Scrum | Fabio Cruz . com. Disponível em <<http://www.fabiocruz.com/outros/artefatos-scrum/> > Acesso em 24 abr. 2012

[KRUEGER, 2011] KRUEGER, Charles W. "The emerging practice of software product line development". Austin, TX. 2011. Disponível em <<http://mil-embedded.com/PDFs/BigLever.Fall06.pdf>>

[KRUEGER, 2006] Introduction to the Emerging Practice of Software Product Line Development. Disponível em<<http://www.methodsandtools.com/archive/archive.php?id=45>> Acesso em 07 nov. 2012

[SOUZA, 2012] SOUZA, D, S. Seleção de práticas ágeis para o desenvolvimento de linhas de produto. 2012. 103 f. Trabalho de conclusão de curso (Bacharelado em Ciências da Computação) – Universidade Federal de Santa Catarina. Santa Catarina. 2012.

[AG, 2010] ACADEMIA AGILE. Práticas Ágeis de Engenharia de Software com XP. 2010.

[ANDROID, 2013] Android Developers. Disponível em <<http://source.android.com/>> Acesso em 19 mar. 2013

[DEVELOPERS, 2013] Android Developers. Disponível em <<http://developer.android.com>> Acesso em 19 mar. 2013

[TELES, 2005] TELES, Vinícius, M. UM ESTUDO DE CASO DA ADOÇÃO DAS PRÁTICAS E VALORES DO EXTREME PROGRAMMING. 2005. 181 f.

[LECHETA, 2010] LECHETA, Ricardo R. “Google Android – Aprenda a criar aplicações para dispositivos moveis com o Android SDK”. Novatec, 2010

[SOUZA, 2013] SOUZA, Diego S., VILAIN, P. Selecting Agile Practices for Developing Software Product Lines. SEKE 2013.

[DESIGN, 2013] Design Principles | Android Developers. Disponível em <<http://developer.android.com/design/get-started/principles.html>> Acesso em 19 mar. 2013

[FAGUNDES, 2005] FAGUNDES, P. B. Framework Para Comparação e Análise de Métodos Ágeis. 2005. 134f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2005.

8. Anexos

8.1. Artigo

Aplicativos Móveis para Consulta de Informações de Gerência Ágil de Processos utilizando LPS

Bruno Fortes dos Santos

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

bruno.fortes@inf.ufsc.br

Abstract. *To render the monitoring of projects and iterations more accessible to customers and agile teams members, this paper proposes the development of three Android apps that access the information generated by web tools for agile project management. Due to the existence of common parts and points of variation between web tools, this study used a Software Product Line as a mechanism for development the proposed applications.*

Resumo. *Para tornar o acompanhamento de projetos e iterações mais acessíveis a clientes e membros de equipes ágeis, esse trabalho propõe o desenvolvimento de três aplicativos para Android que acessem as informações geradas por ferramentas web de gerência de projetos ágeis. Devido a existência de partes comuns e pontos de variação entre as ferramentas web, esse trabalho utilizou de uma Linha de Produtos de Software como mecanismo para desenvolvimento dos aplicativos propostos.*

1. Introdução

Os métodos ágeis surgiram como uma alternativa aos métodos tradicionais de desenvolvimento de software e vem se popularizando cada vez mais entre as equipes de desenvolvimento. O desenvolvimento de software em pequenas releases, permite um acompanhamento por parte do cliente do andamento do produto, permitindo um feedback sobre as funcionalidades implementadas. [SOARES, 2004]

Diversas ferramentas surgiram para auxiliar as equipes de desenvolvimento na produção de softwares utilizando métodos ágeis. Com o desenvolvimento iterativo e incremental, é importante que toda a equipe possa acompanhar o andamento das iterações, avaliando durante o andamento de cada uma delas se o ritmo está sendo suficiente para produzir uma release ao final da iteração.

O sistema operacional Android vem se popularizando e se tornando o principal sistema operacional para smartphones, tendo como característica forte o fato de ser gratuito e que suas aplicações podem ser desenvolvidas a partir de qualquer computador.

Junto a essa popularização do Android, está a necessidade de disponibilizar as informações geradas pelas ferramentas de gerência de métodos ágeis para todos os envolvidos do projeto. Sendo assim, uma forma de disseminar essas informações é através dos dispositivos móveis dos envolvidos no projeto.

As ferramentas de apoio à gestão de projetos que utilizam métodos ágeis possuem características diferentes entre si, apesar dos seus objetivos em comum. Para lidar com essas diferenças e semelhanças e convergir para uma série de produtos para dispositivos móveis que satisfaça as equipes de desenvolvimento ágil, as linhas de produtos de software surgem com uma excelente alternativa de desenvolvimento.

O presente artigo apresenta o desenvolvimento de diferentes aplicações para dispositivos móveis na plataforma Android que proveem acesso às informações geradas por funcionalidades selecionadas de algumas ferramentas de gerenciamento de projetos para equipes ágeis. O desenvolvimento desses aplicativos ocorre utilizando a abordagem de linha de produtos de software (LPS).

Este artigo está organizado da seguinte forma: na seção 2 é abordado o conceito de métodos ágeis e apresentação de algumas ferramentas de apoio à gestão de projetos que utilizam métodos ágeis; na seção 3 é feita a apresentação do conceito de LPS, abordando um framework estendido para desenvolvimento de LPS; na seção 4 é feita uma contextualização do sistema operacional Android; por fim são apresentados os aplicativos Android que acessam as informações de ferramentas para gestão de equipes ágeis desenvolvidos, com a descrição do processo de desenvolvimento, desde as práticas utilizadas, passando pela sua arquitetura e seu desenvolvimento.

2. Desenvolvimento ágil de software

O desenvolvimento ágil de software surgiu a partir de algumas necessidades não contempladas pelos métodos tradicionais de desenvolvimento de software. Eles surgem valorizando a iteração do cliente com o desenvolvimento e a entrega contínua de partes do software que agreguem valores. O desenvolvimento de forma incremental e entrega de pequenas partes do software, permite que novos requisitos sejam incluídos ao longo do processo de desenvolvimento e que o software comece a agregar valor para o cliente antes de sua entrega total.

O reconhecimento dos métodos ágeis se dá a partir de 2001 com a publicação do “Manifesto para o desenvolvimento ágil de software” [MANIFESTO, 2012].

Apoiados no desenvolvimento ágil de software surgem diversos métodos ágeis. Abaixo serão descritos os métodos mais praticados, que são o Scrum e o Extreme Programming, esses métodos também costumam ser os mais abordados pelas ferramentas de gerência de métodos ágeis.

2.1. Extreme Programming(XP)

O Extreme Programming(XP) é um método ágil criado na década de 90 por Kent Beck, um dos principais apoiadores do Manifesto Ágil. O método XP surge com a proposta de ser utilizado por equipes de dois a dez programadores, com foco no desenvolvimento de softwares com requisitos vagos que se modificam rapidamente [BECK, 2004].

O XP contribui para evitar desperdícios com funcionalidades desenvolvidas que podem ser poucas ou nunca utilizadas pelo cliente, através de iterações semanais com a priorização no desenvolvimento de funcionalidades que agreguem mais valores ao software proposto, as tarefas que agreguem menor valor são deixadas para o final do projeto, com isso caso ocorra uma diminuição do escopo, as funcionalidades que agregam menor valor deixam de ser desenvolvidas [AG, 2010].

2.2. Scrum

O Scrum é um framework que fornece práticas iterativas e incrementais para gerenciamento de equipes de desenvolvimento de software. No Scrum as iterações são chamadas de Sprint, sendo uma Sprint um ciclo onde um conjunto de atividades é executado.

Um projeto Scrum começa com a elaboração do Product Backlog que contém a lista de requisitos do sistema a ser desenvolvido pelo projeto. Ele é constantemente modificado para identificar o que o produto necessita para ser apropriado, competitivo e eficiente.

O desenvolvimento ocorre através de sprints, que são timebox de 30 dias consecutivos. Sendo esse o tempo necessário para a equipe produzir uma versão do produto que pode ser entregue e que atenda a um interesse significativo do Product Owner e dos stakeholders.

2.3. Ferramentas de Apoio aos métodos ágeis

Para o desenvolvimento desse trabalho foram selecionadas três dessas ferramentas de apoio à gestão de projetos ágeis. Como critérios para seleção foram definidos: (1) a existência de uma versão Free da ferramenta (independente da quantidade de usuários liberados); (2) possibilidade de instalação da ferramenta na máquina local ou em servidor específico (critério importante na facilitação no processo de desenvolvimento e teste da ferramenta); e (3) possibilidade de acessar os dados gerados pelas funcionalidades selecionadas na ferramenta, via API disponibilizada por ela ou através de acesso direto ao seu banco de dados.

Foram identificadas nas ferramentas, as funcionalidades que atendessem as principais práticas dos métodos ágeis, como gestão de histórias de usuários, tarefas, iterações e quadro de acompanhamento das iterações.

Sendo assim, as ferramentas selecionadas foram: tinyPM; Pronto e Agilefant.

