UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ROBERSON RECH

SOFTWARE PARA CONTROLE DE PROJETOS UTILIZANDO A METODOLOGIA SCRUM

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

ROBERSON RECH

SOFTWARE PARA CONTROLE DE PROJETOS UTILIZANDO A METODOLOGIA SCRUM

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito para obtenção do título de Tecnólogo.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

SOFTWARE PARA CONTROLE DE PROJETOS UTILIZANDO A METODOLOGIA SCRUM

POR

ROBERSON RECH

Este trabalho de conclusão de curso foi apresentado no dia 24 de novembro de 2017, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Prof. MSc. Vinicius Pegorini
Orientador

Profª Drª Beatriz Terezinha Borsoi

Profª Esp. Adriana Ariati

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

RECH, Roberson. Software para controle de projetos utilizando a metodologia Scrum. 2017. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2017.

O gerenciamento de projetos para o desenvolvimento de software impõe algumas dificuldades a seus gestores. Para auxiliar, eles podem contar com a Engenharia de Software, que proporciona metodologias capazes de auxiliar na gestão desses projetos. Uma dessas metodologias é o *Scrum*, que é o assunto principal deste trabalho. Essa metodologia proporciona aos gestores de projetos um envolvimento maior com o cliente e aos participantes do projeto maior liberdade de trabalho, sem impor processos e com pouca documentação a ser feita. Este trabalho apresenta o desenvolvimento de um sistema *desktop* para gerenciar projetos por meio da metodologia *Scrum*. Os processos envolvidos na gestão de projetos utilizando o *Scrum* são abordados no decorrer do trabalho, descrevendo suas funções e objetivos. Por fim, são apresentados os resultados do estudo da aplicação da metodologia *Scrum* com o objetivo de mostrar que é possível gerenciar projetos de softwares com esta metodologia, que pode ser utilizada desde projetos mais simples até os mais complexos e inovadores.

Palavras-chave: Engenharia de Software. Metodologia Ágil. *Scrum*.

ABSTRACT

RECH, Roberson. Project Management Software for Scrum. 2017. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2017.

The management of software development projects imposes some difficulties for its managers. To improve the experience with software development we have the Software Engineering, which provides us with many methodologies able to assist in the management of these projects, one of these methodologies is Scrum, which will be the main subject of this work, this methodology gives project managers greater involvement with the client and gives the participants of this project a greater freedom of work without imposing processes and with little documentation to be done. This work presents the development of a desktop system capable of managing projects through the Scrum methodology. The processes involved in project management through Scrum are addressed during the work, describing its functions and objectives. Finally, the results of the study of the application of the Scrum methodology are presented with the objective of demonstrating that it is possible to manage software projects with this methodology, which is suitable to be used from the simplest to the most complex and innovative projects.

Keywords: Software Engineering. Agile Methodology. Scrum.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

ADM	*Advanced Development Methods*
CASE	*Computer-Aaided Ssoftware Eengineering*
DSDM	*Dynamic Systems Development Method*
FDD	*Feature Driven Development*
IDE	*Integrated Development Environment*
MDI	*Multiple-Document Interface*
OMG	*Object Management Group*
OpenUP	*Open Unified Process*
POO	Programação Orientada a Objetos
RF	Requisito Funcional
RNF	Requisito não Funcional
VCL	*Visual Component Library*
WIP	*Work in Progress*
XP	*eXtreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

No desenvolvimento de novos softwares costuma-se dar início a novos projetos. Esses, segundo Pressman (2011), ajudam a determinar princípios, conceitos e práticas que serão utilizadas durante o desenvolvimento do projeto para que se possa garantir a qualidade de software, que é essencial para se obter um produto final resultante de um projeto com boa qualidade.

Por meio da Engenharia de Software foram utilizadas durante muito tempo metodologias referenciadas no modelo em cascata, que tem sua abordagem de maneira sequencial, e, por isso é necessário que haja o levantamento de todos os requisitos necessários para a obtenção do resultado final do projeto antes mesmo de se iniciar o desenvolvimento do software (SILVA; SOUZA NETO, 2015).

Porém, no desenvolvimento de software dificilmente é possível prever todos os requisitos antes de se obter um *feedback* do cliente, com isso, pode-se trabalhar com as Metodologias Ágeis, que têm como um de seus princípios obter *feedback* utilizando protótipos para obter a validação dos requisitos (SILVA; SOUZA NETO NETO, 2015).

Uma dessas metodologias é o *Scrum*, o qual possui processos leves e fáceis de aprender, com o objetivo de gerenciar o desenvolvimento de software deixando de lado o tradicional método cascata, adotando uma prática iterativa e incremental que exige poucos artefatos (documentações de requisitos e modelagem, especificações e diagramas), mantendo seu foco no gerenciamento do projeto (MARTINS, 2007).

Neste trabalho será desenvolvido um software para gestão de projetos. Esta gestão terá como base para a metodologia *Scrum*, que segundo Schwaber e

Sutherland (2016) é utilizada para tratar e resolver problemas complexos e adaptativos empregando várias técnicas e processos.

1.2 OBJETIVOS

Neste subcapítulo serão apresentados os objetivos deste trabalho.

1.2.1 OBJETIVO GERAL

Desenvolver um sistema de controle de projetos com mecanismos de gestão e de planejamento baseados na metodologia *Scrum*.

1.2.2 OBJETIVOS ESPECÍFICOS

- Facilitar a gestão de projetos da empresa por meio da metodologia ágil.
- Controlar requisitos.
- Controlar *Product Backlog*.
- Controlar *Sprints*.
- Controlar *Sprint Backlog*.
- Oferecer uma agenda para a gestão de compromissos do usuário.

1.3 JUSTIFICATIVA

Dado o número de tecnologias existentes nos dias de hoje, se tornou muito difícil prever como um sistema evoluirá com o tempo. Por isso, em muitas situações não é possível definir plenamente todos os requisitos de um sistema. Isso pode acabar implicando mudanças dos requisitos. Contudo, para os modelos clássicos de gestão de projetos da engenharia de software essas mudanças acabam exigindo

uma grande demanda de trabalho, resultando em um aumento dos custos do projeto e atrasos nos prazos de entrega do sistema (PRESSMAN, 2011).

Uma solução para essas constantes mudanças nos requisitos é a utilização das metodologias ágeis, que segundo Sommerville (2011), são capazes de lidar com essas situações e permite que o sistema seja rapidamente desenvolvido e entregue ao cliente. Uma das metodologias ágeis mais utilizadas é *Scrum*, que é o tema deste trabalho, tendo como resultado uma ferramenta para suporte à utilização deste método permitindo que seja acompanhado todo o processo do desenvolvimento do software.

1.4 ESTRUTURA DO TRABALHO

O trabalho é composto por cinco seções que estão distribuídas da seguinte maneira: no primeiro capitulo é feita a introdução e contextualização ao tema no qual este trabalho está situado, juntamente estão a justificativa e os objetivos a serem atingidos. No segundo capítulo é abordado detalhadamente os conceitos da Metodologia Ágil *Scrum*. No terceiro capítulo são apresentados os materiais e os métodos que foram utilizados para o desenvolvimento deste trabalho. No quarto capítulo é feita a apresentação do produto obtido como resultado da conclusão deste trabalho. Por fim, no quinto capítulo, está a conclusão obtida com desenvolvimento deste trabalho.

2. METODOLOGIAS ÁGEIS

Segundo Sommerville (2011), na década de 1980 e início de 1990, a visão era de que os melhores softwares eram produzidos por meio de um planejamento cuidadoso do projeto, com qualidade de segurança formalizada, com ferramentas *Computer-Aided Software Engineering (CASE*) apoiando o projeto e análises, além de um processo de desenvolvimento rigoroso e controlado. Essa percepção veio da comunidade de Engenharia de Software responsável por grandes sistemas, como por exemplo, governamentais e aeroespaciais. No entanto, quando esta abordagem é aplicada a sistemas corporativos de pequeno e médio porte, acaba-se gastando mais tempo com a documentação do que com o desenvolvimento e os testes do sistema.

Por isso, na década de 1990, com a insatisfação das abordagens tradicionais da Engenharia de Software, levou um grupo de desenvolvedores a propor os novos 'métodos ágeis' que permitiram a equipe de desenvolvimento focar no software em vez de sua concepção e documentação. Por isso, as metodologias ágeis têm como objetivo reduzir a burocracia do projeto e, por isso, são mais adequadas para aplicativos aos quais os requisitos podem mudar durante o processo de desenvolvimento, que precisam ser entregues rapidamente, em funcionamento, para que em seguida, possam ser feitas mudanças com os possíveis novos requisitos.

A filosofia por trás dos métodos ágeis é refletida no manifesto ágil, que foi acordado por muitos dos principais desenvolvedores desses métodos. Esse manifesto afirma:

> Estamos descobrindo melhores maneiras de desenvolver softwares, fazendo-o e ajudando outros a fazê-lo. Através desse trabalho, valorizamos mais:
> - Indivíduos e interação do que processos e ferramentas;
> - Software em funcionamento do que documentação abrangente;
> - Colaboração do cliente do que negociação de contrato;
> - Respostas a mudanças do que seguir um plano;
>
> Ou seja, embora os itens à direita sejam importantes, valorizamos mais os que estão à esquerda (SOMMERVILLE, 2011).

Além desses valores, os métodos ágeis também compartilham um conjunto de princípios baseados no manifesto ágil (SOMMERVILLE, 2011):

1. Envolvimento dos clientes: os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
2. Entrega incremental: o software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
3. Pessoas, não processos: as habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
4. Aceitar as mudanças: deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
5. Manter a simplicidade: focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Segundo Sommerville (2011), os métodos ágeis são todos baseados na noção de desenvolvimento e entrega incremental propondo diferentes processos, mas compartilham um conjunto de princípios, com base no Manifesto Ágil, por isso tem muito em comum.

A abordagem *Scrum* foi adotada neste trabalho por estar mais focada em agregar valor para o cliente e em gerenciar os riscos, fornecendo um ambiente seguro e também por que ela pode ser utilizada na gestão do projeto aliada a outras metodologias ágeis mais focadas no desenvolvimento como *eXtreme Programming* (XP), *Feature Driven Development* (FDD), *Open Unified Process* (OpenUP), *Dynamic Systems Development Method* (DSDM) e Crystal.

2.1 SCRUM

Nesta seção são abordados os processos que envolvem a metodologia ágil *Scrum*.

2.1.1 ORIGEM

Segundo Pham (2011), o termo *Scrum* surgiu em um artigo publicado por Hirotaka Takeuchi e Ikujiro Nonaka na *Harvad Business Review* de 1986. Mais tarde, Jeff Sutherland, então vice-presidente de engenharia da Easel, Inc., queria um processo semelhante ao *Scrum*, em que, ao final de iterações curtas pudesse ver a demonstração de código funcional e não apenas documentos.

