UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

APLICAÇÃO WEB PARA ACOMPANHAMENTO DE PROJETOS ÁGEIS UTILIZANDO O MÉTODO KANBAN

LUIZ FERNANDO DEBATIN

BLUMENAU
2011

2011/1-15

LUIZ FERNANDO DEBATIN

APLICAÇÃO WEB PARA ACOMPANHAMENTO DE PROJETOS ÁGEIS UTILIZANDO O MÉTODO KANBAN

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Everaldo Artur Grahl, Mestre - Orientador

BLUMENAU
2011

2011/1-15

APLICAÇÃO WEB PARA ACOMPANHAMENTO DE PROJETOS ÁGEIS UTILIZANDO O MÉTODO KANBAN

Por

LUIZ FERNANDO DEBATIN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
	_____
Membro:	Prof. Roosevelt dos Santos Jr, Especialista – FURB
	_____
Membro:	Prof. Wilson Pedro Carli, Mestre – FURB

Blumenau, 29 de junho de 2011.

Dedico este trabalho à minha família e a todos que direta ou indiretamente contribuíram para a realização deste trabalho.

AGRADECIMENTOS

Primeiramente a Deus, pois sem Ele, nada seria possível e não estaríamos aqui reunidos, desfrutando, juntos, destes momentos que nos são tão importantes.

Aos meus pais Luiz Carlos e Zanira; pelo esforço, dedicação e compreensão, em todos os momentos desta e de outras caminhadas.

Á minha namorada Jaqueline pela paciência, compreensão, incentivo e ajuda na realização deste trabalho.

Aos meus amigos pela força e apoio.

Ao meu orientador Everaldo Artur Grahl por ter acreditado na conclusão deste trabalho.

Para realizar grandes conquistas, devemos não apenas agir, mas também sonhar; não apenas planejar, mas também acreditar.

Anatole France

RESUMO

O desenvolvimento de software vem evoluindo a cada dia com a criação de novas tecnologias e metodologias, visando alcançar produtos com melhor qualidade e em menor tempo. Controlar e acompanhar o processo de desenvolvimento é peça chave para alcançar bons resultados. Para isso o *Kanban*, desenvolvido no Sistema Toyota de Produção, mostrou-se ser uma valiosa metodologia tornando o processo visível a todos os envolvidos, identificando problemas antes da finalização e produzindo apenas o necessário. Este trabalho apresenta o desenvolvimento de uma aplicação *web* para o acompanhamento de projetos, baseada no método *Kanban*. A aplicação foi desenvolvida em PHP utilizando o *framework* cakePHP e banco de dados MySQL. Como resultado destaca-se a facilidade da equipe de desenvolvimento em visualizar o trabalho que está sendo feito em cada processo.

Palavras-chave: *Kanban*. Acompanhamento de Projeto. Sistema Toyota de Produção. CakePHP.

ABSTRACT

The software development is evolving every day with the creation of new technologies and methodologies in order to achieve products with best quality and in less time. Control and monitor the development process is key to achieve good results. For this, *Kanban*, developed in the Toyota Production System, proved to be a valuable methodology making the process visible to all involved, identifying problems before finalizing and producing only what is necessary. This paper presents the development of a web application for tracking projects, based on the *Kanban* method. The application was developed in PHP using the CakePHP framework and MySQL database As a result there is the ease of the development team to visualize the work being done in each process.

Key-words: Kanban. Project Tracking. Toyota Production System. CakePHP.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

DOM - *Document Object Model*

FDD - *Feature Driven Development*

HTML – Hyper Text Markup Language

IDE - *Integrated Development Environment*

PHP - *Hypertext Preprocessor*

STP – Sistema Toyota de Produção

UML - *Unified Modeling Language*

XP - *eXtreme Programming*

WIP – *Work in Process*

SUMÁRIO

1 INTRODUÇÃO

O mundo produtivo industrial observou, no Japão, a partir da década de 1950, uma reestruturação na gestão da produção e na forma de organização do trabalho decorrentes do desenvolvimento do Sistema Toyota de Produção (STP) na Toyota Motor Company (MAGEE, 2008). Com a constatação do promissor desempenho da Toyota, muitos estudos voltaram-se, a partir de então, a desvendar quais os fatores responsáveis por tais resultados. Dentre os métodos utilizados pela Toyota, o sistema *Kanban*, usado para controlar os estoques em processo, a produção e o suprimento de componentes, tornou-se um dos principais focos de pesquisa e, com isso, foi amplamente estudado e difundido.

Kanban é uma das técnicas usadas para implementar o conceito de *Lean Manufacturing* (Produção enxuta), onde a saída de produtos acabados, ao final da linha de montagem, dita o ritmo da introdução de matéria-prima no sistema. Desta forma evita-se o acúmulo de produtos inacabados ao longo da linha, diminuindo a quantidade de Trabalho em Processo *Work In Process* (WIP). Com menos produtos intermediários tem-se uma sobrecarga menor no sistema e pode-se adaptar melhor e mais rápido às mudanças na demanda dos clientes.

Nos projetos de desenvolvimento de software, o *Kanban* complementa muito bem as abordagens ágeis, como *Feature Driven Development* (FDD), Scrum e *eXtreme Programming* (XP), entre outras.

A gestão de projetos é um conjunto de ferramentas gerenciais que permitem que a organização desenvolva um conjunto de habilidades, incluindo conhecimento e capacidades individuais, destinados ao controle de eventos não repetitivos, únicos e complexos, dentro de um cenário de tempo, custo e qualidade predeterminados (VARGAS, 2005). Paula Filho (2003) afirma que tradicionalmente as iniciativas de desenvolvimento de software se preocupam quase que exclusivamente com os problemas técnicos, dedicando poucos recursos e tempo ao lado gerencial e de suporte. Isto ocorre muitas vezes, pois a organização possui conhecimento técnico para o desenvolvimento de software, mas faltam os conhecimentos administrativos, o que é essencial para que a organização cresça e seja competitiva.

Problemas de gerenciamento no processo de desenvolvimento de software, que também ocorrem nas outras áreas da engenharia, são ainda mais difíceis de gerir devido aos aspectos peculiares do software: é um produto intangível e não existe um processo "industrial" normalizado (TOMAYKO; HALLMAN, 1999). De acordo com McAdam e

Fulton (2002) o desenvolvimento de software comparado com outros tipos de produção requer maior controle, mas esse controle é mais difícil de definir e mais difícil de aplicar.

Este trabalho apresenta uma aplicação *web* que auxilie o acompanhamento de projetos de software, introduzindo no ambiente de desenvolvimento o quadro de *Kanban* a fim de possibilitar a visualização das tarefas que estão sendo implementadas em tempo real e em que estágio a mesma se encontra.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver uma aplicação *web* para acompanhamento de projetos ágeis de desenvolvimento de software, baseado no método *Kanban*, com o intuito de disponibilizar um ambiente onde todos os envolvidos possam visualizar o trabalho em progresso e evidenciar os problemas antes de finalizar o processo.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está disposto em quatro capítulos.

No primeiro capítulo apresenta-se a introdução, os objetivos e a estrutura do trabalho.

No segundo capítulo tem-se a fundamentação teórica, destacando-se os conceitos de cada elemento envolvido neste processo, com ênfase no método *Kanban*, bem como os trabalhos correlatos.

No terceiro capítulo é apresentado o desenvolvimento do aplicativo, incluindo detalhes sobre a especificação, implementação e tecnologia utilizada.

No quarto capítulo apresenta-se a conclusão sobre o trabalho, enfatizando os objetivos alcançados, bem como sugestões para trabalhos futuros como extensão deste.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os principais assuntos que auxiliam o melhor entendimento deste trabalho, assim como os trabalhos correlatos.

2.1 MEDOTOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

O processo de desenvolvimento de software é composto basicamente de fases que compreendem a modelagem do negócio, as especificações dos dados e processos envolvidos e a programação propriamente dita. Uma metodologia de desenvolvimento de sistemas é um guia que descreve o processo ser seguido pelos desenvolvedores. Para expressar os resultados das diversas fases da metodologia, utilizam-se representações diagramáticas, como os diagramas de caso de uso da *Unified Modeling Language* (UML) (FURTADO, 2002).

Para Rezende (2005), uma metodologia completa constitui-se de uma abordagem organizada para atingir um objetivo, por meio de passos preestabelecidos. É um roteiro, um processo dinâmico e iterativo para desenvolvimento estruturado de projetos, sistemas ou software, visando à qualidade, produtividade e efetividade de projeto.

Uma boa metodologia de desenvolvimento de sistemas facilita o entendimento do negócio e dos processos envolvidos em um sistema informatizado, facilitando sua construção e, posteriormente, sua manutenção. Além disso, é possível propiciar um bom nível de entendimento do sistema.