3. Linha de Produtos de Software

Uma Linha de Produtos de Software (LPS) é um conjunto de softwares que compartilham em comum, uma série de características que satisfazem as necessidades específicas de um determinado segmento de mercado, e que são desenvolvidos a partir de um conjunto base de componentes previamente planejados para esse segmento.

O que difere a LPS do reuso oportunista de software, é que o desenvolvimento de componentes para a LPS ocorre de forma previamente planejada. Ao invés de se colocar todos os componentes dentro de uma biblioteca com a esperança de surgirem oportunidades para reutilizá-los, na LPS os artefatos de software só são criados quando é previsto que serão utilizados em um ou mais produtos da LPS [KRUEGER, 2011].

O desenvolvimento de uma LPS consiste em duas atividades principais, que são engenharia de domínio e engenharia de aplicação. A engenharia de domínio trata do levantamento de requisitos e elaboração dos componentes da LPS, já a engenharia de aplicação consiste no desenvolvido de aplicações, através da seleção de componentes obtidos no processo de engenharia de domínio.

3.1. Framework estendido

O framework de métodos ágeis proposto inicialmente em [FAGUNDES, 2005] foi elaborado a partir de uma análise comparativa de diferentes métodos ágeis e reunia as melhores práticas dos métodos ágeis, porém, não era adequado ao desenvolvimento de LPS. O trabalho realizado em [SOUZA, 2012] acrescentou práticas utilizadas na LPS ao framework elaborado em [FAGUNDES, 2005], essa extensão “permitiu a especificação de um framework que mostra as etapas e práticas para o desenvolvimento de toda uma linha de produtos de software” [SOUZA, 2012 – p. 57].

O processo de desenvolvimento da LPS é dividido em dois ciclos, denominados “Engenharia de Domínio” e “Engenharia de Aplicação”, e cada ciclo fica subdividido em atividades menores. Durante o processo de engenharia de domínio, são coletadas as informações e requisitos referentes ao domínio da LPS que se pretende desenvolver, são definidos quais requisitos são comuns a todos os produtos da LPS e quais fazem parte dos pontos de variabilidade. Durante o processo de engenharia da aplicação, um produto específico é desenvolvido a partir da LPS para desenvolvimento, ou seja, o produto é obtido através da utilização dos artefatos produzidos na engenharia de domínio.

4. Android

Segundo o site oficial do projeto Android [ANDROID, 2013], o “Android é um software open-source criado para telefones móveis e outros dispositivos”. O projeto é liderado pela Google, que fica encarregada pelo desenvolvimento e manutenção do Android.

O Android teve sua origem através de um grupo de empresas chamado Open Handset Alliance (OHA), liderado pela Google. O Projeto Android tem como objetivo principal “criar um produto real e de sucesso, que melhore a experiência de usuários finais com dispositivos móveis” [ANDROID, 2013]. Para alcançar tal objetivo, o projeto Android quer assegurar a existência de uma plataforma aberta onde operadoras, fabricante de equipamentos e desenvolvedores possam transformar suas ideias inovadoras em realidade.

O Android é intencionalmente open-source, dessa forma as organizações que compartilham necessidades em comum, podem agrupar recursos para contribuir na implementação de um único produto compartilhado que cada colaborador possa adaptar e customizar [ANDROID, 2013]. Apesar de receber várias contribuições, o Android é visto como um produto único e integral, e não uma distribuição, especificação ou coleção de partes reutilizáveis.

O Android SDK (Software Development Kit) prove a API e as ferramentas necessárias para construir, testar e depurar aplicativos para Android. O Android SDK é composto de pacotes modulares que podem ser baixados separadamente através do Android SDK Manager.

5. Aplicativos para acesso às informações das ferramentas de apoio a métodos ágeis

Nesta seção será descrito o desenvolvimento da LPS proposta neste trabalho, as práticas selecionadas, as tecnologias envolvidas e a arquitetura da LPS.

5.1. Práticas ágeis para o desenvolvimento da LPS

No estudo realizado em [SOUZA, 2012] é proposto um framework para seleção de práticas ágeis que atividades, o framework sugere várias práticas a serem adotadas na definição de um processo para a Engenharia de Domínio e Engenharia de Aplicação.

Abaixo estão listadas as práticas sugeridas pelo framework e assinaladas com “X” as práticas que serão utilizadas no desenvolvimento da LPS proposta neste trabalho. Na figura 1 estão as práticas selecionadas da Engenharia de Domínio e na figura 2 as práticas da Engenharia de Aplicação.

Engenharia de Domínio	Lista de requisitos de cada aplicação (obrigatório)	X
	Modelo de features (obrigatório)	X
	Documentação	
	Projeto da arquitetura do sistema (obrigatório)	X
	Representação da arquitetura de domínio (obrigatório)	X
	Planejamento da iteração (obrigatório)	X
	Duração da iteração (obrigatório)	X
	Distribuição dos módulos para os responsáveis	
	Reunião diária	
	Projeto da iteração	
	Desenvolvimento da iteração (obrigatório)	X
	Escrita dos testes de unidades	X
	Escrita dos testes de aceitação	
	Desenvolvimento coletivo do código	
	Programação em pares	
	Refatoração	
	Programação lado-a-lado	
	Inspeção do código	X

Figura 1. Práticas selecionadas da Engenharia de Domínio.

Engenharia de Aplicação	Lista de requisitos (Obrigatório)	X
	Seleção dos componentes (Obrigatório)	X
	Modelo geral	
	Documentação Inicial	
	Planejamento da iteração (Obrigatório)	X
	Duração de iterações (Obrigatório)	X
	Distribuição de requisitos para os responsáveis (Obrigatório)	X
	Reuniões diárias	
	Estórias do usuário	
	Casos de uso	
	Projeto da iteração	
	Desenvolvimento da iteração (Obrigatório)	X
	Escrita dos testes de unidade	
	Escrita dos testes de aceitação	
	Desenvolvimento coletivo do código	
	Programação em pares	
	Refatoração	
	Programação lado-a-lado	
	Inspeção de código	X
	Reunião de revisão da integração	X
	Geração de uma breve documentação	
	Entrega do sistema ao cliente	X

Figura 2. Práticas selecionadas da Engenharia de Domínio.

5.2. Engenharia de Domínio

5.2.1. Lista de Requisitos

Abaixo são listados os requisitos selecionados para a LPS proposta nesse trabalho:

- Product backlog (comum);
- Detalhamento de estória do backlog (comum);
- Pesquisa de estórias do backlog (comum);
- Andamento da iteração atual;
- Pesquisa de ticktes do taskboard ;
- Detalhamento de ticket;
- Pesquisa de estória do taskboard;
- Detalhamento de estória;
- Detalhes da iteração atual;
- Backlog da iteração atual;
- Possibilidade de comunicação com os sistemas Pronto, TinyPM e AgileFant.
- Interface Gráfica para dispositivo Android.

5.2.2. Modelo do Domínio

Na Figura 3, é apresentado o modelo do domínio elaborado para a LPS proposta nesse trabalho.

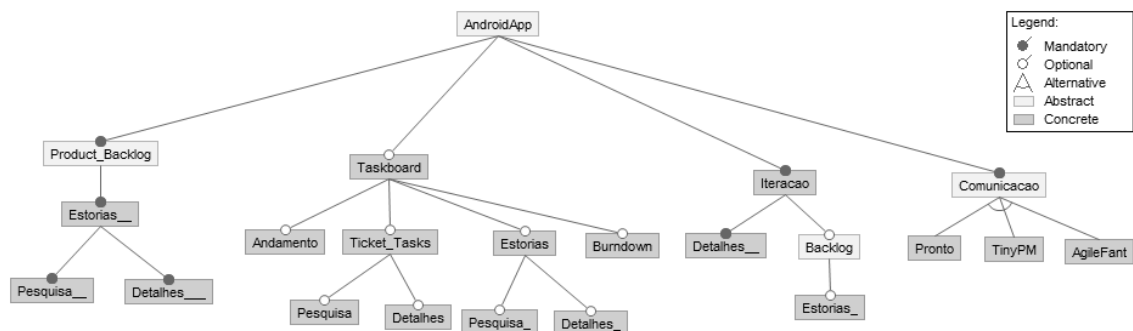


Figura 3. Modelo do domínio.

5.2.3. Projeto de Componentes

Componentes para os pontos comuns da LPS:

- Interface Gráfica;
- Product Backlog;
- Pesquisa de estórias do Product Backlog;
- Detalhes de estórias do Product Backlog;
- Detalhes da Iteração Atual.

Componentes que farão parte dos pontos de variação da Linha de Produtos:

- Taskboard;
- Andamento dos Tickets;
- Pesquisa dos Tickets do Taskboard;
- Detalhe dos Tickets do Taskboard;
- Backlog da Iteração;
- Comunicação.