Neste mesmo período, Ken Schwaber, começou uma pesquisa sobre como outras empresas vendedoras de software e bem-sucedidas construíam seus softwares para descobrir como ele poderia ajudar sua empresa, *Advanced Development Methods* (ADM), a melhorar seu processo com o objetivo de aumentar a produtividade das equipes.

Em 1995, a pedido da *Object Management Group* (OMG), Jeff e Ken trabalharam em conjunto para resumir o que haviam aprendido ao longo dos anos, resultando em uma nova metodologia chamada de *Scrum*, descrita em um artigo de Shwaber, "*Scrum and the perfect storm*" (Pham, 2011).

2.1.2 TEORIA DO SCRUM

Segundo seus criadores, Schwaber e Sutherland (2016), o *Scrum* emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. Fundamentado em teorias empíricas de controle de processos apoiandas em três pilares:

1. Transparência, os aspectos significativos do processo devem ser visíveis aos responsáveis. Para que estes compartilhem um entendimento em comum, requer aspectos definidos por um padrão.
2. Inspeção, os usuários *Scrum*, devem inspecionar os artefatos e o progresso procurando por variações, porém, a execução destas inspeções não deve atrapalhar na execução das tarefas.
3. Adaptação, se forem encontrados na inspeção um ou mais aspectos de um processo fora dos limites aceitáveis, este deverá ser ajustado o mais breve possível para minimizar os desvios.

2.1.3 THE SCRUM TEAM

O *Scrum Team* é composto pelo *Product Owner, Development Team* e o *Scrum Master*. Essa metodologia foi projetada para aperfeiçoar a flexibilidade, a criatividade e a produtividade, para isso, o time deverá ser auto-organizável, sendo possível escolher a melhor forma para completar seu trabalho em vez de serem dirigidos por outros membros de fora da equipe. E o time deverá ser multifuncional, para que todas as competências necessárias para se completar um trabalho possam ser feitas sem depender de outros que não fazem parte da equipe (SCHWABER; SUTHERLAND, 2016).

2.1.3.1 THE PRODUCT OWNER

Segundo Schwaber e Sutherland (2016), o *Product Owner* deverá ser capaz de maximizar o valor de seu produto e do trabalho feito pelo time de desenvolvimento, sendo a única pessoa responsável por gerenciar o *Product Backlog*[1], que inclui:

- Expressar claramente os itens do *Product Backlog*;
- Ordenar os itens do *Product Backlog* para alcançar melhor as metas e missões;
- Garantir o valor do trabalho realizado pelo *Development Team*;
- Garantir que o *Product Backlog* seja visível, transparente, claro para todos, e mostrar o que o *Scrum Team* vai trabalhar a seguir; e,
- Garantir que o *Development Team* entenda os itens do *Product Backlog* no nível necessário.

Este gerenciamento pode ser feito pelo próprio *Product Owner* ou delegado para o *Development Team*, mas a responsabilidade continua sendo toda do *Product Owner*. Portanto, se necessária alguma alteração no conteúdo ou prioridades dos itens do *Product Backlog*, é ele quem deve ser convencido e, para que se tenha sucesso, suas decisões devem ser respeitadas, ou seja, somente ele deverá falar para o *Development Team* sobre alterações e o *Development Team* não deverá agir sobre ordens de outras pessoas.

[1] *Product Backlog* é uma listagem de requisitos necessários para o desenvolvimento do produto, no capítulo 2.1.5.1 é feita sua apresentação.

2.1.3.2 THE DEVELOPMENT TEAM

Segundo Schwaber e Sutherland (2016), os *Development Teams* são estruturados e organizados para que sejam capazes de gerenciar seu próprio trabalho e de entregar, ao final de cada *Sprint*[1], um produto usável e que agregue valores ao produto final a cada incremento.

O *Development Team* do *Scrum* deve conter de três a nove membros, sendo possível, assim, manter-se ágil o suficiente para completar os trabalhos da *Sprint* dentro dos limites. Equipes menores podem encontrar restrições de habilidades durante a *Sprint*, tornando o time incapaz de entregar um incremento potencialmente utilizável. Equipes maiores requerem muita coordenação e, por isso, geram complexidade para um processo empírico gerenciar.

Os papéis de *Product Owner* e *Scrum Master* não entram nesta contagem, a menos que eles também executem o trabalho do *Product Backlog*.

2.1.3.3 THE SCRUM MASTER

Segundo Schwaber (2004), quem está neste papel tem a responsabilidade de garantir que todos do *Scrum Team* entendam e sigam as regras e práticas aplicadas do *Scrum*, ajudando também a aqueles que estão de fora do time a entender as iterações produzidas para maximizar o valor criado pelo *Scrum Team*.

O *Scrum Master* pode ajudar o *Product Owner* de várias maneiras, incluindo (SCHWABER; SUTHERLAND, 2016):

- Encontrando técnicas para o gerenciamento efetivo do *Product Backlog*;
- Claramente comunicar a visão, objetivo e itens do *Product Backlog* para o *Development Team*;
- Ensinar a *Scrum Team* a criar itens de *Product Backlog* de forma clara e concisa;
- Compreender a longo prazo o planejamento do produto no ambiente empírico;
- Compreender e praticar a agilidade; e,
- Facilitar os eventos *Scrum* conforme exigidos ou necessários.

[1] *Sprint* são iterações de períodos curtos.

O *Scrum Master* também deve ajudar o *Development Team* de várias maneiras, incluindo (SCHWABER; SUTHERLAND, 2016):

- Treinar o *Development Team* em autogerenciamento e interdisciplinaridade;
- Ensinar e liderar o *Development Team* na criação de produtos de alto valor;
- Remover impedimentos para o progresso do *Development Team*;
- Facilitar os eventos *Scrum* conforme exigidos ou necessários; e,
- Treinar o *Development Team* em ambientes organizacionais nos quais o *Scrum* não é totalmente adotado e compreendido.

O *Scrum Master* ajuda a Organização de várias maneiras, incluindo (SCHWABER; SUTHERLAND, 2016):

- Liderando e treinando a organização na adoção do *Scrum*;
- Planejando implementações *Scrum* dentro da organização;
- Ajudando funcionários e partes interessadas a compreender e tornar aplicável o *Scrum* e o desenvolvimento de produto empírico;
- Causando mudanças que aumentam a produtividade do *Scrum Team*; e,
- Trabalhando com outros *Scrum Masters* para aumentar a eficácia da aplicação do *Scrum* nas organizações.

2.1.4 SCRUM EVENTS

Segundo Schwaber e Sutherland (2016), um dos propósitos do *Scrum Events* é criar uma rotina para eliminar a possibilidade de reuniões não definidas no *Scrum*. Assim, todos os *Scrum Events* tem seu tempo máximo definido, como na *Sprint*, na qual a sua duração é fixada e não pode ser alterada. O restante dos eventos podem terminar sempre que seu objetivo for alcançado para garantir que não haja perda de tempo no processo.

Esses eventos são projetados para permitir transparência e inspeção criteriosa, sendo possível realizar algumas adaptações. A não inclusão de qualquer evento pode resultar na perda dessas propriedades.

2.1.4.1 SPRINT

Para Schwaber e Sutherland (2016) a *Sprint* é o principal evento do *Scrum*, pode durar até 30 dias e ao seu término são gerados incrementos potencialmente utilizáveis do produto. Uma nova *Sprint* inicia após a conclusão da *Sprint* atual.

Uma *Sprint* é composta por uma reunião de planejamento reuniões diárias, trabalho do desenvolvimento, uma revisão e a sua retrospectiva. Em sua definição, está o que deve ser construído e, durante sua execução, o escopo é definido entre o *Product Owner* e o *Development Team* com esclarecimentos e renegociações, desde que isso não afete os objetivos da *Sprint*.

Sprints devem ser limitadas a um mês corrido, mais tempo que isto pode aumentar a complexidade fazendo com que os riscos sejam maiores e a sua definição do que deve ser construído pode mudar.

Devido a sua curta duração é raro ocorrer cancelamento da *Sprint*, mas se ocorrer será feito somente pelo *Product Owner*, podendo ser feito quando seus objetivos se tornam obsoletos devido às mudanças nas condições de mercado ou das tecnologias. Quando é cancelada, os itens do *Product Backlog* que já estiverem prontos são revisados pelo *Product Owner* e os itens incompletos são estimados novamente e voltam ao *Product Backlog*.

2.1.4.2 SPRINT PLANNING

De acordo com Schwaber e Sutherland (2016), a *Sprint Planning* é uma reunião no qual é realizado o planejamento de uma *Sprint*, determinando qual será seu resultado e como este será atingido. Para *Sprint* com duração de um mês, esta reunião deverá durar até oito horas e para *Sprint* menores, o tempo deverá ser proporcional. O *Scrum Master* deverá garantir que o evento esteja dentro dos limites de tempo e que seus participantes entendam seu propósito.

Na primeira parte da reunião, o *Development Team* trabalha para prever as funcionalidades que serão desenvolvidas durante a *Sprint*, o objetivo e os itens do *Product Backlog* que ajudarão a atingi-lo são debatidos pelo *Product Owner*, o número de itens selecionados do *Product Backlog* é feito pelo *Development Team*,

contando com a ajuda do *Product Owner* para esclarecer dúvidas dos itens e ajudando nas decisões conflituosas, já que somente o *Development Team* poderá avaliar o que poderá ser completado ao longo da próxima *Sprint*. Após prever os itens do *Product Backlog*, o *Scrum Team* determina a meta da *Sprint*, que é o objetivo que será conhecido dentro da *Sprint* por meio da implementação do *Product Backlog*, este fornece uma orientação ao *Development Team* sobre o porquê construir o incremento.

Na segunda parte da reunião, o *Development Team* decide como será feita a implementação dos itens selecionados do *Product Backlog*. Junto com o plano de entrega estes itens selecionados são denominados de *Sprint Backlog*, determinando o trabalho necessário para converter estes itens em um incremento do produto. O *Development Team,* então, se auto-organiza para realizar todo o trabalho do *Sprint Backlog* tanto durante o planejamento quanto no que for necessário durante o decorrer da *Sprint*.

Caso o *Development Team* determine que há excesso ou falta de trabalho, juntamente com o *Product Owner* é feita renegociação dos itens selecionados do *Product Backlog*. Ao final o time deverá ser capaz de explicar ao *Product Owner* e o *Scrum Master* como pretende trabalhar para completar o objetivo da *Sprint*.

2.1.4.3 DAILY SCRUM

Segundo Schwaber e Sutherland (2016), a *Daily Scrum* é uma reunião de até 15 minutos, realizada sempre no mesmo local todos os dias para reduzir a complexidade, no qual os membros do *Development Team* devem repassar entre eles o trabalho feito desde a última *Daily Scrum* e o trabalho que será feito até a próxima *Daily Scrum* respondendo a três perguntas:

- O que foi feito desde a última reunião?
- O que será feito até a próxima reunião?
- Há algo impedindo a conclusão do trabalho?