Rezende (2005), ainda afirma que uma metodologia não é uma técnica tão somente, pois se pode utilizar qualquer técnica para desenvolvimento de projetos, sistema ou software, de acordo com a preferência e competência da equipe multidisciplinar envolvida. Como exemplo de técnicas, pode ser utilizado a Análise Estrutural, a Análise Essencial, a Análise Orientada a Objetos e a UML. Desse modo, a metodologia é um roteiro que permite o uso de uma ou várias técnicas por opção dos desenvolvedores do sistema de informação ou software.

2.1.1 Metodologias ágeis

Também chamados de processo ágil ou processo leve, os métodos ágeis possuem seu foco na eficiência, abordando como premissa o compromisso entre "nada de processo" e processos rigorosos (ORTH; PRIKLADNICKI, 2009).

Métodos ágeis enfatizam comunicações em tempo real, preferencialmente face a face, a documentos escritos. A maioria dos componentes de um grupo ágil devem estar agrupados em uma sala. Isto inclui todas as pessoas necessárias para terminar o software. No mínimo, isto inclui os programadores e seus clientes (clientes são as pessoas que definem o produto, eles podem ser os gerentes, analistas de negócio, ou realmente os clientes). Nesta sala devem também se encontrar os testadores, projetistas de iteração, redatores técnicos e gerentes.

Métodos ágeis também enfatizam trabalho no software como uma medida primária de progresso. Combinado com a comunicação face a face, métodos ágeis produzem pouca documentação em relação a outros métodos, sendo este um dos pontos que podem ser considerados negativos. É recomendada a produção de documentação que realmente será útil (FOWLER, 2006).

De acordo com Spyer (2009), uma metodologia ágil busca prover um modo de trabalho para desenvolvimento de software que ataca os riscos mais comuns desta atividade. Este modo é baseado em princípios que, ao contrário de outras metodologias mais pesadas e formais, valorizam garantir a satisfação do usuário através de ciclos curtos de desenvolvimento, chamados de iterações e da interação freqüente dos desenvolvedores e outros atores envolvidos no processo.

Com isso garante que cada iteração tenha-se uma aplicação usável, os métodos ágeis atacam os riscos comuns dos métodos que envolvem um grande esforço de planejamento e especificação antes de se iniciar o desenvolvimento propriamente dito.

Uma característica comum dos métodos ágeis é a capacidade de funcionar em ambientes muito exigentes que tem um grande número de incertezas e mudanças que podem vir de várias fontes como a equipe em processo de formação que ainda não trabalhou junto em outros projetos, requisitos voláteis, baixo conhecimento do domínio de negócio pela equipe, a doção de novas tecnologias, as novas ferramentas, as mudanças muito bruscas e rápidas no ambiente de negócios das empresas as novos concorrentes os novos produtos os novos modelos de negócio (ORTH; PRIKLADNOCKI, 2009).

2.1.2 Gerenciamento de projetos

De acordo com Project Management Institute (2001), um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. Os projetos e as operações diferem, principalmente, no fato de que os projetos são temporários e exclusivos, enquanto as operações são contínuas e repetitivas.

Os projetos são normalmente autorizados como resultado de uma ou mais considerações estratégicas. Estas podem ser uma demanda de mercado, necessidade organizacional, solicitação de um cliente, avanço tecnológico ou requisito legal.

As principais características dos projetos são:

a) temporários, possuem um início e um fim definidos;
b) planejados, executado e controlado;
c) entregam produtos, serviços ou resultados exclusivos;
d) desenvolvidos em etapas e continuam por incremento com uma elaboração progressiva;
e) realizados por pessoas;
f) com recursos limitados.

Segundo Martins (2007), gerência de projetos é a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas para atingir um conjunto de objetivos pré-definidos, num certo prazo, com certo custo e qualidade, através da mobilização de recursos técnicos e humanos.

Na indústria de informática, geralmente há dois tipos de abordagens comumente utilizadas no gerenciamento de projetos. As abordagens do tipo "tradicional" identificam uma seqüência de passos a serem completados. Essas abordagens contrastam com a abordagem conhecida como desenvolvimento ágil de software, em que o projeto é visto como um conjunto de pequenas tarefas, ao invés de um processo completo.

O acompanhamento, ou retroalimentação, deve ser imediato e específico, podendo construir-se em elogio ou repreensão – treinamento. Se for vago e atrasado, não funcionará como um instrumento eficiente de treinamento.

2.2 KANBAN

A tradução literal da palavra japonesa *Kanban* é Registro visível ou Placa visível. Porém, numa interpretação mais fiel ao aspecto de tal registro do que à própria tradução literal, pode-se afirmar que *Kanban* significa cartão.

O *Kanban* foi criado na década de 1950 pelo ex-vice-presidente da Toyota Motor Company do Japão, Tiichi Ohni. De acordo com Moura (1989), as idéias de Ohno sobre o *Kanban* foram inspiradas no supermercado americano, onde as prateleiras eram reabastecidas quando esvaziadas. Como o espaço de cada item era limitado, somente se traziam mais itens quando havia necessidade. A teoria de Ohno diz que tudo que existir além da quantidade mínima de materiais, peças, equipamentos e operários (horas de trabalho), necessária para fazer um dado produto é "perda" e, portanto, só aumenta os custos em todo o sistema.

Para Pace (2003), *Kanban* é uma técnica empregada ao nível de chão de fábrica para auxílio do controle da produção. Faz parte de uma técnica mais abrangente, chamada *Just-in-Time*. Consiste no emprego de cartões, tanto para requisitar material de um centro produtor para um centro consumidor, quanto para ordenar o centro produtor a produzir determinado produto em determinado momento. Visa à redução dos estoques em processo a limites mínimos e à produção somente quando necessário. Ou seja, o centro produtor somente produz quando o centro seguinte (consumidor) ordena. Essa ordem é feita por meio de cartões – daí a origem do nome do sistema. Dessa forma evita-se produzir o que é desnecessário no momento.

> Uma ferramenta de comunicação, no sistema '*Just-in-time'* de controle de produção e inventário, desenvolvido por Taiichi Ohno na Toyota. Um *Kanban*, ou cartão, é anexado a peças específicas na linha de produção, significando a entrega de uma determinada quantidade. Quando todas as peças tiverem sido consumidas, o mesmo cartão é enviado de volta a sua origem, onde torna-se um pedido de mais peças (IMAI, 1994).

Para Monden (1984), o *Kanban* é um sistema de informação para controlar harmoniosamente as quantidades de produção em todo o processo. Usualmente, é um cartão colocado em um envelope retangular de vinil. São usados dois tipos principais de cartões: *Kanban* de Movimentação e *Kanban* de Produção.

Porém, para implantar um sistema de *Kanban* é necessária toda uma mudança de estrutura, implementação de conceitos e ferramentas de forma padronizada e planejada. Desta forma, devem-se padronizar as atividades e processos com a criação de procedimentos padrão a fim de que não se percam as melhorias alcançadas.

No caso de uma ferramenta, o sistema de *Kanban* se adapta e age como um sequenciador automático da produção, informando cada operador sobre sua próxima tarefa e qual a urgência da mesma.

Conforme Moura (1989) a sistemática de utilização do *Kanban* é a seguinte:

a) o cliente "puxa", através de seu pedido;
b) dispara-se todo o processo, envolvendo a montagem, a produção, o transporte, a reposição e os fornecedores;
c) mantém-se, em cada etapa, uma quantidade mínima de componentes, suficiente para atender aos diversos clientes dos processos.