5.2.4. Representação da Arquitetura

A Figura 4 apresenta a representação da arquitetura para desenvolvimento da LPS proposta nesse trabalho:

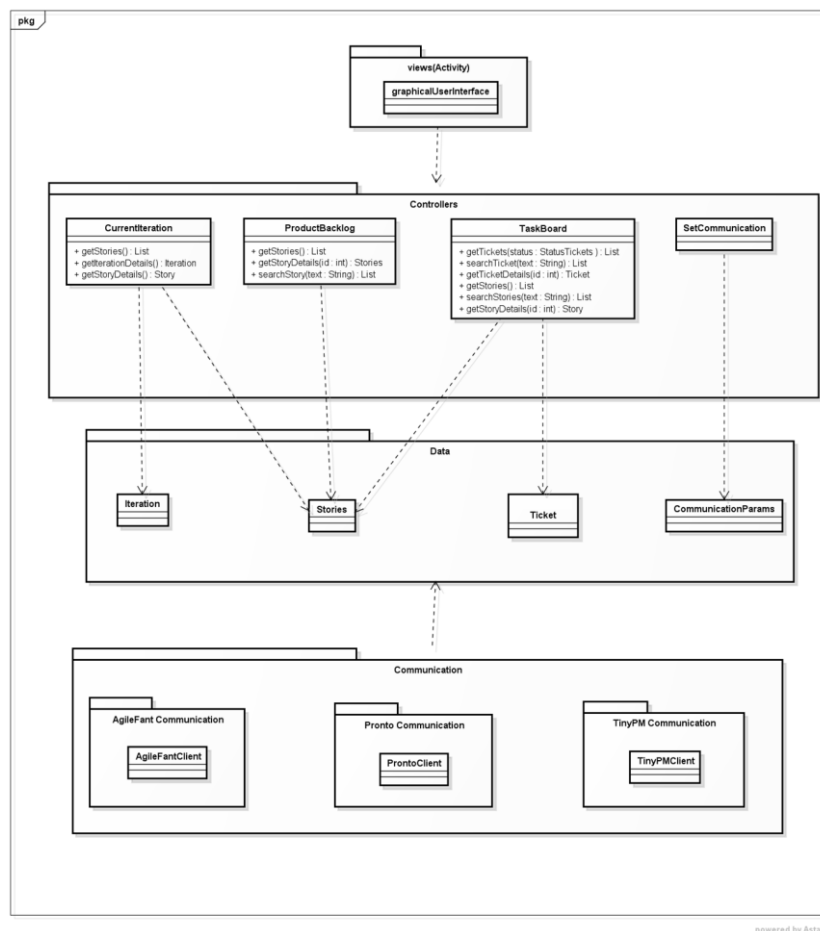


Figura 4. Representação da arquitetura.

5.2.5. Iteração 1

Nessa iteração foram desenvolvidas as classes para comunicação com a API de cada uma das ferramentas TinyPM, Pronto e AgileFant. Cada classe obterá um json com os dados requisitados e armazenará esses dados em um banco de dados SQLite local. Os dados retornados serão as informações relacionadas à iteração atual, ao taskboard e ao product backlog.

A figura 5 apresenta as classes desenvolvidas durante a iteração 1:



Figura 5. Classes produzidas durante a iteração 1.

5.2.6. Iteração 2

Nessa iteração foram desenvolvidas as camadas controller e view. Foi realizada a Codificação dos objetos CurrentIterations, ProductBacklog e Taskboard. Desenvolvimento de interface gráfica que respeite o guia de estilos estipulado pela Google.

A figura 6 apresenta as classes desenvolvidas durante a iteração 2:

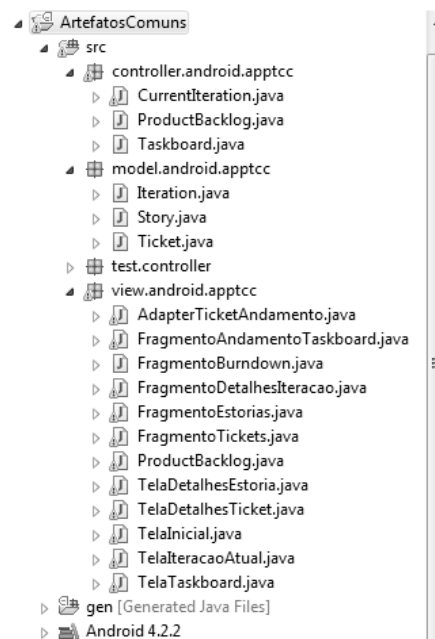


Figura 6. Classes produzidas durante a iteração 2.

5.3. Engenharia de Aplicação

Nesta etapa serão desenvolvidos três produtos da LPS, cada produto será responsável pela comunicação com cada uma das ferramentas web para gerenciamento de projetos ágeis, selecionadas para esse trabalho.

Abaixo, serão apresentados cada um dos aplicativos desenvolvidos.

5.3.1. Primeiro produto – AppTCCPronto

O segundo produto é um aplicativo Android para comunicação com a ferramenta web Pronto, chamado de AppTCCPronto. Na Figura 7 é apresentado o modelo de domínio do produto desenvolvido nessa etapa:

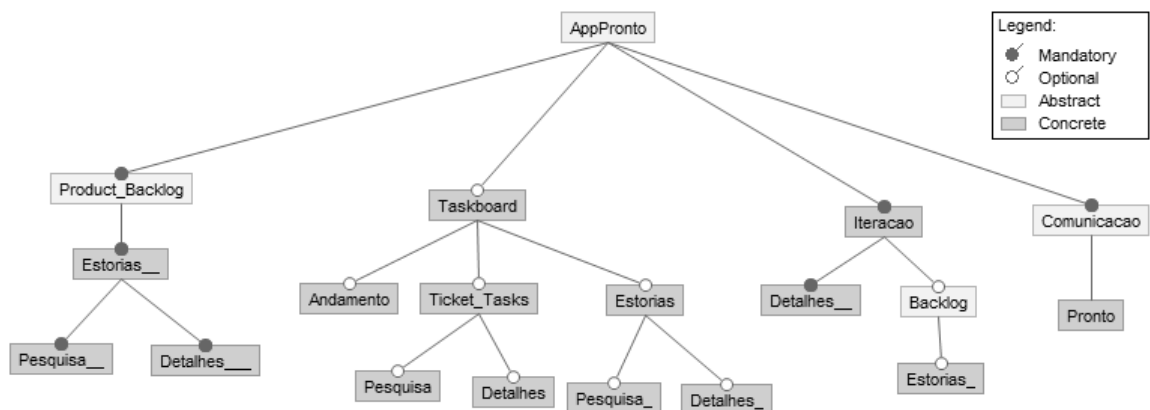


Figura 7. Modelo de domínio do Aplicativo AppTCCPronto.

5.3.2. Segundo produto – AppTCCAgilefant

O segundo produto é um aplicativo Android para comunicação com a ferramenta web Agilefant, chamado de AppTCCAgilefant. Na Figura 8 é apresentado o modelo de domínio do produto desenvolvido nessa etapa:

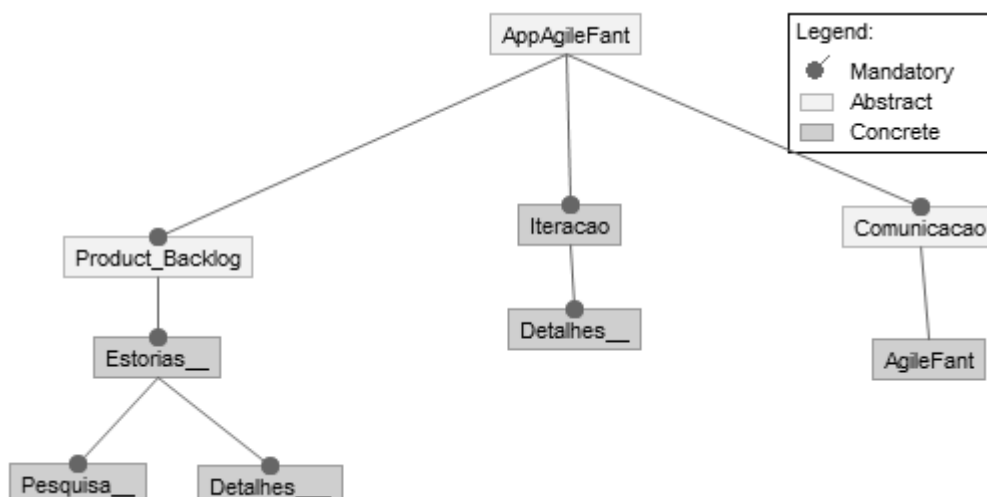


Figura 8. Modelo de domínio do Aplicativo AppTCCAgilefant.