Com isto, é feita uma avaliação do progresso da *Sprint* em direção ao seu objetivo, e para isso, o *Development Team* deve entender como trabalhar em conjunto, como um time auto organizado. Após a *Daily Scrum*, o *Development Team*

pode se reunir para realizar discussões detalhadas, adaptações e planejamentos futuros para o restante do trabalho da *Sprint*.

O *Scrum Master* deverá assegurar que a reunião seja realizada dentro do prazo de 15 minutos, reforçando a regra de que somente o *Development Team* deve participar da reunião.

2.1.4.4 SPRINT REVIEW

A *Sprint Review* é uma reunião executada ao final da *Sprint* na qual o *Development Team* e as partes interessadas participam. Nela é apresentado tudo o que foi realizado na *Sprint*, com as possíveis mudanças, e o que foi entregue no incremento. Com base nisso, os participantes podem colaborar sobre as próximas adaptações a serem realizadas no próximo incremento (SCHWABER; SUTHERLAND, 2016).

A reunião deverá durar até 4 horas para uma *Sprint* de duração de um mês, para *Sprints* de menor duração o tempo deverá ser proporcional. O *Scrum Master* é responsável por garantir este evento e que os participantes entendam seu objetivo (SCHWABER; SUTHERLAND, 2016).

Na reunião de *Sprint Review* ocorrem as seguintes etapas: o *Product Owner* identifica quais itens do *Product Backlog* foram realizados e quais não foram, em seguida, o *Development Team* discute o que foi bem durante a *Sprint*, quais foram os problemas e como eles foram resolvidos. Após esse resumo, todos os participantes colaboram para definir quais serão as adaptações mais importantes que devem ser feitas, fornecendo dados para a *Sprint Planning* da próxima *Sprint* (SCHWABER; SUTHERLAND, 2016).

O resultado da reunião é *Product Backlog* revisado para ser utilizado para as próximas *Sprints* podendo ser ajustado completamente para atender novas oportunidades (SCHWABER; SUTHERLAND, 2016).

2.1.4.5 SPRINT RETROSPECTIVE

A *Sprint Retrospective* é um evento que tem duração de aproximadamente 3 horas para uma *Sprint* que dura um mês, e para *Sprint* de menor duração o tempo deverá ser proporcional. A *Sprint Retrospective* é realizada visando identificar os principais itens que foram bem e para que o *Scrum Team* possa inspecionar a si próprio e determinar potenciais melhorias a serem aplicadas na próxima *Sprint* através da criação de planos (SCHWABER; SUTHERLAND, 2016).

O *Scrum Master* deverá garantir que este evento ocorra e que os participantes entendam seu propósito, ele também pode participar desta reunião como um membro, devido a sua importância e responsabilidade pelo processo *Scrum*. O resultado da *Sprint Retrospective* é a identificação de melhorias que possam ser aplicadas não somente nas próximas *Sprint*, mas a qualquer momento de seu decorrer (SCHWABER; SUTHERLAND, 2016).

2.1.5 SCRUM ARTIFACTS

Os *Scrums Artifacts* são projetados para fornecer transparência, eles representam trabalhos e valores das informações chaves de modo que todos tenham um mesmo entendimento dos *Scrums Artifacts* (SCHWABER; SUTHERLAND, 2016).

2.1.5.1 PRODUCT BACKLOG

O *Product Backlog* é uma listagem que contém tudo o que é necessário para o produto, ela é gerida pelo *Product Owner* que fornecerá seu conteúdo, disponibilidade e ordenação. O *Product Backlog* muda constantemente para que o produto possa ter o que é preciso para ser mais apropriado, competitivo e útil no

ambiente no qual ele será utilizado, portanto, enquanto o produto existir, existirá o *Product Backlog* (SCHWABER; SUTHERLAND, 2016).

Nesta lista podem ser encontradas características, funções, requisitos, melhorias e correções que devem ser feitas no produto para suas futuras versões, os itens possuem os atributos de descrição, ordem, estimativa e valor. Quando múltiplos *Scrum Teams* trabalham em um mesmo produto, é utilizado um atributo no *Product Backlog* para fazer um agrupamento por *Scrum Team* (SCHWABER; SUTHERLAND, 2016).

O refinamento do *Product Backlog* é um processo de adicionar detalhes, estimativas e ordens aos itens, é realizado pelo *Product Owner* e o *Development Team* continuamente com análises e revisões. Os itens que tiverem ordem mais alta, estão no topo da lista, devem ser mais detalhados do que os de ordem mais baixa. Esse detalhamento ajudará a obter estimativas mais precisas. As estimativas são de responsabilidade do *Development Team*, o *Product Owner* pode ajudar no entendimento e nas decisões conflituosas, mas as pessoas que realizarão o trabalho é quem fazem a estimativa final (SCHWABER; SUTHERLAND, 2016).

2.1.5.2 SPRINT BACKLOG

O *Sprint Backlog* é uma listagem dos itens selecionados do *Product Backlog* juntamente ao plano de entrega do incremento do produto, o *Sprint Backlog* possui as funcionalidades que estarão no próximo incremento e demonstra todo o trabalho necessário que o *Development Team* terá para entregá-lo e atingir o objetivo da *Sprint* (SCHWABER; SUTHERLAND, 2016).

Mudanças na *Sprint Backlog* durante o decorrer de uma *Sprint* podem ser feitas somente pelo *Development Team* e pode surgir de entendimentos obtidos na *Daily Scrum* sobre o trabalho necessário para se alcançar o objetivo da *Sprint* (SCHWABER; SUTHERLAND, 2016).

Como o *Sprint Backlog* pertence exclusivamente ao *Development Team* e é uma imagem em tempo real do trabalho restante planejado para se completar na *Sprint*, sempre que for necessário um novo trabalho, o *Development Team* é quem adiciona este novo requisito ao *Sprint Backlog,* assim que ele for concluído é

realizado uma revisão do trabalho restante à ser realizado (SCHWABER; SUTHERLAND, 2016).

2.1.5.3 INCREMENT

O *Increment* é o resultado obtido do conjunto dos itens selecionados do *Product Backlog* que foram desenvolvidos durante a *Sprint* e os incrementos das *Sprints* anteriores. Um novo incremento deve ser entregue ao final de uma *Sprint*, e devem estar em condições de uso, independente se o *Product Owner* decidir liberá-lo ou não (SCHWABER; SUTHERLAND, 2016).

2.1.6 ARTIFACT TRANSPARENCY

A *Artifact Transparency* ajuda otimizar o valor e aumentar o controle de riscos com base na percepção existente no estado dos artefatos. O *Scrum Master* trabalha com o *Scrum Team* para organizar o aumento das transparências dos artefatos, podendo detectar transparência incompleta por meio de inspeções dos artefatos, percebendo padrões e detectando diferenças entre o esperado e o resultado (SCHWABER; SUTHERLAND, 2016).

2.2 FERRAMENTAS

Kniberg (2009) é da opinião que usar as ferramentas adequadas vai ajudar a ter sucesso, mas isso não vai garantir o sucesso. É fácil confundir sucesso/falha de um projeto com sucesso/falha da ferramenta.

Nesta seção são apresentadas brevemente as ferramentas utilizadas no suporte ao processo de monitoramento e controle do trabalho, sendo eles o *Kanban* e o gráfico de *Burndown*.

2.2.1 KANBAN

Traduzindo literalmente do japonês a palavra *Kanban*, significa cartão ou placa visível, indicando algo por meio de um sinal visual. A ideia do *Kanban* baseia-se no controle visual de agendamento sendo possível acompanhar o progresso do trabalho (OHNO, 1997).

Segundo Gaither e Fraizer (2001), o *Kanban* é um sistema simples de controle de produção e planejamento. A metodologia possui como principais premissas: visualizar o fluxo de trabalho, limitar o trabalho em progresso - *Work in Progress* (WIP) - e medir o tempo da tarefa (IKONEN, 2010).

Cada etapa do processo produtivo é representada por uma coluna e cada coluna representa um estado de fluxo de trabalho e indicam onde cada tarefa está neste fluxo. Pries (2010) acredita que o quadro de três colunas é simples e eficaz, pois, por meio dele é possível identificar o que está para fazer, sendo feito e já feito. Porém, nada impede que outras colunas possam ser adicionadas e combinadas (KNIBERG, 2009).

Ao usar o *Kanban* é esperado que se consiga visualizar quais práticas estão sendo positivas e quais estão sendo negativas e isso conduzirá a mudanças nas práticas atuais de trabalho (GHISI, 2012).

2.2.2 GRÁFICO BURNDOWN

O gráfico de *Burndown* representa a conclusão das tarefas em função do tempo, exibindo o acumulado das horas a fazer e em seguida as horas efetivamente consumidas *versus* as horas planejadas, para representar o consumo das horas previstas (PRIES, 2010).

O gráfico de *Burndown* mostra dois eixos, o eixo X mostra o tempo decorrido e o eixo Y o tempo restante, podendo ser em horas ou qualquer outra medida de tamanho por requisitos (MAHNIC, 2012).

Mahnic (2012) explica que este gráfico costuma ser atualizado após as reuniões diárias, podendo assim acompanhar a cada dia a evolução da equipe e tomar as devidas medidas caso algum problema seja encontrado.

3 MATERIAIS E MÉTODO

Neste capítulo, serão apresentados os materiais e os métodos utilizados para alcançar o objetivo deste trabalho.

3.1 MATERIAIS

As ferramentas utilizadas para o desenvolvimento desse projeto estão listadas no Quadro 1.

Ferramenta / Tecnologia	Versão	Finalidade
Astah	7.1.0	Modelagem UML.
Firebird	3.0	Banco de dados.
IB Expert	12.21.1	Gerenciador/Modelagem de banco de dados.
Delphi XE	10.2	Ambiente de desenvolvimento.
FastReport	5	Ferramenta para geração de relatórios
TeeChart Std	v2017	Ferramenta para geração de gráficos
dbExpress	2017	Driver para comunicação com banco de dados Firebird

Quadro 1 - Lista de ferramentas e tecnologias

3.2 MÉTODO

O modelo de processo de desenvolvimento de software que foi utilizado está baseado na metodologia de ciclo de vida proposta por Sommerville (2011). Nessa metodologia é feito um planejamento por meio das atividades de análise e definição de requisitos, projeto de sistema e software, implementação e teste unitário, integração e teste de sistema, operação e manutenção. A seguir as atividades são definidas para este trabalho:

a) Análise e Definição de Requisitos – A definição dos requisitos foi realizada por meio de pesquisas a sistemas bem-sucedidos, de propósitos semelhantes ao aqui proposto. Os resultados dessas pesquisas foram complementados por pesquisas em bibliografias de autores conhecidos na área de Engenharia de Software e também ao material oficial da metodologia *Scrum* proporcionado por seus criadores. Os requisitos levantados são os necessários para se obter um software de gerenciamento de projetos com uso da metodologia *Scrum*. Após levantar os requisitos, foi realizada a análise dos requisitos definindo sua estrutura e as formas de interação com o usuário.

b) Projeto de Sistema e Software – Neste processo foram alocados os requisitos por meio da definição da arquitetura geral do sistema de software, também foram realizados a identificação e a descrição das abstrações fundamenteis do sistema e seus relacionamentos.

c) Implementação – A implementação foi realizada por meio da IDE Embarcadero Delphi juntamente com a linguagem Delphi. Para definir o leiaute dos formulários foi utilizada a biblioteca *Visual Component Library* (VCL)[1] em combinação com o padrão *Multiple-document interface* (MDI)[2], o que permite que sejam abertos vários formulários em uma única aplicação. Para a comunicação com o banco de dados Firebird foi utilizado o *framework dbExpress*[3]. Para a criação dos gráficos foi utilizado o *framework TeeChart Std*[4]*.* Para a geração de relatórios do sistema foi utilizado o framework *FastReport*[5]*.*

d) Testes – Os testes foram realizados de maneira informal pelo autor do trabalho, visando verificar o código desenvolvido e a usabilidade de interface.