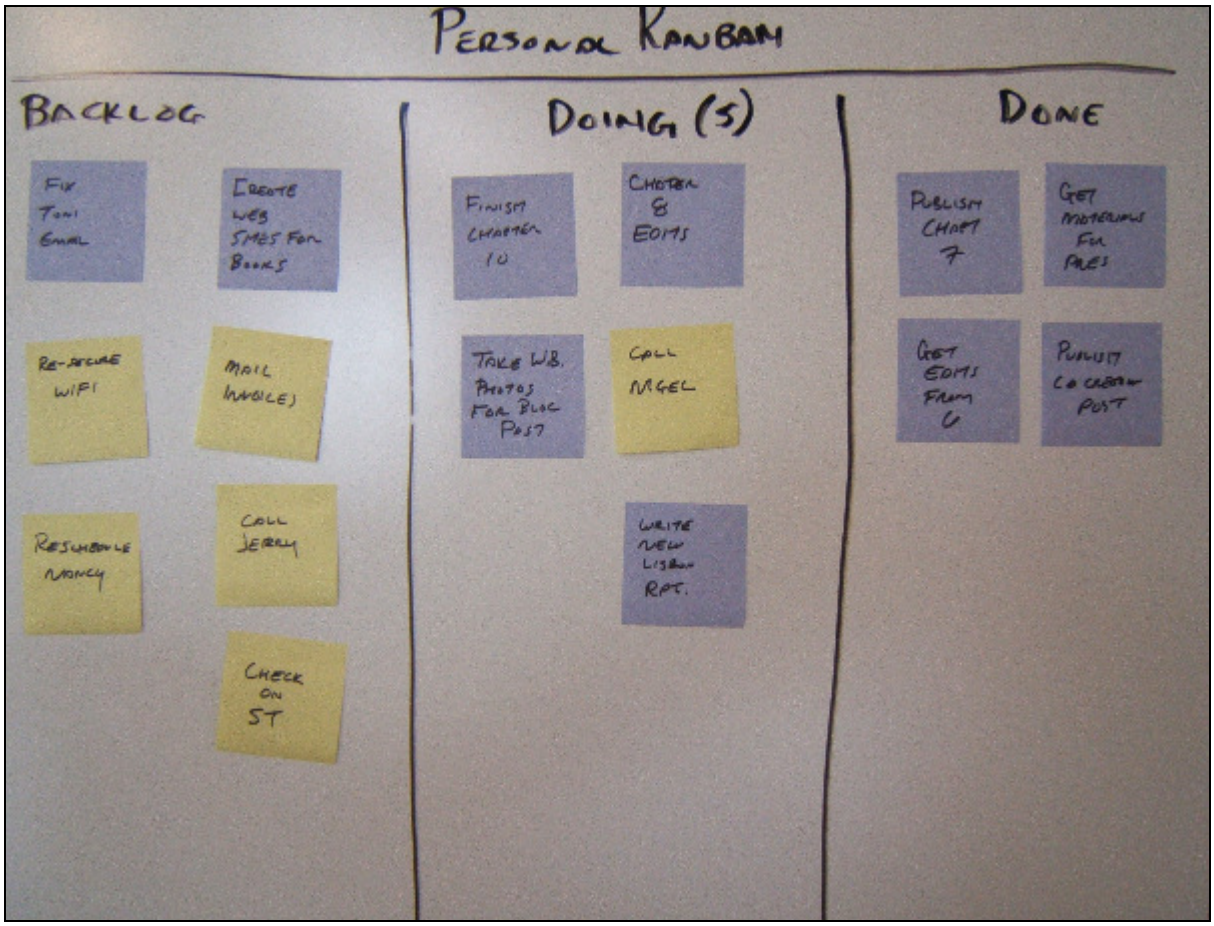
Cabe ressaltar que, com a implantação deste sistema, nenhuma atividade poderá ser executada sem que haja um cartão autorizando. Sendo assim, o *Kanban* é utilizado com o intuito de possuir um controle visual dos processos, evitar atrasos de entrega, bem como, desperdícios com a produção de peças duplicadas, ou criadas antecipadamente.

2.2.1 Quadro de Kanban

Também chamado de Painel Porta-*Kanban*, trata-se de uma placa, geralmente com formato retangular, pintadas com três faixas horizontais, sendo a primeira, de cima para baixo, de cor vermelha, a intermediária de cor amarela e a inferior, de cor verde. Pode ser feito de madeira ou qualquer outro material apropriado às condições de momento. Sua principal função é receber os *Kanbans* (PACE, 2003).

De acordo com Linker e Meier (2006), não existe um modelo padrão para o quadro, visto que algumas vezes ele é até dispensado, preferindo-se pintar as raias no próprio piso da fábrica, eliminando-se dessa forma até mesmo o uso de cartões de produção.

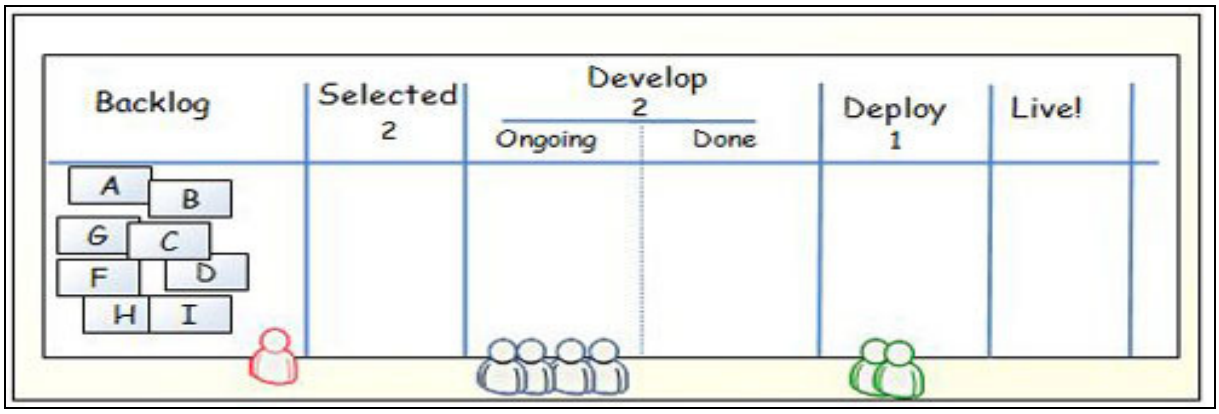
Já Moura (1989), ainda ressalta que o quadro pode se adequar aos processos e estágios da fábrica, sendo assim bastante maleável. Na Figura 1, pode-se visualizar um exemplo de quadro de *Kanban*. Neste exemplo o quadro foi divido em três colunas o *Backlog*, o *Doing* e o *Done*, cada uma delas representando uma etapa no processo. Na coluna *Doing* tem-se o número 5, entre parênteses, este representa o WIP máximo da coluna, ou seja, o número máximo de cartões que a coluna suporta ao mesmo tempo.



Fonte: INFOQ (2010).

Figura 1 – Quadro de *Kanban*

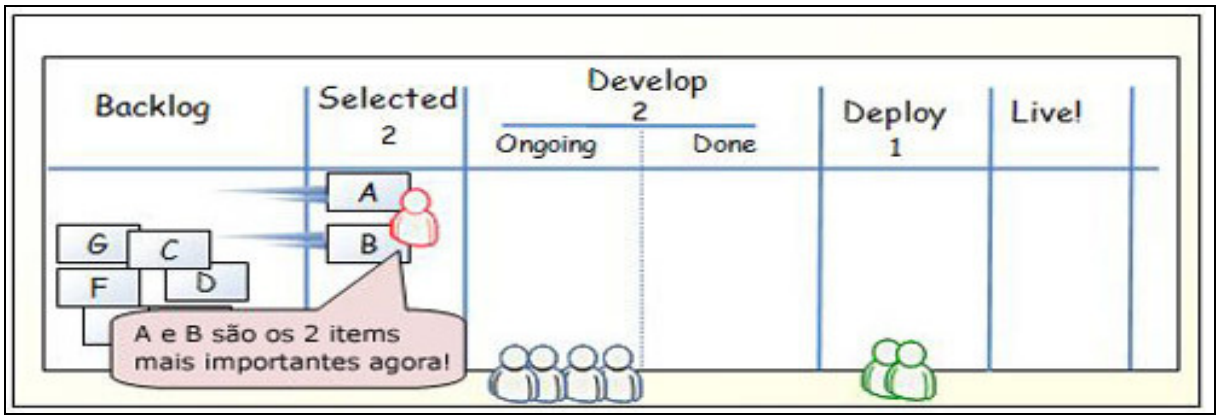
Para entender melhor o funcionamento do quadro de *Kanban* da Figura 2 até a 13, tem-se um exemplo de como é feita a movimentação dos cartões no quadro respeitando o WIP de cada coluna. Na Figura 2 pode-se verificar que o quadro está divido em cinco colunas, *Backlog, Selected, Develop, Deploy e Live* respectivamente. Pode-se observar também que as colunas *Selected, Develop e Deploy* possuem limitadores de WIP, sendo este o número máximo de cartões que podem ser desenvolvidos simultaneamente nesta coluna. Neste exemplo o processo é desempenhado por três papéis o responsável por selecionar os cartões que devem ser desenvolvidos (representado pelo boneco vermelho), os responsáveis por desenvolver as tarefas dos cartões (representados pelos bonecos azuis) e os responsáveis por executar o *deploy* e liberar os cartões (representados pelos bonecos verdes).



Fonte: CASTRO (2009).

Figura 2 – Exemplo de utilização do quadro de *Kanban* parte 1

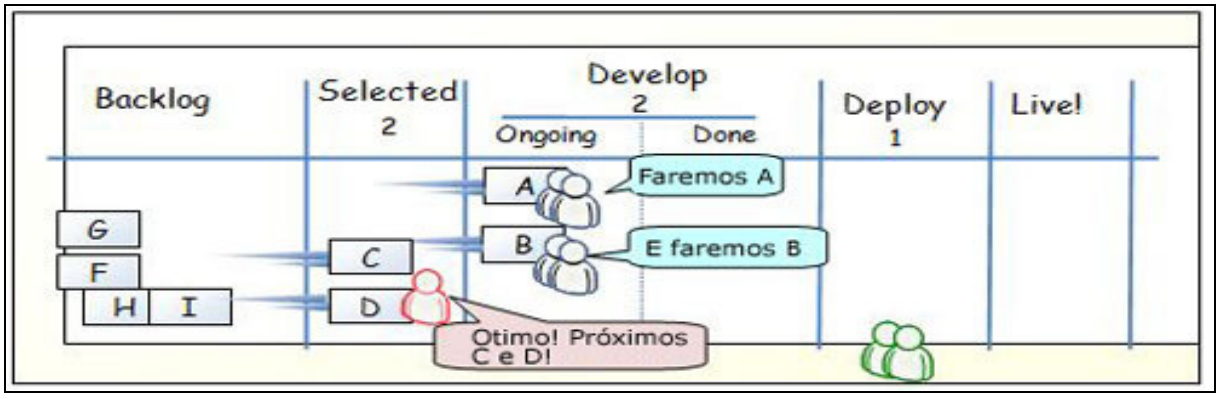
Na Figura 3 os cartões A e B são selecionados para iniciarem os trabalhos no quadro.