5.3.3. Terceiro produto – AppTCCTinyPM

O terceiro produto é um aplicativo Android para comunicação com a ferramenta web TinyPM, chamado de AppTCCTinyPM. Na Figura 9 é apresentado o modelo de domínio do produto desenvolvido nessa etapa:

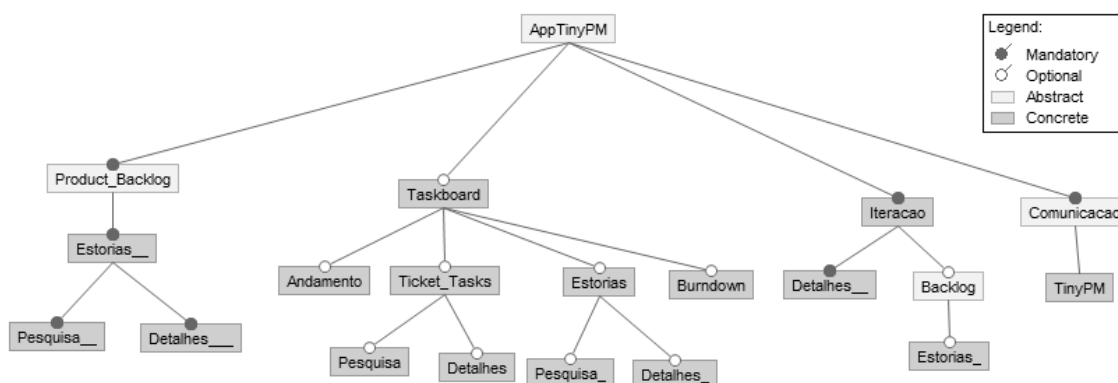


Figura 9. Modelo de domínio do Aplicativo AppTCCTinyPM.

6. Conclusão

Esse trabalho resultou no desenvolvimento de uma LPS composta de três aplicativos para Android, onde cada aplicativo é responsável por acessar os dados gerados por três diferentes ferramentas web para gerência de equipes ágeis selecionadas. As ferramentas web selecionadas foram Pronto, tinyPM e Agilefant, por serem ferramentas de código aberto ou por possuírem uma versão gratuita que pudesse ser baixada e instalada em um servidor próprio.

Uma experiência que vale ressaltar, foi que durante a modelagem da representação da arquitetura, percebeu-se que para cada feature do modelo de features poderia existir um controlador correspondente na representação da arquitetura, e para cada ação (opção) da feature poderia existir um método correspondente dentro do controlador. Dessa forma, a representação da arquitetura espelha o modelo de features, indicando como ele será desenvolvido e garantido que todo ele será atendido pela LPS.

O framework estendido para o desenvolvimento da LPS proposto em [SOUZA, 2013] foi de grande auxílio no desenvolvimento da LPS proposta nesse trabalho, facilitando que os objetivos do trabalho fossem atingidos. O desenvolvimento da LPS proposto através de iterações permitiu que pequenas entregas fossem realizadas e validadas ao longo do processo de elaboração da mesma. A LPS desenvolvida nesse trabalho reforça a eficiência do framework estendido, servindo como um exemplo de sua aplicação e validação das práticas propostas.

Referências

- [SOARES, 2004] SOARES, Michel dos Santos. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. Disponível em: <<http://revistas.facecla.com.br/index.php/reinfo/article/view/146/38>> Acesso em 03 abr. 2012
- [MANIFESTO, 2012] Manifesto para o desenvolvimento ágil de software. Disponível em <<http://manifestoagil.com.br/>> Acesso em 03 abr. 2012

- [BECK, 2004] BECK, Kent. “Programação Extrema Explicada”, Bookman, (2004).
- [AG, 2010] ACADEMIA AGILE. Práticas Ágeis de Engenharia de Software com XP. 2010.
- [KRUEGER, 2011] KRUEGER, Charles W. “The emerging practice of software product line development”. Austin, TX. 2011. Disponível em <<http://mil-embedded.com/PDFs/BigLever.Fall06.pdf>>
- [FAGUNDES, 2005] FAGUNDES, P. B. Framework Para Comparação e Análise de Métodos Ágeis. 2005. 134f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2005.
- [ANDROID, 2013] Android Developers. Disponível em <<http://source.android.com/>> Acesso em 19 mar. 2013
- [SOUZA, 2012] SOUZA, D, S. Seleção de práticas ágeis para o desenvolvimento de linhas de produto. 2012. 103 f. Trabalho de conclusão de curso (Bacharelado em Ciências da Computação) – Universidade Federal de Santa Catarina. Santa Catarina. 2012.
- [SOUZA, 2013] SOUZA, Diego S., VILAIN, P. Selecting Agile Practices for Developing Software Product Lines. SEKE 2013.

8.2. Manual de Instalação

Para a elaboração desse manual, foi considerada a instalação do ambiente em uma máquina com o sistema operacional Windows 7. A instalação pode ser diferente se realizada em outros sistemas operacionais.

1. Instalação das ferramentas web para gerência de projetos ágeis

Primeiramente serão descritos os passos para instalação das ferramentas Pronto, Agilefant e TinyPM, elas são as ferramentas web para gerência de projetos ágeis que foram selecionadas para o trabalho.

Os requisitos para instalação das ferramentas são:

- **Java Runtime Environment (JRE) 6.0** (obrigatório para todas as ferramentas);
- **Apache Tomcat 6.x** (obrigatório para todas as ferramentas);
- **MySQL 5.x** (obrigatório para as ferramentas TinyPM e AgileFant);
- **MySQL Connector/J** (obrigatório para as ferramentas TinyPM e AgileFant);
- **PostgreSQL 8.4** (obrigatório para a ferramenta Pronto).

a. Instalação do Java Runtime Environment

- i. Fazer o download do JRE através do link:
<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html#jre-6u45-oth-JPR>
- ii. Rodar o instalador e seguir os passos indicados por ele, sem nenhum detalhe adicional.

b. Instalação do Apache Tomcat

- i. Fazer o download do arquivo “*32-bit/64-bit Windows Service Installer*” através do link: <http://tomcat.apache.org/download-60.cgi#6.0.37>
- ii. Rodar o instalador e indicar um diretório do computador para sua instalação. Esse diretório ficará conhecido como `%CATALINA_HOME%` para o restante do manual, esse é também o nome da variável de sistema criada pelo Windows para referenciar o diretório de instalação do Tomcat.
- iii. Se a instalação ocorreu de forma correta, aparecerá um *tryIcon* do Tomcat junto ao relógio do Windows. Dê um duplo-clique nesse ícone, clique no botão *Start* na tela que abrirá em seguida.

- iv. O funcionamento do Tomcat pode ser verificado através do link:
<http://localhost:8080/>

c. Instalação do MySQL

- i. Fazer o download do instalador do MySQL através do link:
<http://dev.mysql.com/downloads/mysql/5.0.html>
- ii. Rodar o instalador e seguir os passos indicados nele, nenhum detalhe adicional até aqui.

d. MySQL Connector/J

- i. Fazer o download do arquivo compactado do MySQL Connector/J, através do link:
<http://dev.mysql.com/downloads/connector/j>
<http://dev.mysql.com/downloads/connector/j>
- ii. Descompacta-lo no diretório `%CATALINA_HOME%/lib`

e. Instalação do PostgreSQL

- i. Fazer o download do instalador do PostgreSQL 8.4.17 através do link:
<http://www.enterprisedb.com/products-services-training/pgdownload#windows>
- ii. Rodar o instalador e seguir os passos indicados por ele. Recomendo que seja instalado com o usuário “pronto” e senha “pronto”, isso facilitará um pouco a posterior instalação da ferramenta Pronto.

f. Instalação do TinyPM

- i. Fazer o download do tinyPM versão 2.6.5 através do link:
<http://www.tinypm.com/download>
- ii. Descompactar o arquivo em um diretório do computador.
- iii. Abrir o MySQL Command Line Client e rodar o seguinte comando para criação do banco de dados:

```
CREATE DATABASE tinypmdb CHARACTER SET = 'utf8' COLLATE = 'utf8_general_ci';
```
- iv. Ainda no MySQL Command Line Client, rodar o seguinte comando para liberar acesso ao banco de dados recém criado, :

```
GRANT SELECT, UPDATE, INSERT, LOCK TABLES, DELETE ON tinypmdb.* TO 'username'@'localhost' IDENTIFIED BY 'password';  
FLUSH PRIVILEGES;
```
- v. Rodar os seguintes comandos para criação da estrutura do BD e dos dados iniciais, substituindo o `%PATH_TO_INSTALL_FOLDER%` pelo caminho do diretório onde o arquivo foi compactado:

```
\u tinypmdb
```

```
\. %PATH_TO_INSTALL_FOLDER%/sql/mysql/create_schema-2.x.sql  
\. %PATH_TO_INSTALL_FOLDER%/sql/mysql/create_data-2.x.sql
```

- vi. Copiar todos os arquivos da pasta “dependency” do arquivo descompactado, para dentro do diretório %CATALINA_HOME%/lib.
- vii. Parar o servidor Tomcat, através do *trylcon* ao lado do relógio do Windows.
- viii. Criar o diretório %CATALINA_HOME%/webapps/tinypm e descompactar o arquivo `tinypm.war` para dentro desse diretório.
- ix. Entrar no arquivo %CATALINA_HOME%/webapps/tinypm/WEB-INF/classes/hibernate.properties e modificar o usuário e senha de acordo o que foi definido na instalação do MySQL.
- x. Iniciar o servidor Tomcat.
- xi. O tinyPM deve estar acessível através do link: <http://localhost:8080/tinypm> faça login com os seguintes dados:
 - o Usuário: admin
 - o Senha: admin

g. Instalação do AgileFant

- i. Fazer o download do Agilefant através do link: <http://sourceforge.net/projects/agilefant/>
- ii. Descompactar o arquivo baixado em uma pasta qualquer do computador.
- iii. Abrir o MySQL Command Line Client e rodar os seguintes comandos para criação do banco de dados:

```
mysql -u [mysql username] -p (enter [mysql password] when  
asked) create database [Agilefant instance name];  
  
grant all on [Agilefant instance name].* to [Agilefant  
instance name]@localhost identified by '[strong unique  
password]';
```
- iv. Descompactar o arquivo `agilefant.war` dentro do diretório %CATALINA_HOME%/webapps/agilefant e editar o arquivo `/WEB-INF/agilefant.conf` alterando os dados de conexão com o banco de dados, de acordo com o que foi definido no momento da instalação do MySQL.
- v. Parar o servidor Tomcat.
- vi. Colocar o arquivo `agilefant.war` no diretório: %CATALINA_HOME%/webapps e iniciar o servidor Tomcat.

h. Instalação do Pronto

- i. Fazer o download dos arquivos através do seguinte link:
<https://github.com/bluesoft/pronto-agile>
- ii. Seguir as instruções de instalação contidas em:
<https://github.com/bluesoft/pronto-agile/blob/master/instalar.txt>
- iii. O manual apontando anteriormente é bem completo e escrito em português, por isso recomendo a utilização dele para a instalação do Pronto.