[1] Disponível em: <http://docwiki.embarcadero.com/RADStudio/Tokyo/en/VCL>.
[2] Disponível em: <https://msdn.microsoft.com/pt-br/library/xyhh2e7e(v=vs.110).aspx>.
[3] Disponível em: <https://www.devart.com/dbx/>.
[4] Disponível em: <https://www.steema.com/product/vcl>.
[5] Disponível em: <http://www.fast-report.com/pt/>.

4 RESULTADOS

Neste capítulo será apresentado o resultado da realização deste trabalho que inclui a definição do escopo do sistema, a modelagem e a implementação desse sistema.

4.1 ESCOPO DO SISTEMA

O software para controle de projetos *Scrum* deve gerenciar todos os processos de um projeto de sistema de software que utilize a metodologia *Scrum* como modelo de processos para seu gerenciamento. O sistema é uma aplicação *desktop* e que pode ser instalado em qualquer periférico que possua como sistema operacional *Windows XP* ou versões superiores.

Ao iniciar um novo projeto, deverão ser definidos os participantes do projeto e os usuários do sistema que irão compor o *Scrum Team*. Após definir os seus participantes, o processo a ser seguido pelo sistema passa a ser o mesmo para projetos já existentes. Esses processos irão seguir o fluxo determinado pela metodologia *Scrum*.

O sistema auxiliará neste fluxo dispondo de uma lista dos requisitos do projeto, chamada de *Product Backlog*, uma lista dos requisitos que serão desenvolvidos na *Sprint*, conhecida como *Sprint Backlog*, para as reuniões ou eventos que são realizados durante este fluxo. O sistema também disponibilizará uma agenda pessoal para o usuário, para que ele possa gerir seus horários e reuniões, uma agenda para o projeto, para que se possa gerir seus prazos e reuniões.

O sistema tem como objetivo ser uma ferramenta para auxiliar no gerenciamento de projetos de software. Com foco em projetos que tenham como metodologia de gestão o *Scrum*, caso for utilizado para gerir outras metodologias, pode acabar não suprindo todas as suas necessidades.

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a listagem dos requisitos funcionais identificados para o sistema proposto.

Identificação	Descrição
[RF-01]	Manter Usuários.
[RF-02]	Manter Classificações.
[RF-03]	Manter Projetos.
[RF-04]	Manter Requisitos.
[RF-05]	Manter *Product Backlog*.
[RF-06]	Manter *Sprint Backlog*.
[RF-07]	Possibilitar a consulta dos itens anteriores a qualquer momento.
[RF-08]	Disponibilizar uma agenda para cada usuário.
[RF-09]	Disponibilizar a ferramenta *Kanban*
[RF-10]	Disponibilizar relatório e gráfico de “Previsto x Realizado”

Quadro 2 - Requisitos funcionais

O Quadro 3 apresenta a listagem dos requisitos não funcionais identificados para o sistema.

Identificação	Descrição
[NF-01]	O sistema deverá ter uma interface que atende a padrões de usabilidade.
[NF-02]	O sistema deverá ter alta taxa de disponibilidade.
[NF-03]	O sistema deverá ter um tempo de resposta aceitável.
[NF-04]	O sistema deverá manter a integridade dos dados.
[NF-05]	O sistema deverá manter privacidade de acesso a dados.
[NF-06]	O sistema deverá ter segurança.

Quadro 3 - Requisitos não funcionais

O diagrama de casos de uso representado pela Figura 1, mostra os processos e quais usuários participam, conforme determina a metodologia *Scrum*. O *Product Owner* é responsável por manter usuários, projetos, requisitos, cancelamento de *Sprint* e junto com *Development Team* revisar o *Product Backlog*. O *Development Team* também fica responsável pela conclusão dos requisitos e manter o *Sprint Backlog*, por fim, o *Scrum Master* fica responsável por garantir que todas os eventos do *Scrum* sejam realizados.

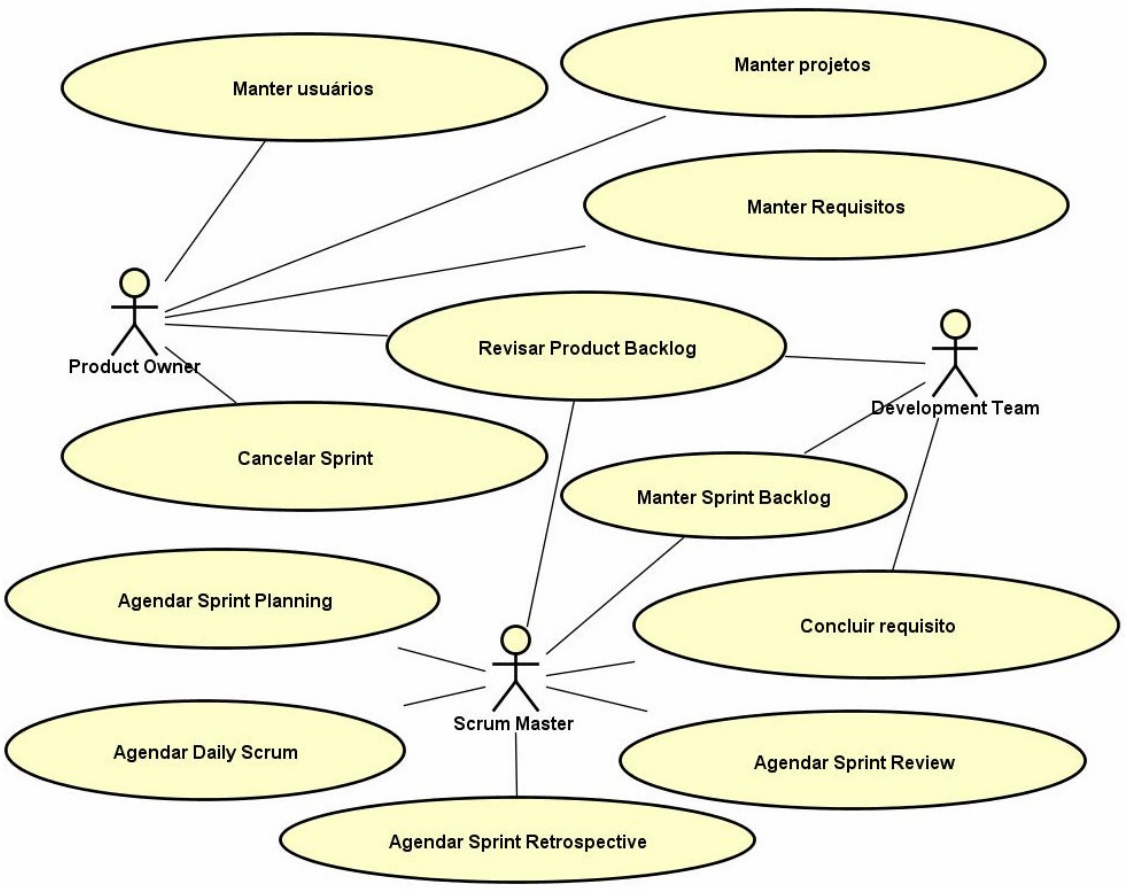


Figura 1 - Caso de Uso

O diagrama de entidade e relacionamento apresentado na Figura 2 apresenta as informações a serem armazenados no banco de dados do sistema.

A tabela de PROJETOS, é uma das principais tabelas do sistema por estar relacionada a quase todas as outras tabelas, sendo possível que um projeto tenha seu *Product Backlog* por meio do campo PROJETO na tabela de REQUISITOS, ter seu *Development Team* pelo relacionamento com a tabela PROJETOS_DEVELOPERS, definir o *Product Owner* e *Scrum Master pelos* campos PRODUCT_OWNER e SCRUM_MASTER respectivamente, ter quantas *Sprints* forem necessárias pela tabela PROJETOS_SPRINTS e seus respectivos *Sprint Backlog* com os dados da tabela PROJETOS_SPRINTS_BACKLOG com a qual está relacionada.

A tabela REQUISITOS é responsável por manter todos os requisitos do sistema, podendo estar vinculado a um projeto. Os requisitos podem ser classificados pela tabela CLASSIFICACOES, que é a responsável por manter as classificações do sistema. Outra tabela vinculada aos requisitos é a tabela de

REQUISITO_LOG_TEMPO, que é a responsável por manter os registros de tempo de desenvolvimento para aquele requisito.

A tabela AGENDA será responsável por manter todos os compromissos dos usuários, podendo estes estarem vinculados a um projeto, *Sprint* ou requisito.

A tabela USUARIOS será responsável por manter todos os usuários do sistema. Vinculada a ela está a tabela de USUARIOS_PERMISSOES, que será responsável por manter todas as permissões deste usuário com relação das ações do usuário no sistema.

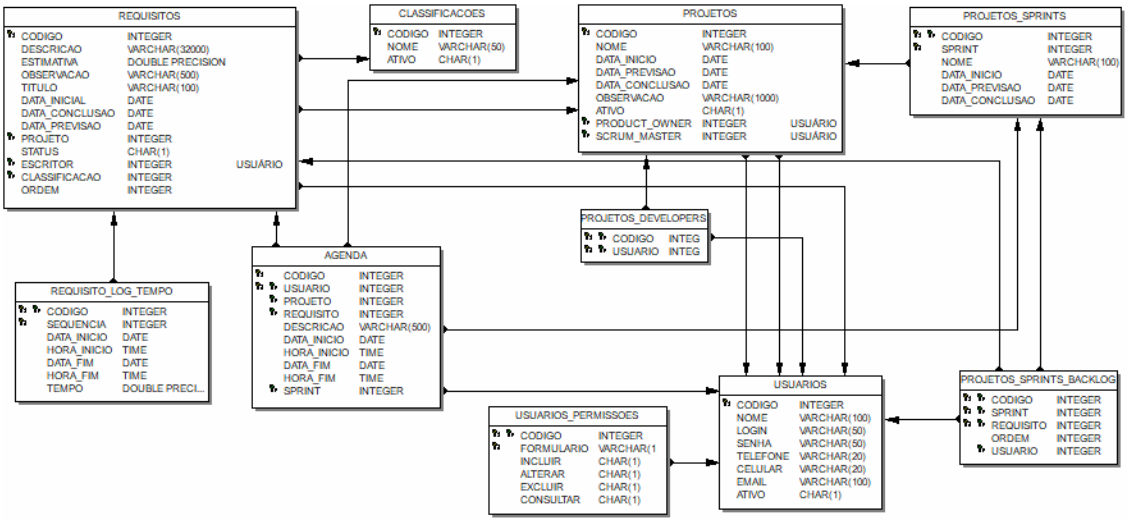


Figura 2 - Diagrama de Entidade-Relacionamento

4.3 APRESENTAÇÃO DO SISTEMA

Neste capítulo será feita a apresentação do sistema, que foi desenvolvido utilizando a linguagem Delphi e a IDE Embarcadero Delphi.