Fonte: CASTRO (2009).

Figura 3 – Exemplo de utilização do quadro de *Kanban* parte 2

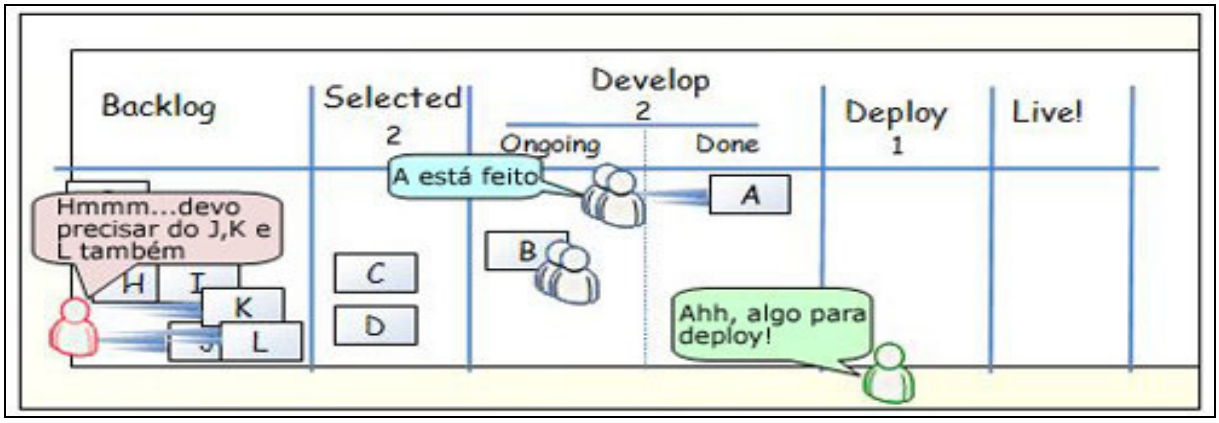
Na Figura 4 pode-se verificar que as equipes de desenvolvimento iniciam os trabalhos dos cartões A e B, enquanto os cartões C e D são selecionados para próximas atividades. Neste ponto pode-se observar que são selecionados apenas dois cartões por vez, pois a coluna *Selected* possui um WIP de 2 cartões.



Fonte: CASTRO (2009).

Figura 4 – Exemplo de utilização do quadro de *Kanban* parte 3

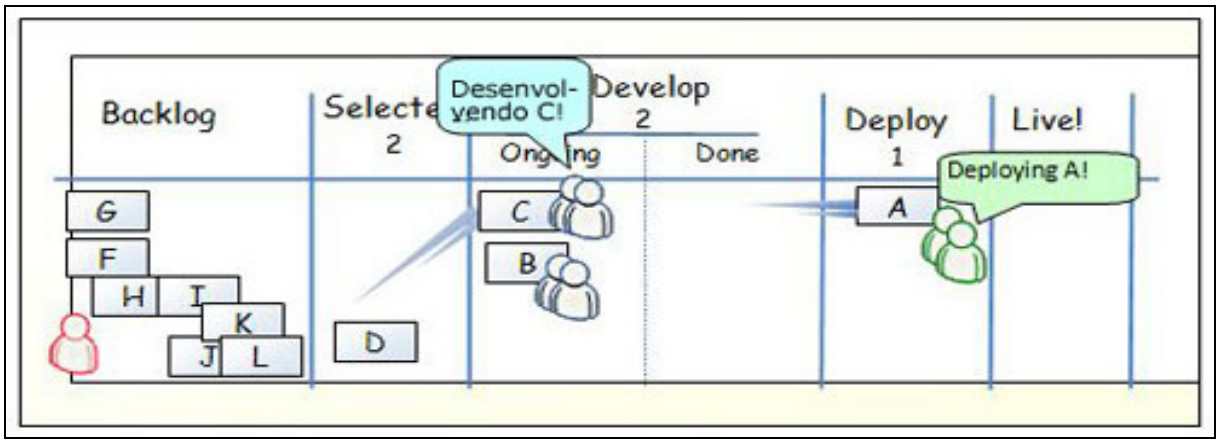
Na Figura 5 o cartão A é finalizado pelo desenvolvimento e pode-se dar inicio aos trabalhos da equipe de *deploy.*



Fonte: CASTRO (2009).

Figura 5 – Exemplo de utilização do quadro de *Kanban* parte 4

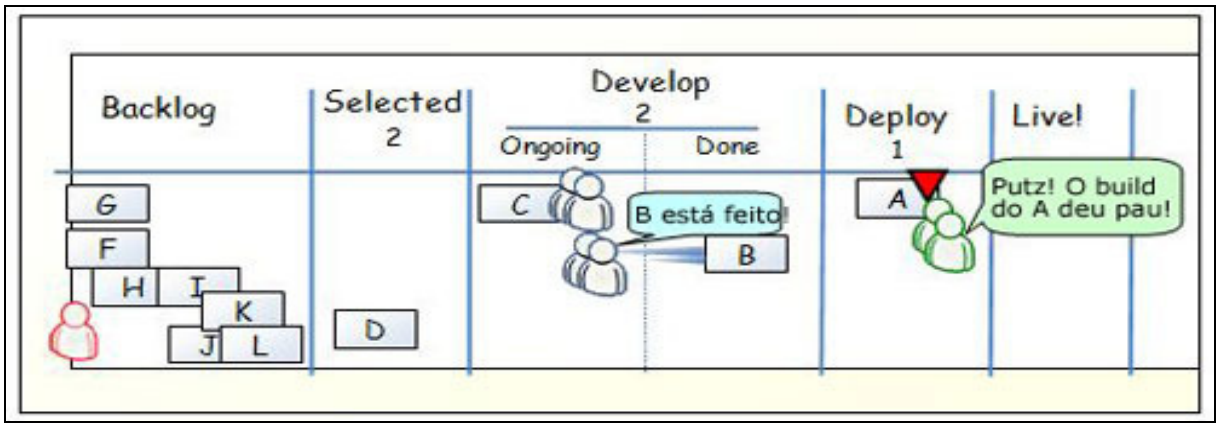
Na Figura 6 é inicia-se o desenvolvimento do cartão C enquanto é dado inicio ao *deploy* do cartão A.



Fonte: CASTRO (2009).

Figura 6 – Exemplo de utilização do quadro de *Kanban* parte 5

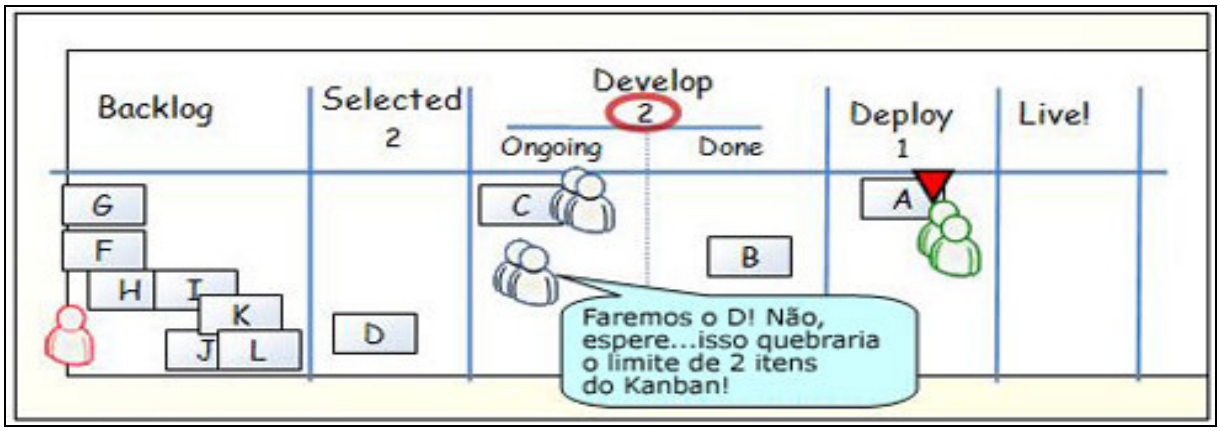
Na Figura 7 é identificado um problema no *deploy* do cartão A e o desenvolvimento do cartão B é finalizado.