2. Instalação dos webservices para comunicação entre as ferramentas e os aplicativos Android

Para comunicação entre as ferramentas web e os aplicativos Android desenvolvidos no trabalho, é necessária a instalação de três webservices que foram desenvolvidos utilizando o framework Ruby on Rails. Abaixo seguem os passos para a instalação desses webservices.

a. Instalação do Ruby on Rails

- i. Fazer o download da versão 1.9.3 p429 do Ruby através do seguinte link:
http://rubyforge.org/frs/?group_id=167
- ii. Rodar o instalador baixado no passo anterior e seguir os passos indicados por ele.
- iii. Fazer o download da versão 1.8.24 do RubyGems através do seguinte link:
http://rubyforge.org/frs/?group_id=126
- iv. Descompactar o arquivo baixado dentro do diretório onde foi instalado o Ruby no passo ii. Entrar no prompt de comandos e executar os seguintes comandos:

```
cd %RUBYGEMS_DIR%
ruby setup.rb init
ruby setup.rb install
```
- v. Rodar o seguinte comando para instalação do Ruby on Rails:
gem install rails
- vi. A execução do comando anterior pode resultar em mensagens de erros, de acordo com o computador onde foi rodado e de programas já previamente instalados. Junto com as mensagens de erros vem uma explicação do que deve ser feito para corrigi-los, se algum erro ocorrer basta seguir as instruções indicadas na mensagem.

b. Rodando os webservices para comunicação

- i. Fazer o download dos webservices disponíveis em:
http://www.inf.ufsc.br/~bruno.fortes/Comunicacao_Ferramentas_web_Android.rar e descompactar o arquivo em algum diretório do computador (no caso de máquina Windows, recomendo que esse diretório esteja contido na unidade "c:\").
- ii. Abrir o prompt de comandos do Windows e executar os seguintes comandos:

```
cd %DIR_WEBSERVICES%\ comTinyPM  
rails server -p 3000  
  
cd %DIR_WEBSERVICES%\ comAgileFant  
rails server -p 3001  
  
cd %DIR_WEBSERVICES%\ comPronto  
rails server -p 3002
```

3. Compilando e executando os aplicativos desenvolvidos

Os aplicativos foram desenvolvidos utilizando o ADT(Android Development Tools), ele é plugin do Eclipse para desenvolvimento de aplicativos Android. Abaixo seguem os passos para download e instalação do ADT e compilação dos aplicativos desenvolvidos:

a. Instalação do ADT

- i. No seguinte link, pode ser realizado o download de uma versão do Eclipse já configurada com o ADT: <http://developer.android.com/sdk/index.html>
- ii. O arquivo só precisa ser baixado e descompactado para um diretório qualquer do computador, dentro da pasta "eclipse" do arquivo descompactado está o arquivo `eclipse.exe` que executa o eclipse com o ADT configurado.

b. Compilação dos aplicativos

- i. No seguinte endereço pode ser obtido os aplicativos desenvolvidos nesse trabalho:
[http://www.inf.ufsc.br/~bruno.fortes/LPS e Aplicativos desenvolvidos.rar](http://www.inf.ufsc.br/~bruno.fortes/LPS_e_Aplicativos_desenvolvidos.rar)
- ii. Descompactar o arquivo em um diretório qualquer do computador.
- iii. Importar cada um dos aplicativos para o Eclipse baixado anteriormente e realizar a compilação deles.
- iv. Os aplicativos estão rodando, através da opção "Configurações" pode-se alterar a url dos webservices instaladas anteriormente.

Obs: Os emuladores Android utilizam o endereço “http://10.0.0.2” para acessar o servidor localhost. Dessa forma, os endereços para os webservices serão “http://10.0.0.2:3000”, “http://10.0.0.2:3001” e “http://10.0.0.2:3002”.

8.3. Código Fonte da Linha de Produtos Desenvolvida

```
-----AgileFant.java-----
package communication.android.apptcc;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import data.android.apptcc.ConfData;
import data.android.apptcc.IterationData;
import data.android.apptcc.StoriesData;

import android.content.ContentValues;
import android.content.Context;

public class AgileFant {

    private String agilefantURL;
    private static AgileFant agilefant;

    private AgileFant() {
    }

    public static AgileFant getInstance(Context context) {
        if (agilefant == null){
            agilefant = new AgileFant();

            agilefant.setAgilefantURL(new
ConfData(context).getURL());
        }
        return agilefant;
    }

    public void requestIterationData(Context context){

        Integer idIteration = 0;
        String response = makeRequest ( this.agilefantURL +
"iteration/current" );

        // Current Iteration details
        try {
            JSONObject jsonIteration = new JSONObject ( response
);

            idIteration = jsonIteration.getInt("id");

            String title = jsonIteration.getString("name");
            String goal = jsonIteration.getString("description");
```

```

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-
dd'T'hh:mm:ss'Z'", Locale.US);
        Date startDate = null;
        Date endDate = null;
        try {
            startDate =
sdf.parse(jsonIteration.getString("startDate"));
            endDate =
sdf.parse(jsonIteration.getString("endDate"));
        } catch (ParseException e) {
            e.printStackTrace();
        }

        IterationData iteration = new IterationData(context);
        iteration.deleteAll();
        iteration.insert(idIteration, title, startDate,
endDate, goal);

    } catch (JSONException e) {
        e.printStackTrace();
    }

    // Iteration backlog
    response = makeRequest ( this.agilefantURL +
"iteration/backlog" );
    try {
        JSONArray jsonStories = new JSONArray ( response );

        StoriesData storiesData = new StoriesData(context);
        storiesData.deleteAll();

        for (int i = 0; i < jsonStories.length(); i++) {
            JSONObject story =
jsonStories.getJSONObject(i);

            ContentValues values = new ContentValues();
            values.put("id", story.getInt("id"));
            values.put("title", story.getString("name"));
            values.put("description",
story.getString("description"));
            values.put("owner", "");

            if (!story.isNull("iteration_id")){
                values.put("iterationID",
story.getInt("iteration_id"));
            }else{
                values.put("iterationID", 0);
            }

            storiesData.insert(values);
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

public void requestProductBacklogData(Context context){
    String response = makeRequest ( this.agilefantURL +
"product/backlog" );

```

```

        try {
            JSONArray jsonStories = new JSONArray ( response );
            StoriesData storiesData = new StoriesData(context);
            storiesData.deleteAll();

            for (int i = 0; i < jsonStories.length(); i++) {
                JSONObject story =
jsonStories.getJSONObject(i);

                ContentValues values = new ContentValues();
                values.put("id", story.getInt("id"));
                values.put("title", story.getString("name"));
                values.put("description",
story.getString("description"));
                values.put("owner", "");

                if (!story.isNull("iteration_id")){
                    values.put("iterationID",
story.getInt("iteration_id"));
                }else{
                    values.put("iterationID", 0);
                }

                storiesData.insert(values);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    private String makeRequest(String urlAddress){
        HttpURLConnection con = null ;
        URL url = null ;
        String response = null ;

        try {
            url = new URL ( urlAddress );
            con = ( HttpURLConnection ) url.openConnection();
            response = readStream ( con.getInputStream());

        } catch ( Exception e) {
            e.printStackTrace();
        } finally {
            con.disconnect ();
        }

        return response ;
    }

    private String readStream(InputStream in){
        BufferedReader reader = null ;
        StringBuilder builder = new StringBuilder ();

        try {
            reader = new BufferedReader (new InputStreamReader
(in));

            String line = null ;

            while (( line = reader.readLine ()) != null ) {
                builder.append ( line + "\n");
            }