Na Figura 3 é representada a tela de autenticação do sistema, no qual o usuário informará seu usuário e senha cadastrados previamente por um administrador do sistema. Ao realizar uma tentativa de acesso, serão feitas as validações dos dados informados, caso estiverem corretos, será liberado o acesso ao sistema, caso estiverem incorretos, será apresentado uma mensagem de erro ao usuário.

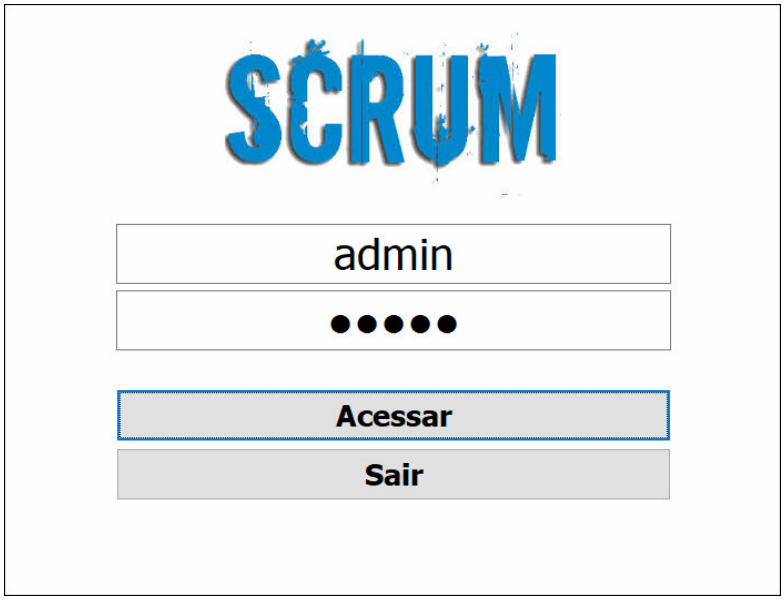


Figura 3 - Login

Na Figura 4 está o menu principal do sistema, no qual o usuário terá acesso a todas as funcionalidades por meio dos menus representados na Figura 4 (a) e ícones para acesso às principais funcionalidades (b).

(a)

(b)

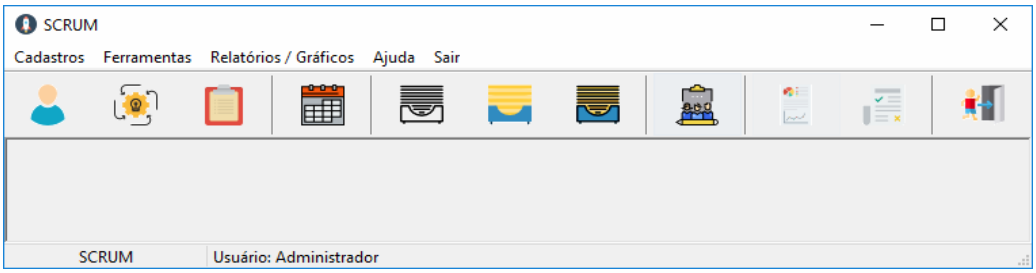


Figura 4 - Menu Principal

Na Figura 5 é exibido o padrão de telas de cadastros do sistema, no qual os botões posicionados ao lado esquerdo possibilitam a inclusão e manutenção do cadastro, enquanto a tabela ao lado apresenta os registros para a consulta dos dados do sistema.

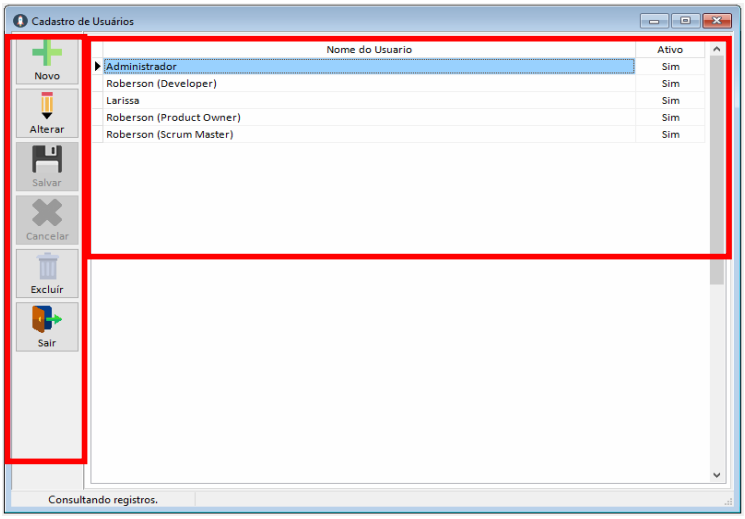


Figura 5 - Tela de cadastros

O padrão utilizado pelo sistema para identificar os campos obrigatórios de um cadastro é realizado por meio do texto “(*)”. Essa validação será executada ao realizar a ação de salvar o registro, como pode ser observado na Figura 6.

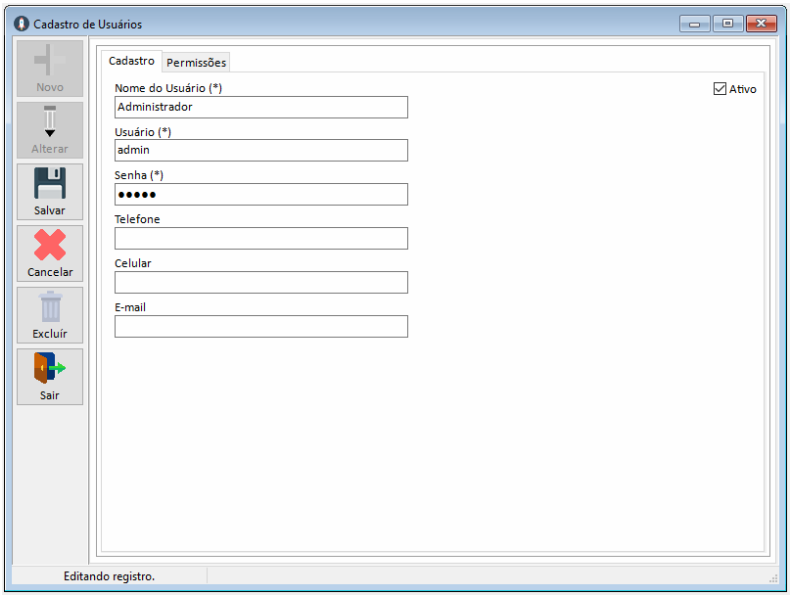


Figura 6 - Campos obrigatórios

Na Figura 7 pode-se observar o padrão das mensagens que serão apresentadas ao usuário em casos de confirmações e erros.

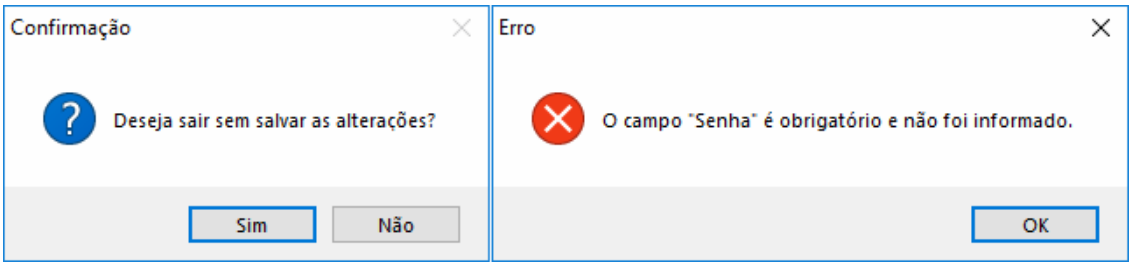
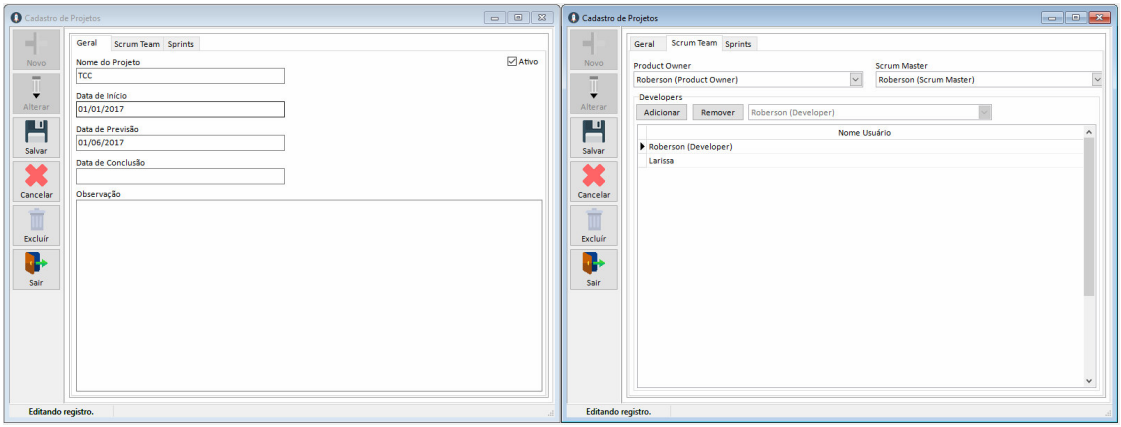


Figura 7 - Mensagem ao usuário

Nas Figuras 8 (a) e (b) é exibido o cadastro de projetos no sistema. A Figura 8 (a) representa as informações gerais do projeto e a Figura 8 (b) representa a formação do *Scrum Team* para determinar os papéis dos usuários do sistema.



(a) **(b)**

Figura 8 - Cadastro de Projeto

Na Figura 9 está representado o cadastro de *Sprint* contido no cadastro de projetos. Nessa tela será possível cadastrar uma ou mais *Sprints* que irão compor o projeto em questão.