Fonte: CASTRO (2009).

Figura 7 – Exemplo de utilização do quadro de *Kanban* parte 6

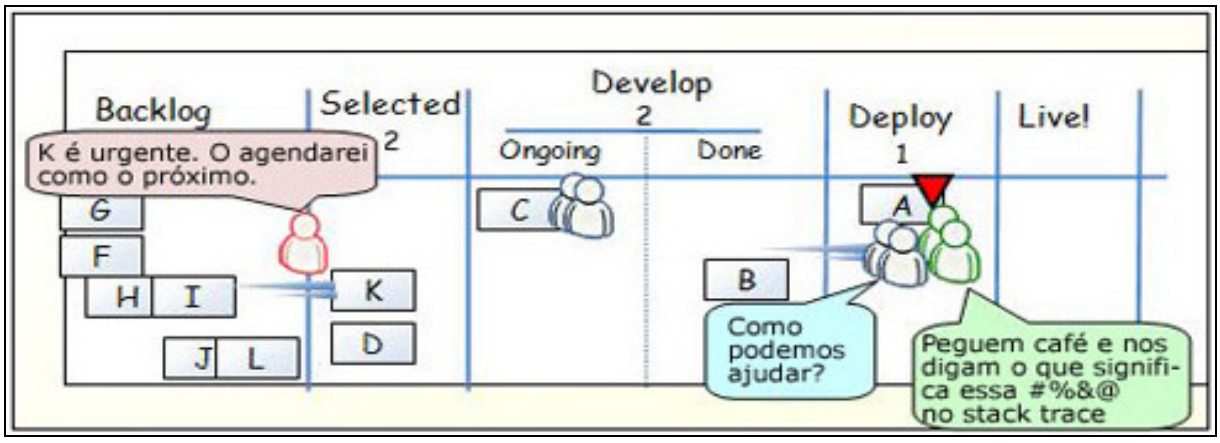
Na Figura 8 a equipe que finalizou o desenvolvimento do cartão B, pretende iniciar o cartão D, mais verifica que já está no limite do WIP da coluna.



Fonte: CASTRO (2009).

Figura 8 – Exemplo de utilização do quadro de *Kanban* parte 7

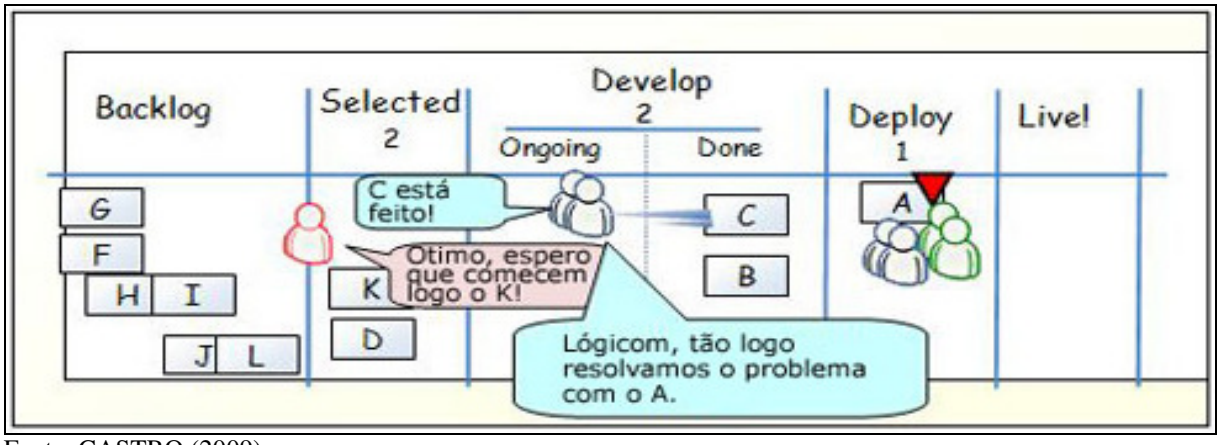
Na Figura 9 pode-se observar que a equipe de desenvolvimento entra em contato com a equipe de *deploy* para auxiliar na resolução do problema do cartão A. Enquanto a seleção de cartões continua sendo feita.



Fonte: CASTRO (2009).

Figura 9 – Exemplo de utilização do quadro de *Kanban* parte 8

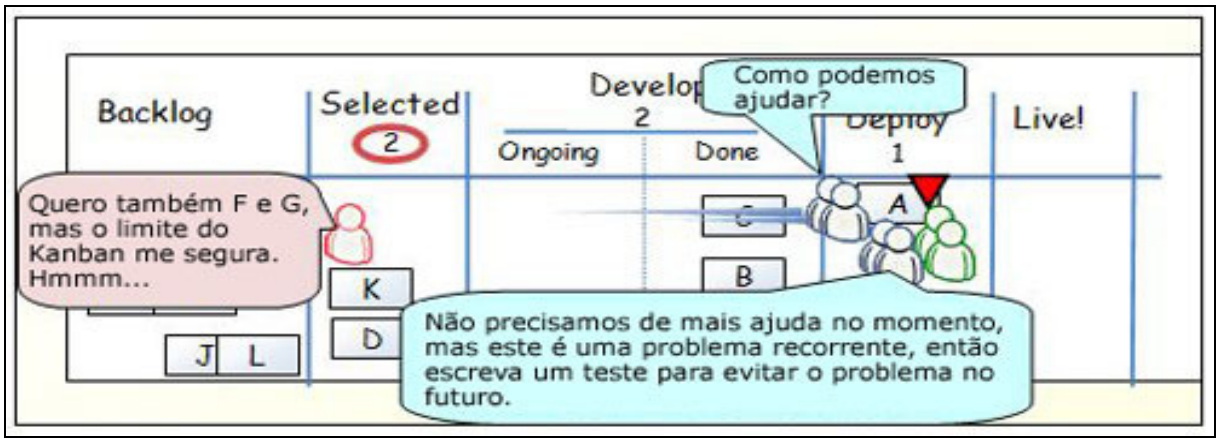
Na Figura 10 o desenvolvimento do cartão C é finalizado pela outra equipe de desenvolvimento que verificar que não pode puxar outro cartão, pois verifica que a coluna ainda está com o limite máximo de cartões.



Fonte: CASTRO (2009).

Figura 10 – Exemplo de utilização do quadro de *Kanban* parte 9

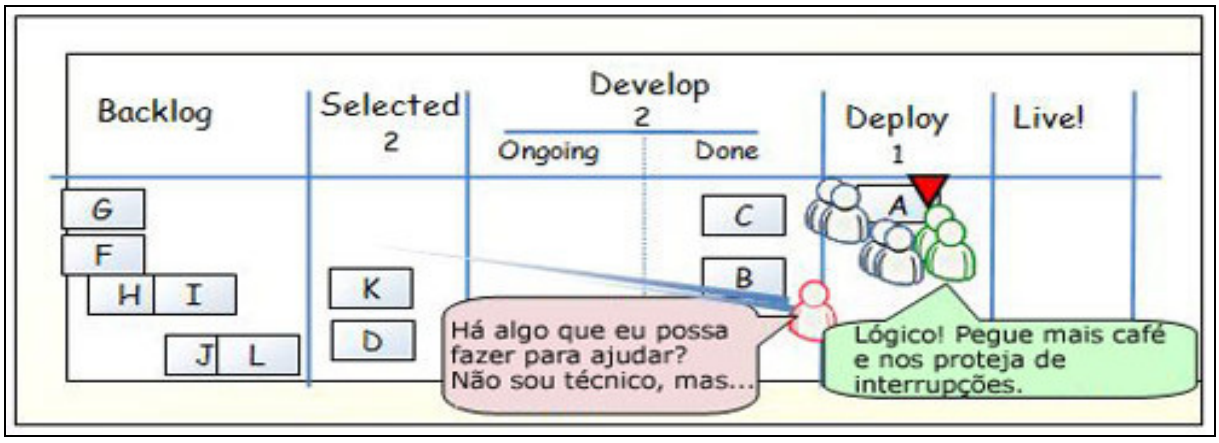
Na Figura 11 a segunda equipe de desenvolvimento se junta a equipe de *deploy* para auxiliar na solução do problema, enquanto o responsável por selecionar os cartões aguarda que os cartões D e K sejam puxados.



Fonte: CASTRO (2009).