```

```

        } catch ( IOException e) {
            e.printStackTrace ();
        } finally {
            if ( reader != null ) {
                try {
                    reader.close ();
                } catch ( IOException e) {
                    e.printStackTrace ();
                }
            }
        }
        return builder.toString ();
    }

    public String getAgilefantURL() {
        return agilefantURL;
    }

    public void setAgilefantURL(String agilefantURL) {
        this.agilefantURL = agilefantURL;
    }
}

-----TinyPM.Java-----
package communication.android.apptcc;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.content.ContentValues;
import android.content.Context;

import data.android.apptcc.ConfData;
import data.android.apptcc.IterationData;
import data.android.apptcc.StoriesData;
import data.android.apptcc.TicketsData;

public class TinyPM {

    private String tinypmURL;
    private static TinyPM tinypm;

    private TinyPM() {

    }

    public static TinyPM getInstance(Context context) {
        if (tinypm == null) {
            tinypm = new TinyPM();
        }
    }
}

```

```

        tinypm.setTinypmURL(new ConfData(context).getURL());
    }
    return tinypm;
}

public void requestIterationData(Context context) {

    // if (URLUtil.isHttpsUrl(this.tinypmURL)){
    Integer idIteration = 0;
    String response = makeRequest(this.tinypmURL +
"iteration/current");

    // Current Iteration details
    try {
        JSONObject jsonIteration = new JSONObject(response);

        idIteration = jsonIteration.getInt("id");
        String title = jsonIteration.getString("name");
        Integer duration = jsonIteration.getInt("duration");
        String goal = jsonIteration.getString("goal");

        SimpleDateFormat sdf = new SimpleDateFormat(
            "yyyy-MM-dd'T'hh:mm:ss'Z'", Locale.US);
        Date startDate = null;
        try {
            startDate =
sdf.parse(jsonIteration.getString("startDate"));
        } catch (ParseException e) {
            e.printStackTrace();
        }

        Calendar cal = Calendar.getInstance();
        cal.setTime(startDate);
        cal.add(Calendar.DAY_OF_MONTH, duration);

        Date endDate = cal.getTime();

        IterationData iteration = new IterationData(context);
        iteration.deleteAll();
        iteration.insert(idIteration, title, startDate,
endDate, goal);

    } catch (JSONException e) {
        e.printStackTrace();
    }

    // Iteration backlog
    response = makeRequest(this.tinypmURL +
"iteration/backlog");
    try {
        JSONArray jsonStories = new JSONArray(response);

        StoriesData storiesData = new StoriesData(context);
        storiesData.deleteAll();

        for (int i = 0; i < jsonStories.length(); i++) {
            JSONObject story =
jsonStories.getJSONObject(i);

            ContentValues values = new ContentValues();

```

```

        values.put("id", story.getInt("id"));
        values.put("title", story.getString("name"));
        values.put("description",
story.getString("description"));
        values.put("owner",
story.getString("createdBy"));

        if (!story.isNull("iterationId")) {
            values.put("iterationID",
story.getInt("iterationId"));
        } else {
            values.put("iterationID", 0);
        }

        storiesData.insert(values);
    }
} catch (JSONException e) {
    e.printStackTrace();
}
// }
}

public void requestTaskboardData(Context context) {

    // if (URLUtil.isHttpsUrl(this.tinyPmURL)) {
    // Taskboard Tickets
    String response = makeRequest(this.tinyPmURL +
"taskboard/tasks");
    try {
        JSONArray jsonTickets = new JSONArray(response);
        TicketsData ticketsData = new TicketsData(context);
        ticketsData.deleteAll();

        for (int i = 0; i < jsonTickets.length(); i++) {
            JSONObject ticket =
jsonTickets.getJSONObject(i);

            ContentValues values = new ContentValues();
            values.put("id", ticket.getInt("id"));
            values.put("title", ticket.getString("name"));
            values.put("description",
ticket.getString("description"));

            if (!ticket.isNull("estimatedEffort")) {
                values.put("effort",
ticket.getInt("estimatedEffort"));
            } else {
                values.put("effort", 0);
            }

            if (!ticket.isNull("progressStateId")) {
                values.put("status",
ticket.getInt("progressStateId") - 1);
            } else {
                values.put("status", 0);
            }

            ticketsData.insert(values);
        }
    } catch (JSONException e) {

```

```

        e.printStackTrace();
    }

    // Taskboard Stories
    response = makeRequest(this.tinypmURL +
"taskboard/backlog");
    try {
        JSONArray jsonStories = new JSONArray(response);
        StoriesData storiesData = new StoriesData(context);
        storiesData.deleteAll();

        for (int i = 0; i < jsonStories.length(); i++) {
            JSONObject story =
jsonStories.getJSONObject(i);

            ContentValues values = new ContentValues();
            values.put("id", story.getInt("id"));
            values.put("title", story.getString("name"));
            values.put("description",
story.getString("description"));
            values.put("owner",
story.getString("createdBy"));

            if (!story.isNull("iterationId")) {
                values.put("iterationID",
story.getInt("iterationId"));
            } else {
                values.put("iterationID", 0);
            }

            storiesData.insert(values);
        }

    } catch (JSONException e) {
        e.printStackTrace();
    }
    // }
}

public void requestProductBacklogData(Context context) {
    // if (URLUtil.isHttpsUrl(this.tinypmURL)){
    String response = makeRequest(this.tinypmURL +
"product/backlog");
    try {
        JSONArray jsonStories = new JSONArray(response);
        StoriesData storiesData = new StoriesData(context);
        storiesData.deleteAll();

        for (int i = 0; i < jsonStories.length(); i++) {
            JSONObject story =
jsonStories.getJSONObject(i);

            ContentValues values = new ContentValues();
            values.put("id", story.getInt("id"));
            values.put("title", story.getString("name"));
            values.put("description",
story.getString("description"));
            values.put("owner",
story.getString("createdBy"));

            if (!story.isNull("iterationId")) {

```

```

        values.put("iterationID",
story.getInt("iterationID"));
    } else {
        values.put("iterationID", 0);
    }

    storiesData.insert(values);
}
} catch (JSONException e) {
    e.printStackTrace();
}
// }
}

private String makeRequest(String urlAddress) {
    HttpURLConnection con = null;
    URL url = null;
    String response = null;

    try {
        url = new URL(urlAddress);
        con = (HttpURLConnection) url.openConnection();
        response = readStream(con.getInputStream());

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        con.disconnect();
    }

    return response;
}

private String readStream(InputStream in) {
    BufferedReader reader = null;
    StringBuilder builder = new StringBuilder();

    try {
        reader = new BufferedReader(new
InputStreamReader(in));
        String line = null;

        while ((line = reader.readLine()) != null) {
            builder.append(line + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    return builder.toString();
}

public String getTinypmURL() {
    return tinypmURL;
}

```

```

    }

    public void setTinypmURL(String tinypmURL) {
        this.tinypmURL = tinypmURL;
    }

}

-----Pronto.java-----
package communication.android.apptcc;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.content.ContentValues;
import android.content.Context;
import data.android.apptcc.ConfData;
import data.android.apptcc.IterationData;
import data.android.apptcc.StoriesData;
import data.android.apptcc.TicketsData;

public class Pronto {

    private String prontoURL;
    private static Pronto pronto;

    private Pronto() {
    }

    public static Pronto getInstance(Context context) {
        if (pronto == null) {
            pronto = new Pronto();

            pronto.setProntoURL(new ConfData(context).getURL());
        }
        return pronto;
    }

    public void requestIterationData(Context context) {

        Integer idIteration = 0;

```

```

        String response = makeRequest(this.prontoURL +
"iteration/current");

        // Current Iteration details
        try {
            JSONObject jsonIteration = new JSONObject(response);

            idIteration = jsonIteration.getInt("sprint_key");
            String title = jsonIteration.getString("nome");
            String goal = jsonIteration.getString("meta");

            SimpleDateFormat sdf = new SimpleDateFormat(
                "yyyy-MM-dd'T'hh:mm:ss'Z'", Locale.US);
            Date startDate = null;
            Date endDate = null;
            try {
                startDate =
sdf.parse(jsonIteration.getString("data_inicial"));
                endDate =
sdf.parse(jsonIteration.getString("data_final"));
            } catch (ParseException e) {
                e.printStackTrace();
            }

            IterationData iteration = new IterationData(context);
            iteration.deleteAll();
            iteration.insert(idIteration, title, startDate,
endDate, goal);

        } catch (JSONException e) {
            e.printStackTrace();
        }

        // Iteration backlog
        response = makeRequest(this.prontoURL +
"iteration/backlog");
        try {
            JSONArray jsonStories = new JSONArray(response);

            StoriesData storiesData = new StoriesData(context);
            storiesData.deleteAll();

            for (int i = 0; i < jsonStories.length(); i++) {
                JSONObject story =
jsonStories.getJSONObject(i);

                ContentValues values = new ContentValues();
                values.put("id", story.getInt("backlog_key"));
                values.put("title", story.getString("titulo"));
                values.put("description",
story.getString("descricao"));
                values.put("owner",
story.getString("solicitador"));