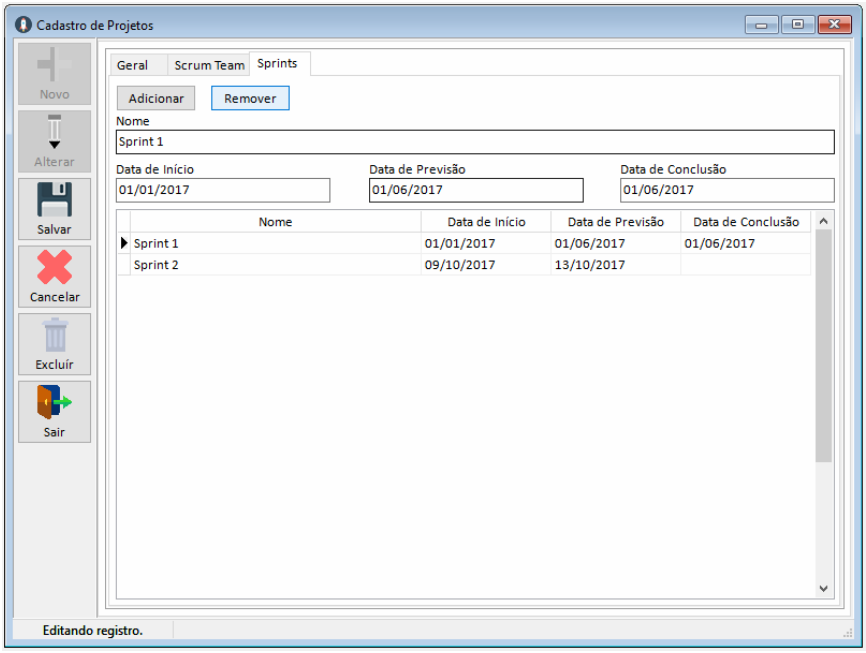


Figura 9 - Cadastro de Sprint

Na Figura 10 está representado o cadastro de requisitos no sistema, no qual poderemos informar seu título, que será utilizado para identificação nas demais telas do sistema, sua estimativa de tempo de desenvolvimento em minutos, datas esperadas para o desenvolvimento, sua classificação, que provem de outro cadastro do sistema, sua descrição, que deverá conter qual é o objetivo deste requisito.

O projeto não é um campo obrigatório, por isso, quando há um requisito cadastrado sem o vínculo a um projeto, ele será considerado um pré-requisito. Caso este requisito seja cadastrado ou vinculado a um projeto posteriormente, então este requisito passa a ser considerado como um item do *Product Backlog.*

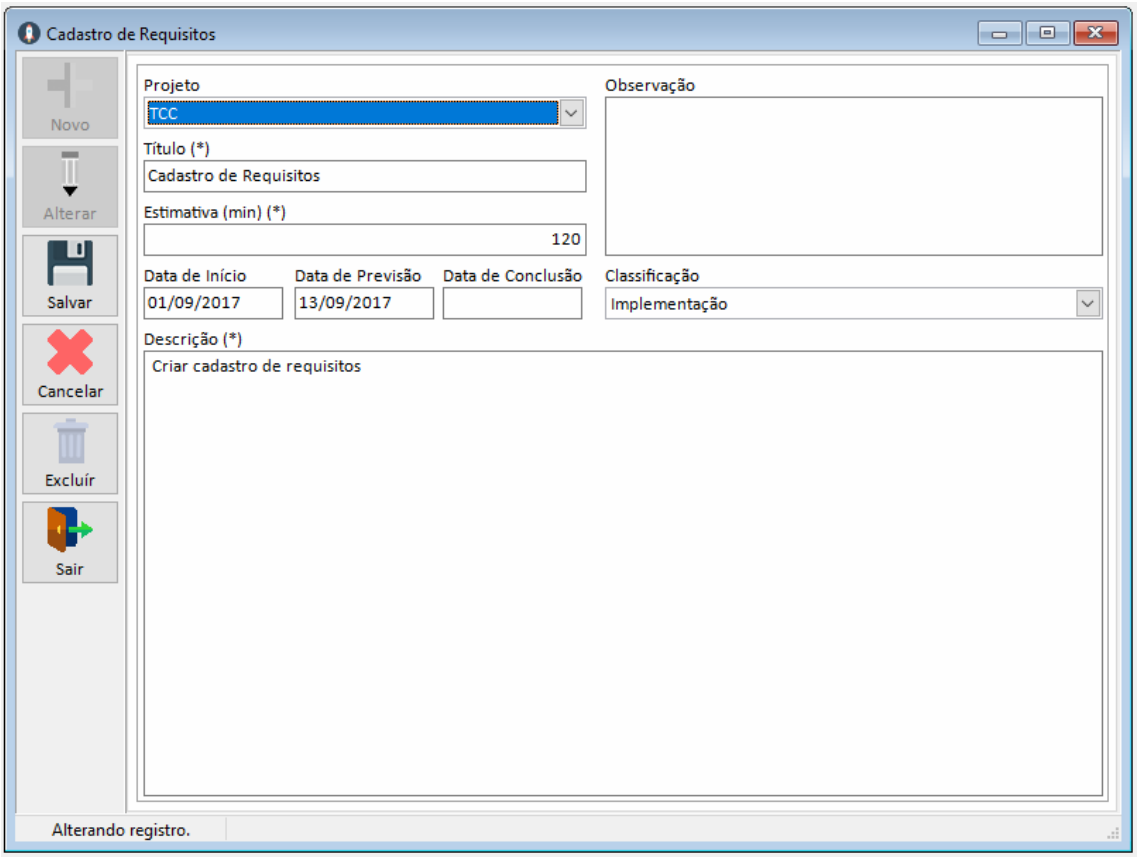
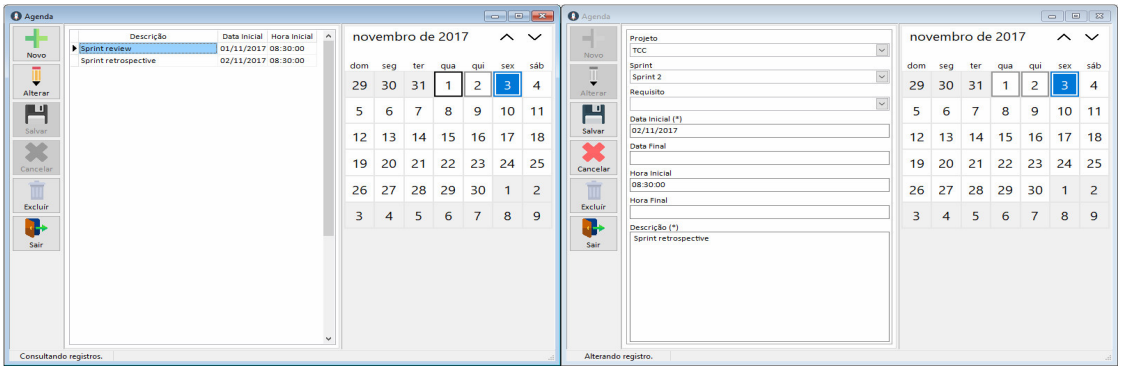


Figura 10 - Cadastro de Requisito

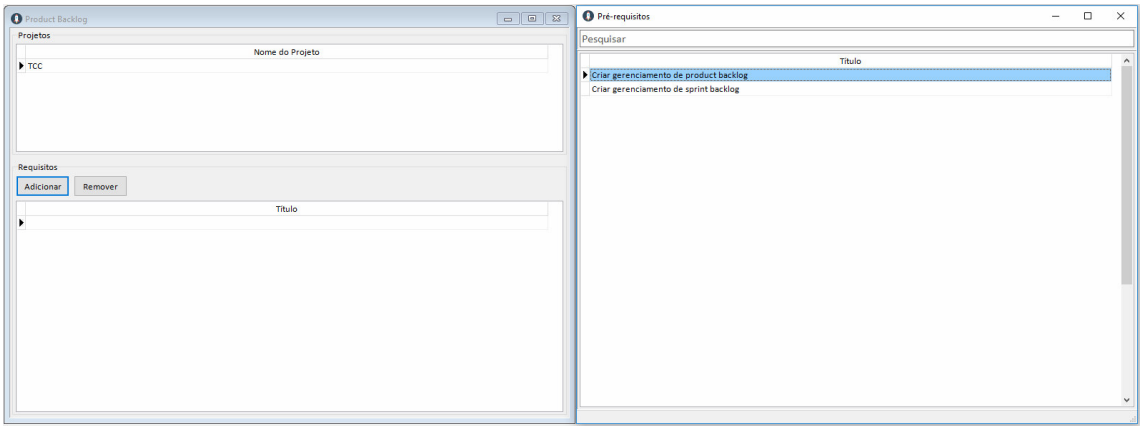
Cada usuário terá sua própria agenda para controle pessoal de compromissos. Na Figura 11 (a) é exibida uma listagem que irá apresentar todos os compromissos do usuário e na Figura 11 (b) pode-se observar o cadastro de um agendamento. No calendário localizado ao lado direto das duas figuras, serão destacados com bordas pretas os dias em que o usuário possui compromissos e em azul estará destacada a data corrente.



(a) **(b)**

Figura 11 - Agenda

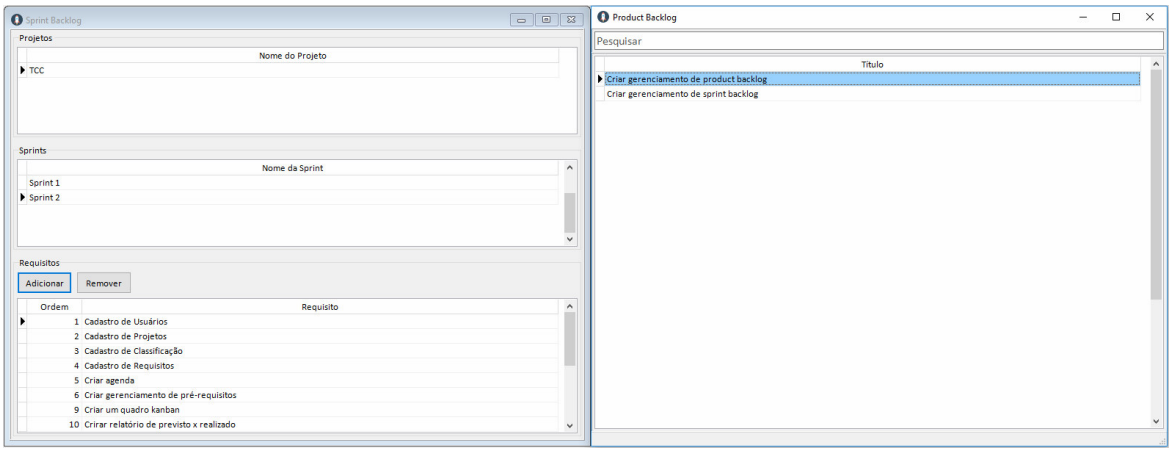
Na Figura 12 está representada a montagem do *Product Backlog*. Na Figura 12 (a) é exibida a lista de projetos cadastrados no sistema na seção Projetos e na seção abaixo estão os requisitos vinculados ao projeto selecionado na primeira tabela. Na Figura 12 (b) está representado a consulta de pré-requisitos que é apresentada ao clicar no botão adicionar presente na Figura 12 (a).