Figura 11 – Exemplo de utilização do quadro de *Kanban* parte 10

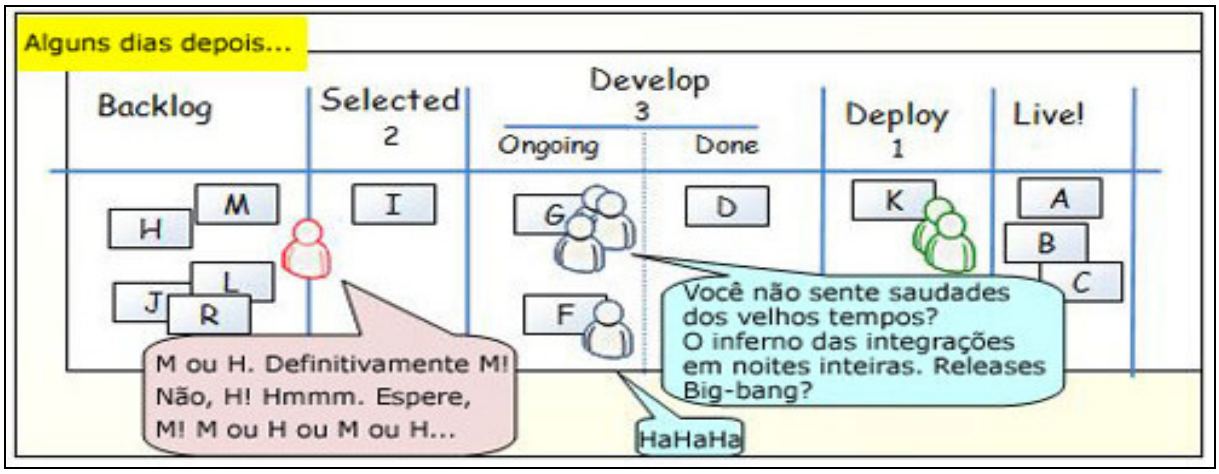
Na Figura 12 pode-se observar que todas as equipes juntam-se a equipe de *deploy* para solucionar o problema do cartão A. Neste ponto o processo de desenvolvimento foi interrompido por inteiro e é onde deve-se parar e analisar junto a equipe o porque que o processo foi interrompido e o que pode-se realizar para melhorar o processo evitando futuras interrupções. Neste exemplo o problema foi decorrente de uma falha no *deploy* do cartão A.



Fonte: CASTRO (2009).

Figura 12 – Exemplo de utilização do quadro de *Kanban* parte 11

Na Figura 13, após a solução dos problemas do cartão A é dado continuidade ao fluxo de movimentação dos cartões sobre o quadro.



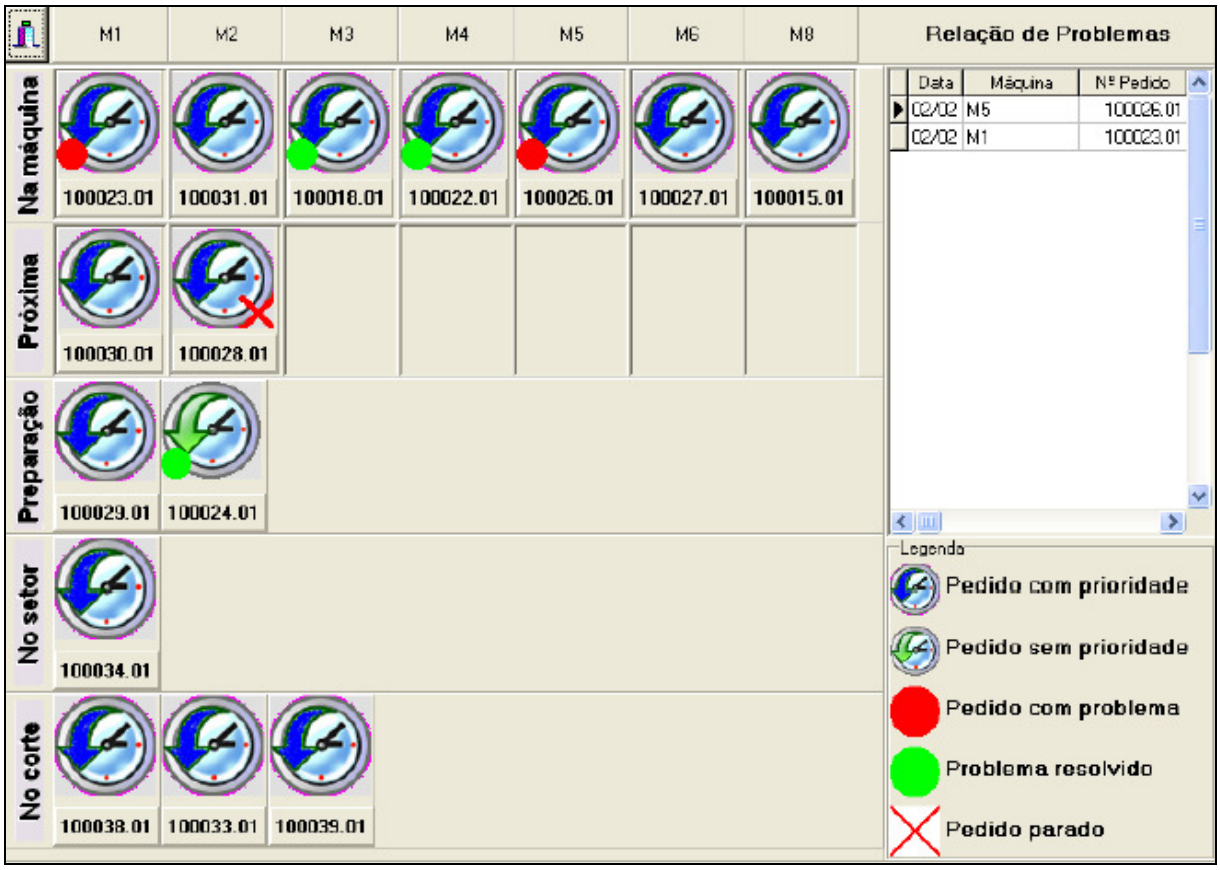
Fonte: CASTRO (2009).

Figura 13 – Exemplo de utilização do quadro de *Kanban* parte 12

2.3 TRABALHOS CORRELATOS

No trabalho de Conclusão de Curso de Alzir Wagner desenvolvido na Universidade Regional de Blumenau (FURB), o mesmo expõe um estudo sobre o departamento de Planejamento e Controle da Produção (PCP) de uma empresa do setor têxtil e como que a utilização do método de *Kanban* pode auxiliar no controle da produção de forma simples e eficiente, tendo como foco principal a fácil visualização dos desvios da produção e diminuição do tempo de processo, assim como um melhor aproveitamento dos recursos físicos (WAGNER, 2008).

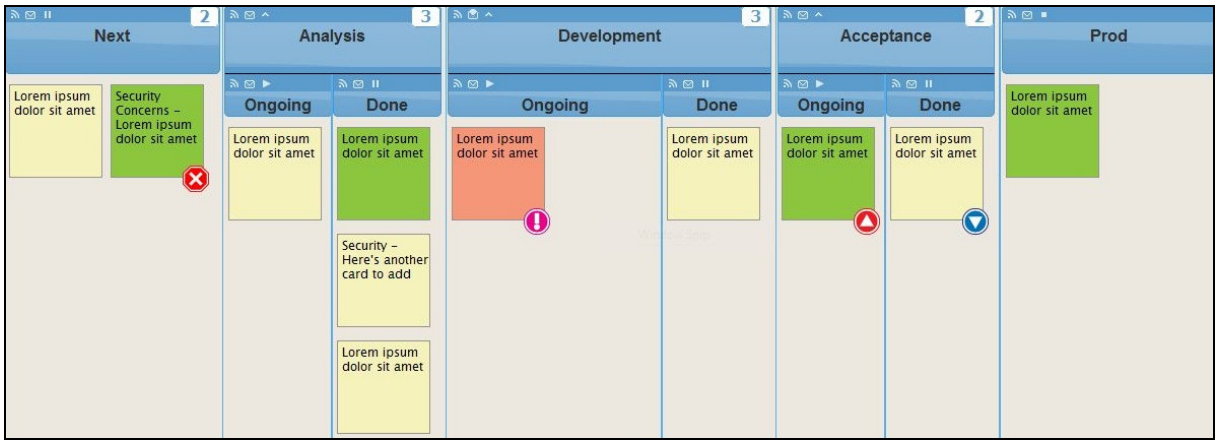
Na Figura 2 pode-se visualizar o quadro de *Kanban* desenvolvido por Wagner. A ferramenta é *desktop* e foi desenvolvida utilizando *Delphi*, nela é apresentado o quadro de *Kanban* onde pode-se visualizar por qual processo o pedido está passando e qual a sua prioridade.



Fonte: WAGNER (2008).

Figura 14 – Quadro de Kanban desenvolvido por Wagner

Outro trabalho pesquisado foi o software “LeanKit Kanban”, desenvolvido pela empresa “Bandit Software”. É um sistema *web* que utiliza técnicas do método *Kanban* para o controle de tarefa de um grupo de usuário previamente cadastrados. O sistema pode ser adquirido por US$ 19,00 mensais. Maiores informações pelo site “http://www.leankitkanban.com/Home/”. Na Figura 3 é apresentado o quadro de *Kanban* do software, nele podem-se definir as colunas e seus WIP, bem como a prioridade e cores de cada tipo de cartão (BANDIT SOFTWARE, 2010).