```

```

        if (!story.isNull("sprint")) {
            values.put("iterationID",
story.getInt("sprint"));
        } else {
            values.put("iterationID", 0);
        }

        storiesData.insert(values);
    }
} catch (JSONException e) {
    e.printStackTrace();
}
}

public void requestTaskboardData(Context context) {

    // Taskboard Tickets
    String response = makeRequest(this.prontoURL +
"taskboard/tickets");
    try {
        JSONArray jsonTickets = new JSONArray(response);
        TicketsData ticketsData = new TicketsData(context);
        ticketsData.deleteAll();

        for (int i = 0; i < jsonTickets.length(); i++) {
            JSONObject ticket =
jsonTickets.getJSONObject(i);

            ContentValues values = new ContentValues();
            values.put("id", ticket.getInt("ticket_key"));
            values.put("title",
ticket.getString("titulo"));
            values.put("description",
ticket.getString("descricao"));
            values.put("effort", 0);

            if (!ticket.isNull("kanban_status_key")) {

                if (ticket.getInt("kanban_status_key") ==
100) {
                    values.put("status", 2);
                } else {
                    values.put("status",

ticket.getInt("kanban_status_key") - 1);
                }

            } else {
                values.put("status", 0);
            }

            ticketsData.insert(values);

```

```

        }

        } catch (JSONException e) {
            e.printStackTrace();
        }

        // Taskboard Stories
        response = makeRequest(this.prontoURL +
"taskboard/backlog");
        try {
            JSONArray jsonStories = new JSONArray(response);
            StoriesData storiesData = new StoriesData(context);
            storiesData.deleteAll();

            for (int i = 0; i < jsonStories.length(); i++) {
                JSONObject story =
jsonStories.getJSONObject(i);

                ContentValues values = new ContentValues();
                values.put("id", story.getInt("id"));
                values.put("title", story.getString("name"));
                values.put("description",
story.getString("description"));
                values.put("owner",
story.getString("createdBy"));

                if (!story.isNull("iterationId")) {
                    values.put("iterationID",
story.getInt("iterationId"));
                } else {
                    values.put("iterationID", 0);
                }

                storiesData.insert(values);
            }

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    public void requestProductBacklogData(Context context) {
        String response = makeRequest(this.prontoURL +
"product/backlog");
        try {
            JSONArray jsonStories = new JSONArray(response);
            StoriesData storiesData = new StoriesData(context);
            storiesData.deleteAll();

            for (int i = 0; i < jsonStories.length(); i++) {
                JSONObject story =
jsonStories.getJSONObject(i);

```

```

        ContentValues values = new ContentValues();
        values.put("id", story.getInt("backlog_key"));
        values.put("title", story.getString("titulo"));
        values.put("description",
story.getString("descricao"));
        values.put("owner",
story.getString("solicitador"));

        if (!story.isNull("sprint")) {
            values.put("iterationID",
story.getInt("sprint"));
        } else {
            values.put("iterationID", 0);
        }

        storiesData.insert(values);
    }
} catch (JSONException e) {
    e.printStackTrace();
}
}

private String makeRequest(String urlAddress) {
    HttpURLConnection con = null;
    URL url = null;
    String response = null;

    try {
        url = new URL(urlAddress);
        con = (HttpURLConnection) url.openConnection();
        response = readStream(con.getInputStream());

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        con.disconnect();
    }

    return response;
}

private String readStream(InputStream in) {
    BufferedReader reader = null;
    StringBuilder builder = new StringBuilder();

    try {
        reader = new BufferedReader(new
InputStreamReader(in));
        String line = null;

        while ((line = reader.readLine()) != null) {
            builder.append(line + "\n");
        }
    }

```

```

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return builder.toString();
    }

    public String getProntoURL() {
        return prontoURL;
    }

    public void setProntoURL(String prontoURL) {
        this.prontoURL = prontoURL;
    }
}

-----CurrentIteration.java-----
package controller.android.apptcc;

import java.util.List;

import android.content.Context;

import data.android.apptcc.IterationData;
import data.android.apptcc.StoriesData;

import model.android.apptcc.Iteration;
import model.android.apptcc.Story;

public class CurrentIteration {

    public CurrentIteration() {

    }

    public List<Story> getStories(Context context) {
        StoriesData stories = new StoriesData(context);
        return stories.getCurrentIterationStories();
    }

    public Story getStoryDetails(int id) {
        return new Story(id, "Estória" + Integer.toString(id),
            "Essa é a descrição da estória " +
            Integer.toString(id),
            "Bruno");
    }
}

```

```

        public Iteration getIterationDetails(Context context) {
            IterationData iteration = new IterationData(context);
            return iteration.getCurrentIterationData();
        }
    }

-----ProductBacklog.java-----
package controller.android.apptcc;

import java.util.List;

import communication.android.apptcc.TinyPM;

import android.content.Context;

import data.android.apptcc.StoriesData;

import model.android.apptcc.Story;

public class ProductBacklog {

    public ProductBacklog() {

    }

    public List<Story> getStories(Context context) {
        StoriesData stories = new StoriesData(context);
        return stories.getProductBacklogStories();
    }

    public Story getStoryDetails(int id) {
        return new Story(id, "Estória" + Integer.toString(id),
            "Essa é a descrição da estória " +
Integer.toString(id),
            "Bruno");
    }

}

-----Taskboard.java-----
package controller.android.apptcc;

import java.util.ArrayList;
import java.util.List;

import data.android.apptcc.StoriesData;
import data.android.apptcc.TicketsData;

import android.content.Context;

import model.android.apptcc.Story;
import model.android.apptcc.Ticket;

```

```

public class Taskboard {

    public Taskboard() {

    }

    public List<Ticket> getTickets(Context context, StatusTickets
status) {
        TicketsData tickets = new TicketsData(context);
        return tickets.getTickets(status);
    }

    public List<Ticket> searchTicket(String text) {
        List<Ticket> lista = new ArrayList<Ticket>();
        for (int i = 0; i < 6; i++) {
            lista.add(new Ticket(i, "Ticket" +
Integer.toString(i),
                                "Essa é a descrição do ticket " +
Integer.toString(i), 50));
        }

        return lista;
    }

    public Ticket getTicketDetails(int id) {
        return new Ticket(id, "Ticket " + Integer.toString(id),
                            "Essa é a descrição do ticket " +
Integer.toString(id),
                            id * 5 + 10);
    }

    public List<Story> getStories(Context context) {
        StoriesData stories = new StoriesData(context);
        return stories.getCurrentIterationStories();
    }

    public List<Story> searchStories(String text) {
        List<Story> lista = new ArrayList<Story>();
        for (int i = 1; i < 6; i++) {
            lista.add(new Story(i, "Estória" +
Integer.toString(i),
                                "Essa é a descrição da estória " +
Integer.toString(i),
                                "Bruno"));
        }

        return lista;
    }

    public enum StatusTickets {
        TODOS, TODO, DOING, DONE;
    }

}

```

```

-----ConfData.java-----
package data.android.apptcc;

```

```

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;

```

```

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class ConfData {

    private static final String DB_NAME = "conf.db";
    private static final int DB_VERSION = 1;
    private static final String TABLE_CONF = "conf";

    public static final String CREATE = "CREATE TABLE conf ( "
        + "    url    VARCHAR( 255 ) " + " ); ";

    private Context context;
    private SQLiteDatabase db;

    public String getURL() {
        Cursor cursor = this.db.rawQuery("SELECT *from conf",
null);

        if (cursor.getCount() > 0) {
            cursor.moveToFirst();
            return cursor.getString(0);
        } else {
            return "http://10.0.2.2:3000/";
        }
    }

    public ConfData(Context context) {
        this.context = context;

        OpenHelper openHelper = new OpenHelper(this.context);

        this.db = openHelper.getWritableDatabase();
    }

    public void insert(ContentValues values) {
        db.insert(TABLE_CONF, null, values);
    }

    public void deleteAll() {
        this.db.delete(TABLE_CONF, null, null);
    }

    private static class OpenHelper extends SQLiteOpenHelper {

        OpenHelper(Context context) {
            super(context, DB_NAME, null, DB_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE);
        }
    }
}

```

```

        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
            }
        }
    }
}

```

```

-----IterationData.java-----
package data.android.apptcc;

```

```

import java.text.SimpleDateFormat;
import java.util.Date;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import model.android.apptcc.Iteration;

public class IterationData {

    private static final String DB_NAME = "iteration.db";
    private static final int DB_VERSION = 1;
    private static final String TABLE = "iteration";

    public static final String CREATE = "CREATE TABLE iteration ( "
        + "    id            INTEGER            PRIMARY KEY, "
        + "    title         VARCHAR( 128 ), "
        + "    startDate    VARCHAR( 30 ), " + "    endDate
VARCHAR( 30 ), "
        + "    goal         VARCHAR( 255 ) " + "); ";

    private Context context;
    private SQLiteDatabase db;

    public IterationData(Context context) {
        this.context = context;

        OpenHelper openHelper = new OpenHelper(this.context);

        this.db = openHelper.getWritableDatabase();
    }

    public Iteration getCurrentIterationData() {
        Cursor cursor = this.db.rawQuery(
            "SELECT id, title, startDate, endDate, goal
FROM iteration",
            null);
        cursor.moveToFirst();
    }
}