(a) **(b)**

Figura 12 - *Product Backlog*

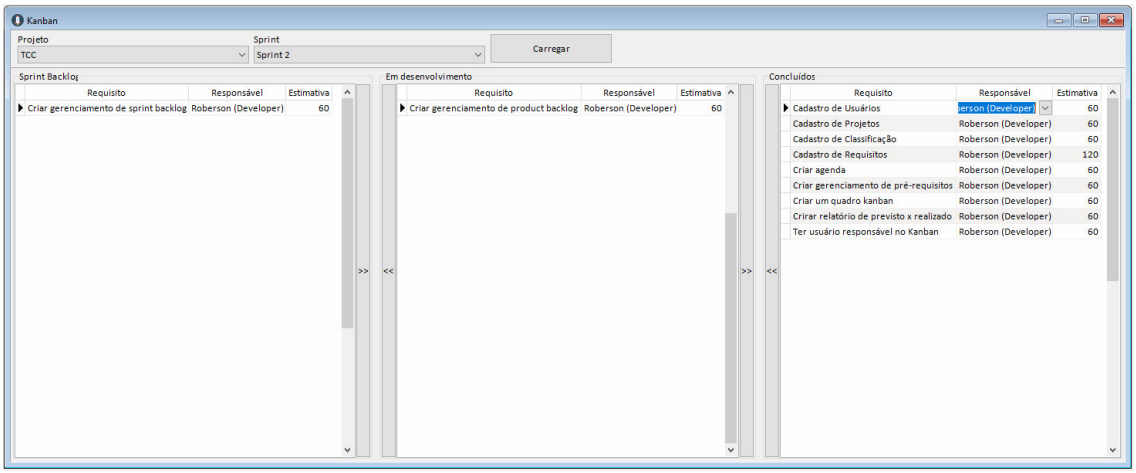
A montagem do *Sprint Backlog* está ilustrada na Figura 13. Na Figura 13 (a) é apresentada a lista de projetos na seção de Projetos, com suas *sprints* logo abaixo na seção *Sprints* e em seguida na seção Requisitos os requisitos presentes na *sprint*. Na Figura 13 (b) é possível visualizar o *Product Backlog* do projeto selecionado na primeira seção e que é apresentado ao clicar no botão adicionar na seção de Requisitos.



(a) **(b)**

Figura 13 - Sprint Backlog

O sistema também possui um quadro *Kanban*, o qual pode ser visto na Figura 14. Neste quadro é possível ter a visão completa de uma *Sprint*. Na Figura 14 (a) são exibidos os requisitos que estão pendentes. Na Figura 14 (b) estão requisitos que estão sendo desenvolvidos no momento. Sempre que um requisito estiver nesta listagem, o sistema irá contabilizar o tempo gasto em desenvolvimento até que ele seja movido para a parte (a) ou (c). Na Figura (c) estão os requisitos já concluídos. Nas três figuras tem-se a coluna responsável, na qual é possível determinar qual é o usuário responsável por aquele requisito.



(a) **(b)** **(c)**

Figura 14 - *Kanban*

Na Figura 15 pode ser observado o gráfico *Burndown*. Neste gráfico é possível acompanhar o andamento de uma *Sprint* para que possa auxiliar na tomada de decisão em relação a *Sprint* atual.

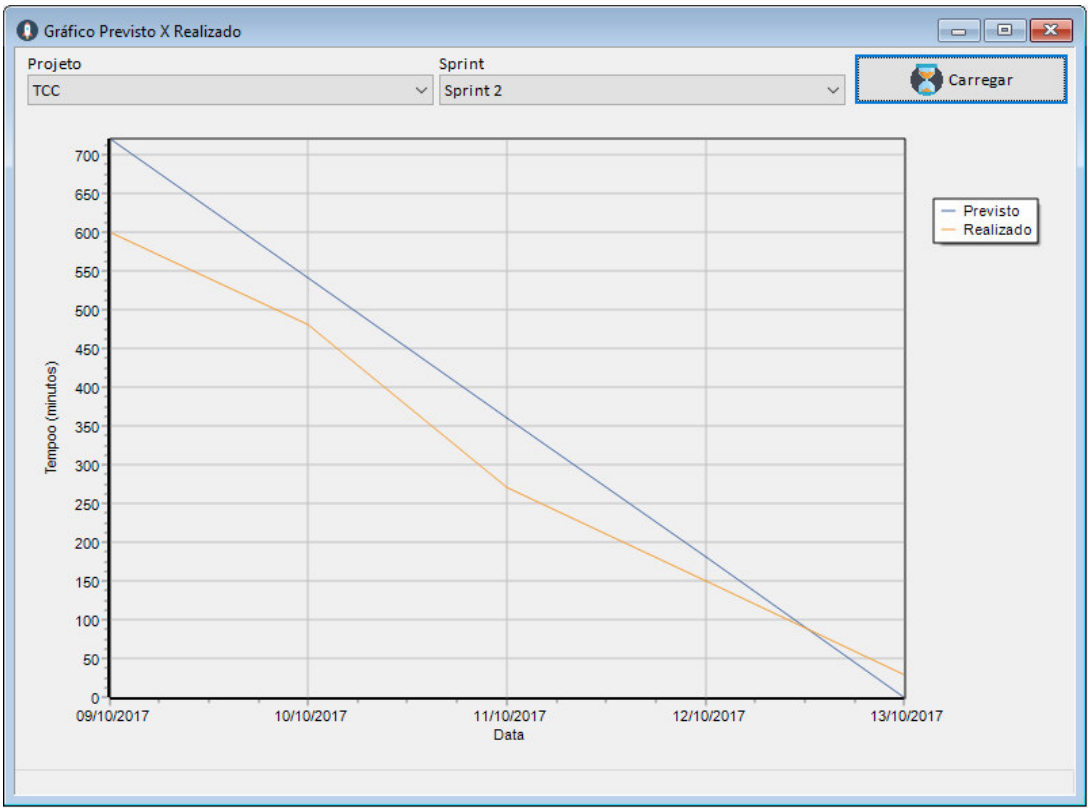


Figura 15 - Gráfico *Burndown*

O sistema permite que o usuário emita um relatório de “Previsto x Realizado”. Neste relatório, apresentado na Figura 16, é possível fazer o mesmo acompanhamento do andamento da *Sprint* que o gráfico *Burndown*.

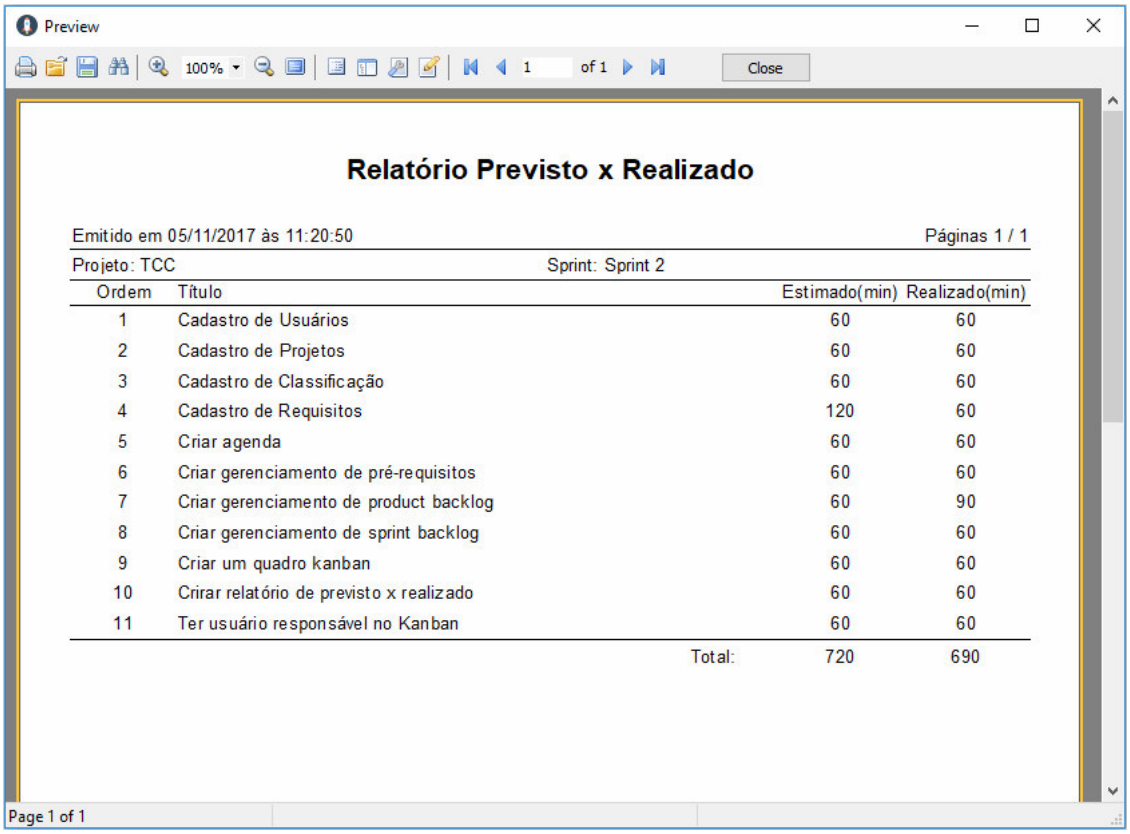


Figura 16 - Relatório “Previsto x Realizado”

Outro relatório disponibilizado pelo sistema é o de projetos, exibido na Figura 17. Ele apresenta ao usuário um resumo geral de todo o projeto, contendo suas informações gerais, de suas *sprints* e requisitos.