Fonte: BANDIT SOFTWARE (2010).

Figura 15 – Quadro do LeanKit Kanban

No trabalho de conclusão de curso de Gomes e Faias Junior (2009) foi desenvolvido o sistema "Pronto. *agile software development*", que possibilita o cadastro, gerenciamento e acompanhamento de solicitações com objetivo de atingir a satisfação dos clientes, demonstrando maior transparência do que será entregue nas versões do produto. Foi desenvolvido baseado na metodologia ágil *Scrum*. Na Figura 4 pode-se visualizar o quadro de *Kanban* do sistema.

Fonte: GOMES;FARIAS JUNIOR (2009).

Figura 16 – Quadro do *Kanban* do Pronto

3 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo são descritos as especificações da aplicação *web*, as suas características, os principais requisitos, os diagramas de caso de uso, o modelo entidade relacionamento, a operacionalidade do sistema e ao final os resultados e discussão.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Através da análise dos trabalhos correlatos, junto a vivencia no setor de desenvolvimento foi possível realizar o levantamento da funcionalidade da aplicação desenvolvida.

A aplicação desenvolvida, denominada Quick Screen, é uma aplicação *web* para auxiliar no desenvolvimento de software, proporcionando um acompanhamento das etapas do processo. O Quick Screen procura facilitar a identificação de problemas e melhorias antes do fim do processo. Para isso baseia-se no método *Kanban*, evidência de forma rápida, e a todos os envolvidos, o andamento do processo em cada etapa.

A principal funcionalidade da aplicação é o quadro de *Kanban* e suas colunas que podem ser configuradas e moldadas conforme as etapas do processo em uso e para cada coluna pode-se definir um WIP restringindo assim o número de cartões que a coluna pode receber simultaneamente. Através da análise da movimentação dos cartões sobre as colunas e verificando se o WIP está de acordo com a realidade da equipe, o Administrador tem a visão de quais etapas do processo estão mais ociosas ou sobrecarregadas e pode-se realizar mudanças e ajustes antes de finalizar o processo. Buscando manter o fluxo continuo de cartões sobre o quadro.

Os cartões são as atividades que devem ser executados pela equipe, todos os usuários com acesso ao quadro podem assumir a responsabilidade, registrar providências dos trabalhos desenvolvidos e movimenta os cartões sobre o quadro. Todas as ações realizadas com os cartões do quadro são gravadas em um histórico de movimentações do cartão e podem ser consultadas posteriormente.

A aplicação conta com dois níveis de acesso a do Administrador e do Usuário. O Administrador é o usuário com maior nível de privilégios na aplicação tendo acesso a todos os

cadastros e funcionalidade, geralmente é o individuo responsável por coordenar o processo. Os usuários são os demais membros da equipe, que de fato irão executar os trabalhos descritos nos cartões, basicamente as funcionalidades que tem acesso estão no próprio quadro.

3.2 ESPECIFICAÇÃO

Os requisitos funcionais apresentam às funcionalidades e o comportamento que o sistema deve possuir em determinadas situações. Os requisitos não funcionais apresentam as restrições que o sistema terá sobre alguns serviços ou funções oferecidas como usabilidade, navegabilidade, portabilidade, segurança e hardware. A seguir são apresentados os quadros de requisitos funcionais, requisitos não funcionais, casos de uso e modelo de entidade-relacionamento.

3.2.1 Requisitos funcionais

O Quadro 1 apresenta os requisitos funcionais previstos para o sistema e sua rastreabilidade, ou seja, vinculação com os casos de uso associados.

Requisitos Funcionais	**Caso de Uso**
RF01: A aplicação deverá permitir o cadastro de usuários.	UC01
RF02: A aplicação deverá permitir o cadastro de quadros.	UC02
RF03: A aplicação deverá permitir o cadastro de colunas.	UC03
RF04: A aplicação deverá permitir o cadastro o WIP da coluna.	UC04
RF05: A aplicação deverá permitir o cadastro de prioridades.	UC05
RF06: A aplicação deverá permitir o cadastro de tipos de cartões.	UC06
RF07: A aplicação deverá permitir o cadastro de cartões.	UC07
RF08: A aplicação deverá permitir o cadastro de providências do cartão.	UC08
RF09: A aplicação deverá permitir assumir a responsabilidade do cartão.	UC09
RF10: A aplicação deverá permitir a movimentação dos cartões sobre as colunas do quadro.	UC10
RF11: A aplicação deverá permitir a visualização do histórico de movimentações do cartão.	UC11
RF12: A aplicação deverá permitir a definição de prioridade do cartão.	UC12
RF13: A aplicação deverá permitir a geração de relatório de listagem de cartões por quadro.	UC14
RF14: A aplicação deverá permitir a geração de relatório de listagem de	UC15

usuários por setor.	

Quadro 1 – Requisitos funcionais

3.2.2 Requisitos não funcionais

O Quadro 2 lista os requisitos não funcionais previstos para o sistema.

Requisitos Não Funcionais
RNF01: A aplicação deverá ser totalmente *WEB*.
RNF02: A aplicação deverá utilizar o banco de dados MYSQL.
RNF03: A aplicação deverá ser desenvolvido utilizando o *framework* cakePHP.
RNF04: A aplicação deverá ser desenvolvido utilizando a IDE eclipse.
RNF05: A aplicação deverá suportar os navegadores Internet Explore 8, Fire Fox 5 e Chrome 12.

Quadro 2 – Requisitos não funcionais

3.2.3 Diagrama de Casos de Uso

Esta sub-seção apresenta os diagramas de casos de uso da aplicação, sendo que o detalhamento dos principais casos de uso estão descritos no Apêndice A.

Na Figura 17, tem-se o diagrama de caso de uso do Administrador.

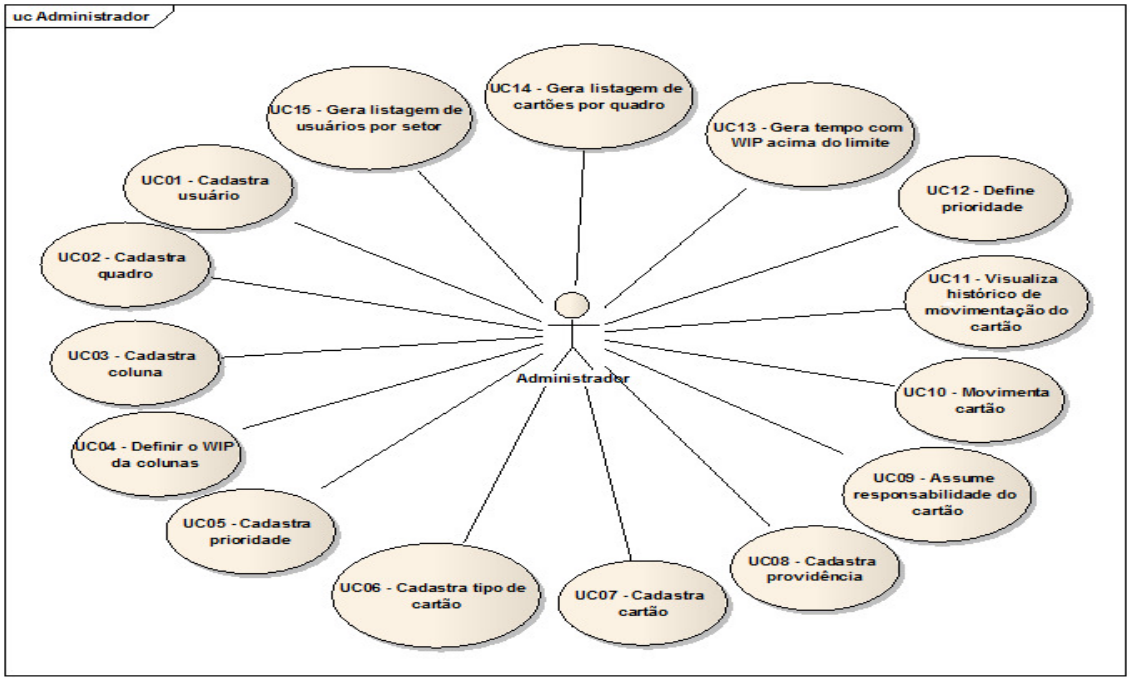


Figura 17 – Diagrama de casos de uso do Administrador

Na Figura 18, tem-se o diagrama de caso de uso do Usuário.