```

```

        return new Iteration(cursor.getInt(0), cursor.getString(1),
            cursor.getString(2), cursor.getString(3),
            cursor.getString(4));
    }

    public void insert(int id, String title, Date startDate, Date
endDate,
        String goal) {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/mm/yyyy");

        ContentValues values = new ContentValues();
        values.put("id", 1);
        values.put("title", title);
        values.put("startDate", sdf.format(startDate));
        values.put("endDate", sdf.format(endDate));
        values.put("goal", goal);
        db.insert(TABLE, null, values);
    }

    public void deleteAll() {
        this.db.delete(TABLE, null, null);
    }

    private static classOpenHelper extends SQLiteOpenHelper {

        OpenHelper(Context context) {
            super(context, DB_NAME, null, DB_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
        }
    }
}

```

```

-----StoriesData.java-----
package data.android.apptcc;

import java.util.ArrayList;
import java.util.List;

import controller.android.apptcc.CurrentIteration;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;

```

```

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import model.android.apptcc.Story;

public class StoriesData {

    private static final String DB_NAME = "stories.db";
    private static final int DB_VERSION = 1;
    private static final String TABLE_STORIES = "stories";

    public static final String CREATE = "CREATE TABLE stories ( "
        + "      id INT PRIMARY KEY, " + "      title
VARCHAR( 128 ), "
        + "      description TEXT, " + "      owner
VARCHAR( 128 ), "
        + "      iterationID INT " + " ); ";

    private Context context;
    private SQLiteDatabase db;

    public StoriesData(Context context) {
        this.context = context;

        OpenHelper openHelper = new OpenHelper(this.context);

        this.db = openHelper.getWritableDatabase();
    }

    public List<Story> getCurrentIterationStories() {
        int currentIterationId = new
CurrentIteration().getIterationDetails(
        context).getId();

        Cursor cursor = this.db.rawQuery(
            "select *from stories where iterationID = "
            +
Integer.toString(currentIterationId), null);
        cursor.moveToFirst();

        List<Story> lista = new ArrayList<Story>();
        for (int i = 0; i < cursor.getCount(); i++) {
            lista.add(new Story(cursor.getInt(0),
cursor.getString(1), cursor
            .getString(2), cursor.getString(3)));
            cursor.moveToNext();
        }

        return lista;
    }

    public List<Story> getProductBacklogStories() {
        Cursor cursor = this.db.rawQuery("select *from stories",
null);

```

```

        cursor.moveToFirst();

        List<Story> lista = new ArrayList<Story>();
        for (int i = 0; i < cursor.getCount(); i++) {
            lista.add(new Story(cursor.getInt(0),
cursor.getString(1), cursor
                        .getString(2), cursor.getString(3)));
            cursor.moveToNext();
        }

        return lista;
    }

    public Story getStoryData(int idStory) {
        Cursor cursor = this.db.rawQuery("select *from stories
where id = "
            + Integer.toString(idStory), null);
        cursor.moveToFirst();

        return new Story(cursor.getInt(0), cursor.getString(1),
            cursor.getString(2), cursor.getString(3));
    }

    public void insert(ContentValues values) {
        db.insert(TABLE_STORIES, null, values);
    }

    public void deleteAll() {
        this.db.delete(TABLE_STORIES, null, null);
    }

    private static class OpenHelper extends SQLiteOpenHelper {

        OpenHelper(Context context) {
            super(context, DB_NAME, null, DB_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
        }
    }
}

```

```

-----TicketsData.java-----
package data.android.apptcc;

import java.util.ArrayList;

```

```

import java.util.List;

import model.android.apptcc.Ticket;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import controller.android.apptcc.Taskboard.StatusTickets;

public class TicketsData {

    private static final String DB_NAME = "tickets.db";
    private static final int DB_VERSION = 1;
    private static final String TABLE_TICKETS = "tickets";

    public static final String CREATE = "CREATE TABLE tickets ( "
        + "      id          PRIMARY KEY, "
        + "      title       VARCHAR( 128 ), " + "
description TEXT, "
        + "      effort     INT, " + "      status      INT " +
    "); ";

    private Context context;
    private SQLiteDatabase db;

    public TicketsData(Context context) {
        this.context = context;

        OpenHelper openHelper = new OpenHelper(this.context);

        this.db = openHelper.getWritableDatabase();
    }

    public List<Ticket> getTickets(StatusTickets status) {

        int istatus = -1;
        switch (status) {
            case TODO: {
                istatus = 0;
            }
                break;
            case DOING: {
                istatus = 1;
            }
                break;
            case DONE: {
                istatus = 2;
            }
                break;
            case TODOS: {
                istatus = 3;
            }
        }
    }
}

```

```

        break;
    }

    Cursor cursor = null;
    if (istatus == 3) {
        cursor = this.db.rawQuery("select *from tickets",
null);
    } else {
        cursor = this.db.rawQuery("select *from tickets where
status = "
                                + Integer.toString(istatus), null);
    }
    cursor.moveToFirst();

    List<Ticket> lista = new ArrayList<Ticket>();
    for (int i = 0; i < cursor.getCount(); i++) {
        lista.add(new Ticket(cursor.getInt(0),
cursor.getString(1), cursor
                                .getString(2), cursor.getInt(3)));
        cursor.moveToNext();
    }

    return lista;
}

public Ticket getTicketData(int idTicket) {
    Cursor cursor = this.db.rawQuery("select *from tickets
where id = "
                                    + Integer.toString(idTicket), null);
    cursor.moveToFirst();

    Ticket result = new Ticket(cursor.getInt(0),
cursor.getString(1),
                                cursor.getString(2), cursor.getInt(3));
    switch (cursor.getInt(4)) {
    case 0: {
        result.setStatus(StatusTickets.TODO);
    }
        break;
    case 1: {
        result.setStatus(StatusTickets.DOING);
    }
        break;
    case 2: {
        result.setStatus(StatusTickets.DONE);
    }
        break;
    }

    return result;
}

public void insert(ContentValues values) {

```

```

        db.insert(TABLE_TICKETS, null, values);
    }

    public void deleteAll() {
        this.db.delete(TABLE_TICKETS, null, null);
    }

    private static class OpenHelper extends SQLiteOpenHelper {

        OpenHelper(Context context) {
            super(context, DB_NAME, null, DB_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
        }
    }
}

-----Iteration.java-----
package model.android.apptcc;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Iteration {

    private int id;
    private String titulo;
    private Date dataInicio;
    private Date dataFim;
    private String objetivo;

    public Iteration() {

    }

    public Iteration(int id, String titulo, Date dataInicio, Date
dataFim,
        String objetivo) {
        super();
        this.id = id;
        this.titulo = titulo;
        this.dataInicio = dataInicio;
        this.dataFim = dataFim;
        this.objetivo = objetivo;
    }
}

```

```

    }

    public Iteration(int id, String titulo, String dataInicio,
String dataFim,
        String objetivo) {
        super();

        SimpleDateFormat sdf = new SimpleDateFormat("dd/mm/yyyy");
        this.id = id;
        this.titulo = titulo;
        try {
            this.dataInicio = sdf.parse(dataInicio);
            this.dataFim = sdf.parse(dataFim);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        this.objetivo = objetivo;
    }

    public int getId() {
        return id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public Date getDataInicio() {
        return dataInicio;
    }

    public void setDataInicio(Date dataInicio) {
        this.dataInicio = dataInicio;
    }

    public Date getDataFim() {
        return dataFim;
    }

    public void setDataFim(Date dataFim) {
        this.dataFim = dataFim;
    }

    public String getObjetivo() {
        return objetivo;
    }

    public void setObjetivo(String objetivo) {
        this.objetivo = objetivo;
    }

```

```

    }

}

-----Story.java-----
package model.android.apptcc;

public class Story {

    private int id;
    private String titulo;
    private String descricao;
    private String solicitador;

    public Story(int id, String titulo, String descricao, String
solicitador) {
        super();
        this.id = id;
        this.titulo = titulo;
        this.descricao = descricao;
        this.solicitador = solicitador;
    }

    public Story() {

    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public String getSolicitador() {
        return solicitador;
    }

    public void setSolicitador(String solicitador) {
        this.solicitador = solicitador;
    }

}

```

```

-----Ticket.java-----
package model.android.apptcc;

import controller.android.apptcc.Taskboard.StatusTickets;

public class Ticket {

    private int id;
    private String titulo;
    private String descricao;
    private int esforco;
    private StatusTickets status;
    private boolean categoria;

    public Ticket() {

    }

    public Ticket(int id, String titulo, String descricao, int
esforco) {
        super();
        this.id = id;
        this.titulo = titulo;
        this.descricao = descricao;
        this.esforco = esforco;
        this.categoria = false;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public int getEsforco() {
        return esforco;
    }

    public void setEsforco(int esforco) {
        this.esforco = esforco;
    }

    public boolean isCategoria() {

```

```
        return categoria;
    }

    public void setCategoria(boolean categoria) {
        this.categoria = categoria;
    }

    public StatusTickets getStatus() {
        return status;
    }

    public void setStatus(StatusTickets status) {
        this.status = status;
    }
}
```