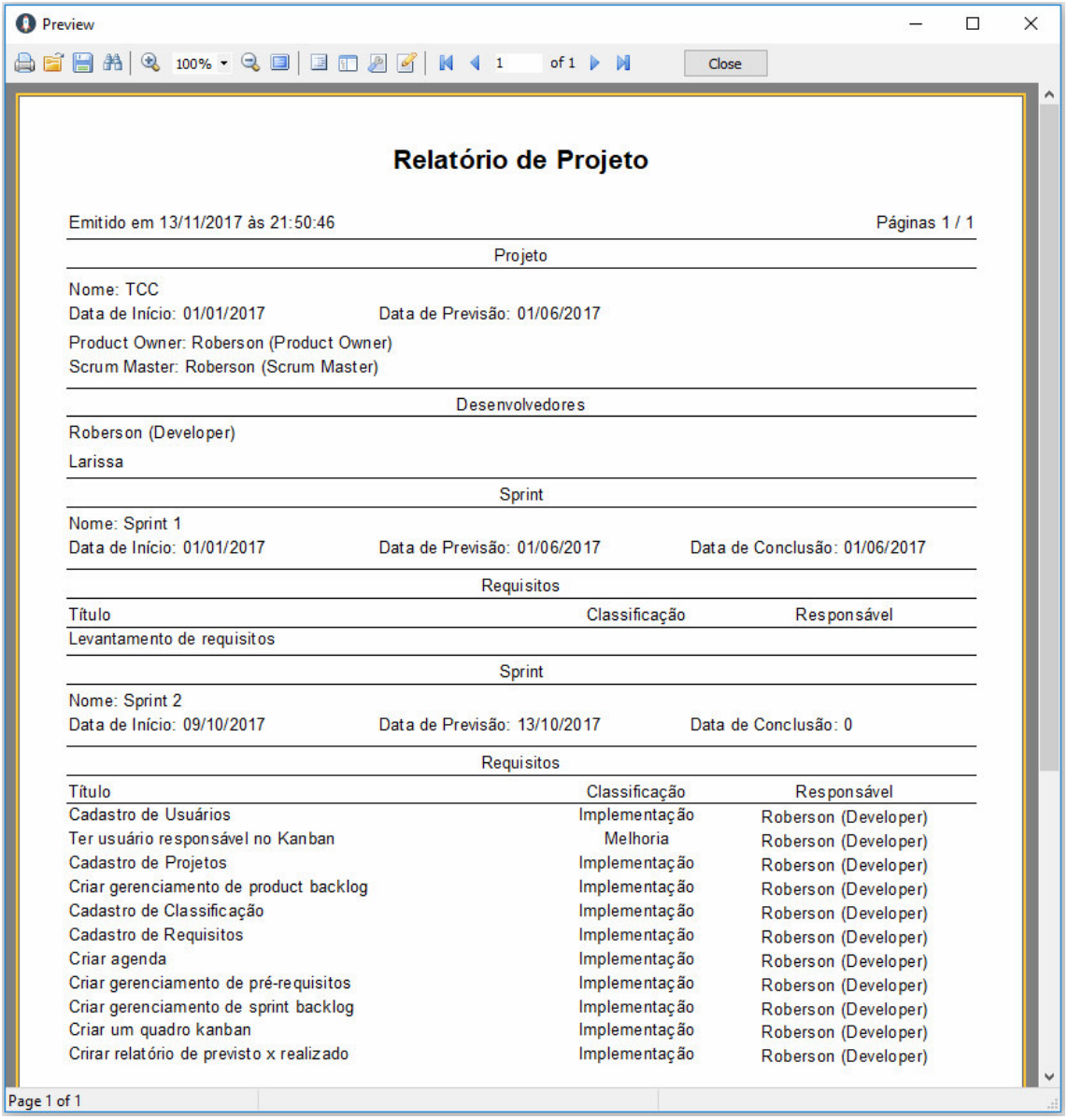


Figura 17 - Relatório de Projeto

4.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção serão apresentadas algumas listagens de códigos que compõem o sistema.

Para o desenvolvimento dos formulários do sistema foi utilizado o padrão de herança da Programação Orientada a Objetos (POO), por isso, na Listagem 1 é apresentada a codificação da criação do formulário base do sistema, do qual as

telas irão herdar suas características. Nessa classe é realizada a validação de permissão de acesso do usuário pode meio de uma pesquisa no banco de dados.

Listagem 1 - Validação de acesso

Para os cadastros do sistema também foi criado um formulário base, que provem do formulário base apresentado na Listagem 1. Seguindo a mesma lógica, na Listagem 2 é realizada a validação de permissões do usuário para incluir, alterar e excluir um registro. Caso o usuário não tenha alguma dessas permissões, serão ocultados os botões que proporcionam essas ações.

Listagem 2 - Validação de inserção, alteração e exclusão.

As ações que podem ser realizadas por um usuário no cadastro do sistema como incluir, alterar, salvar, cancelar, excluir, sair e consultar executam todos a *procedure* representada na Listagem 3. Para cada ação é executado uma *procedure* diferente, as ações são repassadas por meio do parâmetro *Value* e que irá determinar qual *procedure* será executada a seguir, estas *procedures* possuem implementações diferentes para cada ação e serão apresentas a seguir.

Listagem 3 - Ações do usuário

A ação de inserir um novo registro, presente na Linha 3 da Listagem 3, resulta na execução da *procedure* mostrada na Listagem 4, nela pode-se observar que na Linha 3 é realizada a inserção de um registro no *ClientDataSet*[1] sendo o restante apenas manipulação de leiaute para facilitar a iteração do usuário com o sistema.

Listagem 4 - Novo registro

[1] Disponível em: <http://docwiki.embarcadero.com/Libraries/Tokyo/en/Datasnap.DBClient.TClientDataSet>.

Na Listagem 5 é exibida a ação de salvar um registro. Nessa *procedure* pode-se observar a tratativa de erros utilizando a estrutura *try except*[1]. No caso de erro ao salvar o registro no banco de dados, será apresentada uma mensagem ao usuário.

Listagem 5 - Salvar

Ao excluir um registro no sistema, será apresentada ao usuário uma mensagem solicitando a confirmação desta ação para evitar que ocorra uma exclusão por um clique acidental. Na Listagem 6 é exibida a *procedure* que realiza a exclusão. Nessa mesma *procedure* também é realizada a tratativa de erros sendo apresentado uma mensagem ao usuário em caso de erro.

Listagem 6 - Excluir

No fechamento das telas de cadastro do sistema, seja pelo botão “X” na barra de ações do Windows ou pelo botão sair, que executa a ação de sair apresentado na Linha 10 da Listagem 3, a *procedure* mostrada na Listagem 7 será executada. Nela será realizada uma validação para casos em que usuário esteja editando um registro. Nesse caso, o sistema irá solicitar a confirmação da ação do usuário, evitando que não ocorra o fechamento indesejado das telas e sejam perdidas as informações digitadas.

Listagem 7 - Fechamento de telas de cadastro

Na Listagem 8 é exibida a *procedure* que realiza a ação de consultar os registros no banco de dados, mostrada na Linha 11 da Listagem 3. Essa ação não é executada pelo usuário, ela é executada automaticamente ao abrir uma tela de cadastro e ao final das ações de salvar, excluir e cancelar.

Listagem 8 - Consultar

[1] Disponível em: <http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Exceptions_(Delphi)>.

Na utilização do quadro *Kanban* do sistema é possível mover os requisitos entre as colunas para determinar sua situação de desenvolvimento. Para que as alterações desses requisitos possam ser feitas no banco de dados, foi desenvolvido um método, representado na Listagem 9. Esse método espera como parâmetros o código do requisito e qual será sua situação, retornando verdadeiro quando a alteração da situação ocorrer corretamente e falso em casos de falha na alteração, seguido da apresentação de uma mensagem de erro para o usuário.

Listagem 9 - Status do requisito

O controle de tempo de desenvolvimento dos requisitos é realizado por meio do banco de dados com uma tabela para o *log* de tempo e uma *trigger* na tabela dos requisitos. Na Listagem 10 é possível visualizar a implementação desta *trigger,* ela é acionada sempre que for alterada a situação de um requisito, sempre que um requisito recebe a situação “D”, a *trigger* insere um novo registro de *log*. Ao sofrer uma alteração desta situação que seja diferente de “D”, a *trigger* vai atualizar os campos de data e hora final do registro de *log*, inserido anteriormente, possibilitando a contabilização do tempo decorrido para desenvolvimento.

Listagem 10 - Log de Tempo

5 CONCLUSÃO

Com o desenvolvimento deste projeto, foi possível conhecer um pouco melhor a Engenharia de Software e perceber que esta área é de suma importância para se obter o sucesso de um projeto e consequentemente ter um produto de qualidade entregue ao usuário final.

O gerenciamento de projetos utilizando o *Scrum* envolve uma série de processos na gestão de um projeto, como a revisão e a manutenção de requisitos para que sejam feitos durante o decorrer de uma *sprint*, tendo o registro de tempo de desenvolvimento para cada requisito, possibilitando o acompanhamento por meio de gráficos e relatórios para que seja concluída dentro dos prazos estipulados.

A IDE Embarcadero Delphi juntamente com linguagem Delphi se provaram eficientes e supriram todas as necessidades de desenvolvimento de uma aplicação *desktop* com o escopo e requisitos definidos para o produto desenvolvido como resultado da realização deste trabalho.

O sistema desenvolvido possibilita a gestão do desenvolvimento de projetos de software com a utilização do *Scrum*, procurando facilitar o trabalho dos gerentes de projetos e desenvolvedores. Assim, os objetivos propostos foram atingidos.

Como trabalhos futuros podem ser implementados novas funcionalidades no sistema para que possibilite gerir projetos utilizando-se de outras metodologias.

REFERÊNCIAS

GAITHER, Norman; FRAZIER, Greg. **Administração da produção e operações.** 8. ed. São Paulo: Editora Pioneira Thomson Learning, 2001.

GHISI, Thiago. **Kanban no desenvolvimento de projetos de software.** Engenharia de Software Magazine, 45 ed**.** Revista Engenharia de Software Magazine. Devmedia, 2016.

IKONEN, Marko et al. **Exploring the sources of waste in kanban software development projects**. In: EUROMICRO Conference on Software Engineering and Advanced Applications, 36th. Lille: 2010.

KNIBER, Henrik; SKARIN, Mattias. **Kanban vs. Scrum: making the most of both**. 2009. InfoQ.com - USA.

MAHNIC, V.; ZABKAR, N. **Measuring progress of Scrum-based software projects**. Electronics and Electrical Engineering, v 18, n 8, 2010.

MARTINS, J. C. C. **Técnicas para gerenciamento de projeto de software**. Rio de Janeiro: Brasport, 2007.

OHNO, Taiichi. **O Sistema Toyota de produção**. Porto Alegre: Bookman, 1997.

PHAM, Andrew; PHAM, Phuong-Van. **Scrum em ação: gerenciamento de desenvolvimento ágil de projetos de software.** São Paulo, SP: Novatec, 2011.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional.** 7 ed., Porto Alegre: AMGH, 2011.

PRIES, Kim H. QUIGLEY, Jon M. **Scrum project management**. New York, CRC Press. 2010.

SILVA, Reni Elisa da; SOUZA NETO, João. **Contratação do desenvolvimento ágil de *software* na administração pública federal: riscos e ações mitigadoras.** Brasília, DF: Revista do Serviço Público de Brasília. 2015.

SCHWABER, Ken; SUTHERLAND, Jeff. **Um guia definitivo para o Scrum: as regras do jogo.** Disponível em: <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: 04 set. 2016.

SCHWABER, Ken. **Agile project management with Scrum.** Redmond, Washington: Microsoft Press, 2004.

SOMMERVILLE, Ian. **Engenharia de software**. 9 ed., São Paulo: Pearson Prentice Hall, 2011.