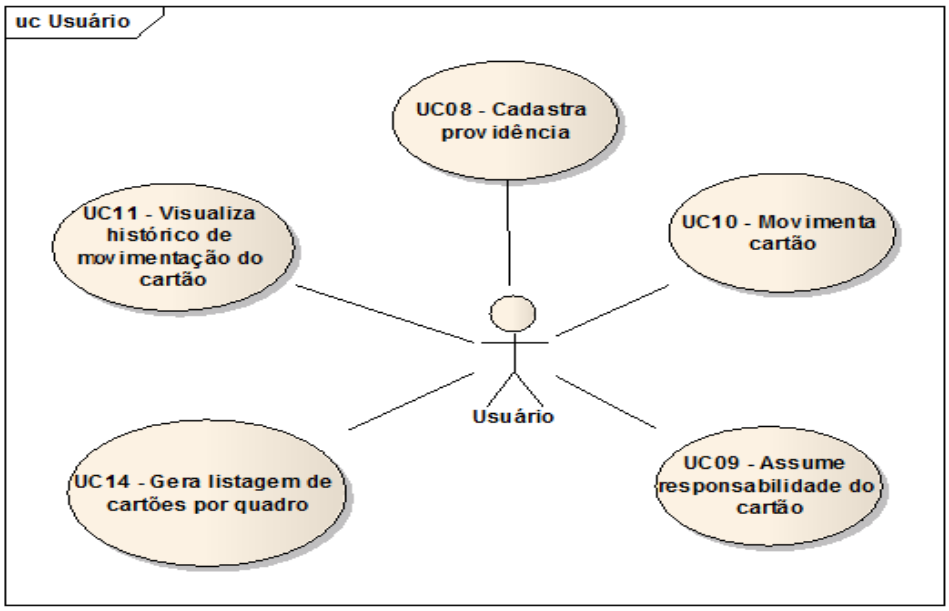


Figura 18 – Diagrama de casos de uso do Usuário

3.2.4 Modelo entidade-relacionamento

A Figura 19 apresenta o modelo entidade e relacionamento que representam as entidades que serão persistidas no banco de dados.

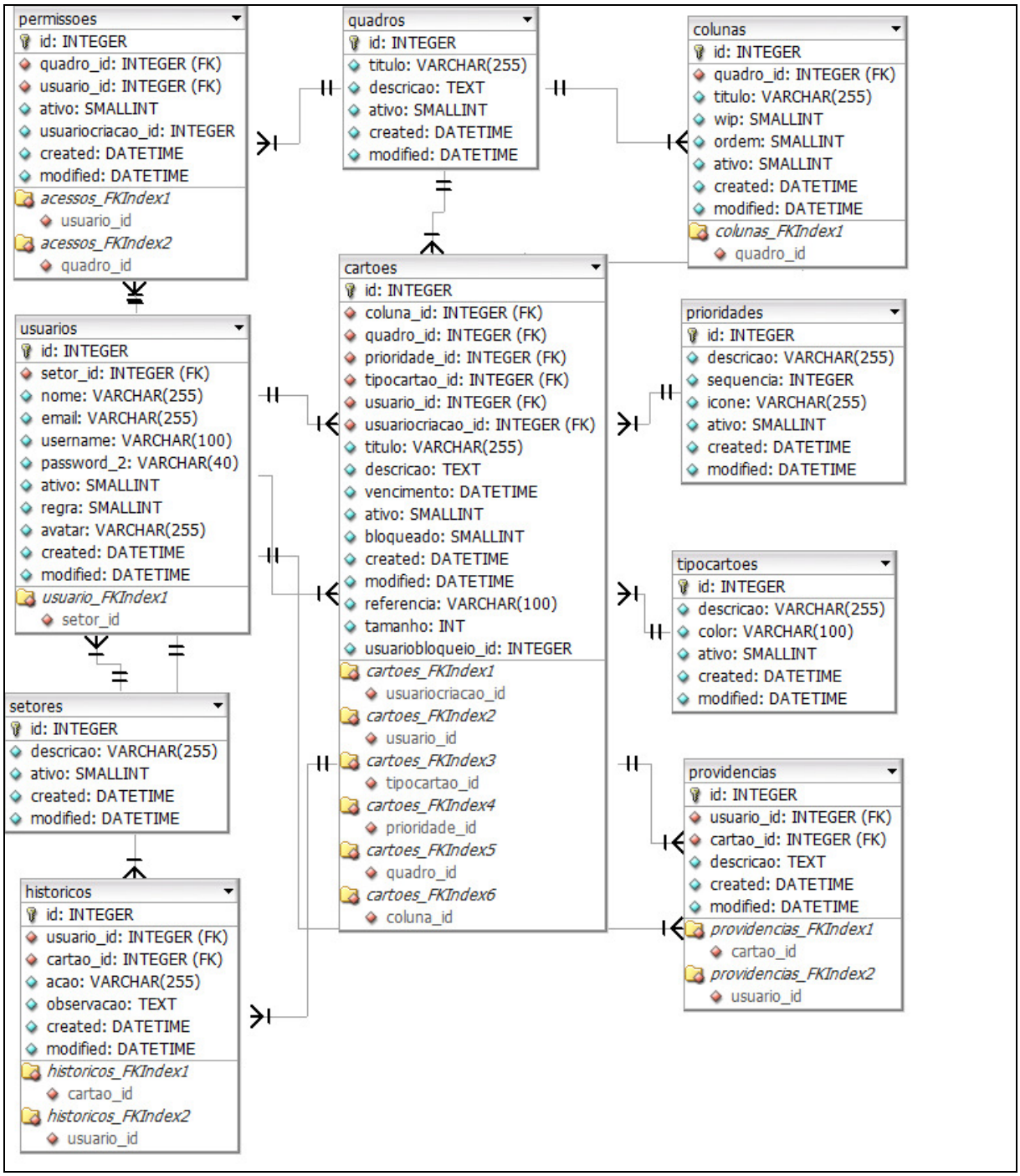


Figura 19 – Modelo entidade-relacionamento

A seguir é apresentada uma breve descrição das entidades utilizadas para o desenvolvimento do sistema, o dicionário de dados está descrito no Apêndice B:

a) setores: entidade responsável por armazenar informações referentes aos setores. Os setores são utilizados como forma de classificação dos usuários cadastrados;

b) usuarios: entidade responsável por armazenar informações referentes aos usuários;

c) permissoes: entidade responsável por armazenar informações referentes às

permissões dos usuários de acesso ao quadros;

d) quadros: entidade responsável por armazenar informações referentes aos quadros;
e) colunas: entidade responsável por armazenar informações referentes às colunas dos quadros, cada coluna corresponde a uma parte ou etapa do processo;
f) tipocartoes: entidade responsável por armazenar informações referentes aos tipos de cartões utilizados como forma de classificação do cartão para facilitar a identificação dos mesmos;
g) prioridades: entidade responsável por armazenar informações referentes às prioridades, item que define a criticidade do cartão;
h) cartoes: entidade responsável por armazenar informações referentes aos cartões, trabalhos a serem desenvolvido no durante o processo;
i) providencias: entidade responsável por armazenar informações referentes aos providencias, registros dos trabalhos efetuados por cada usuário que assumiu a responsabilidade do cartão;
j) historicos: entidade responsável por armazenar informações referentes aos históricos, registro de todas as movimentações dos cartões sobre o quadro e ações dos usuários;

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação desta aplicação.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação foi desenvolvida utilizando a linguagem de programação PHP sob a IDE de desenvolvimento eclipse juntamente com o *framework* cakePHP, *framework* jQuery e o sistema de banco de dados MySQL.

3.3.1.1 *Framework* cakePHP

Segundo o CakePHP Brasil (2007), o CakePHP é um framework escrito em PHP que tem como principais objetivos oferecer uma estrutura que possibilite aos programadores de PHP de todos os níveis desenvolverem aplicações robustas rapidamente, sem perder a flexibilidade. Desta forma o desenvolvedor irá focar seu trabalho nas regras de negócio do sistema.

O CakePHP é baseado no framework Ruby on Rails, que é um framework de código aberto para desenvolvimento de aplicações web escrito em Ruby, e utiliza padrões de desenvolvimento como o *model view controller* (MVC), que permite separar a lógica da aplicação, da interface do usuário e do fluxo da aplicação.

O MVC é uma arquitetura ou padrão que permite dividir as funcionalidades do sistema em três camadas *model, view e controller*, essa divisão é realizada para facilitar resolução de problemas maiores, cada camada desempenha sua respectiva função:

a) o modelo (*Model*) é utilizado para manipular informações de forma mais detalhada, sendo recomendado que, sempre que possível, se utilize dos modelos para realizar consultas, cálculos e todas as regras de negócio do nosso site ou sistema. É o modelo que tem acesso a toda e qualquer informação sendo essa vinda de um banco de dados, arquivo XML;
b) a visão (*view*) é responsável por tudo que o usuário final visualiza, toda a interface, informação, não importando sua fonte de origem, é exibida graças a camada de visão;
c) a controladora (*controller*), como o nome já sugere, é responsável por controlar todo o fluxo de informação que passa pelo sistema. É na controladora que se decide “se”, “o que”, “quando” e “onde” deve funcionar. Define quais informações devem ser geradas, quais regras devem ser acionadas e para onde as informações devem ir, é na controladora que essas operações devem ser executadas. Em resumo, é a controladora que executa uma regra de negócio (*model*) e repassa a informação para a visualização (*view*).

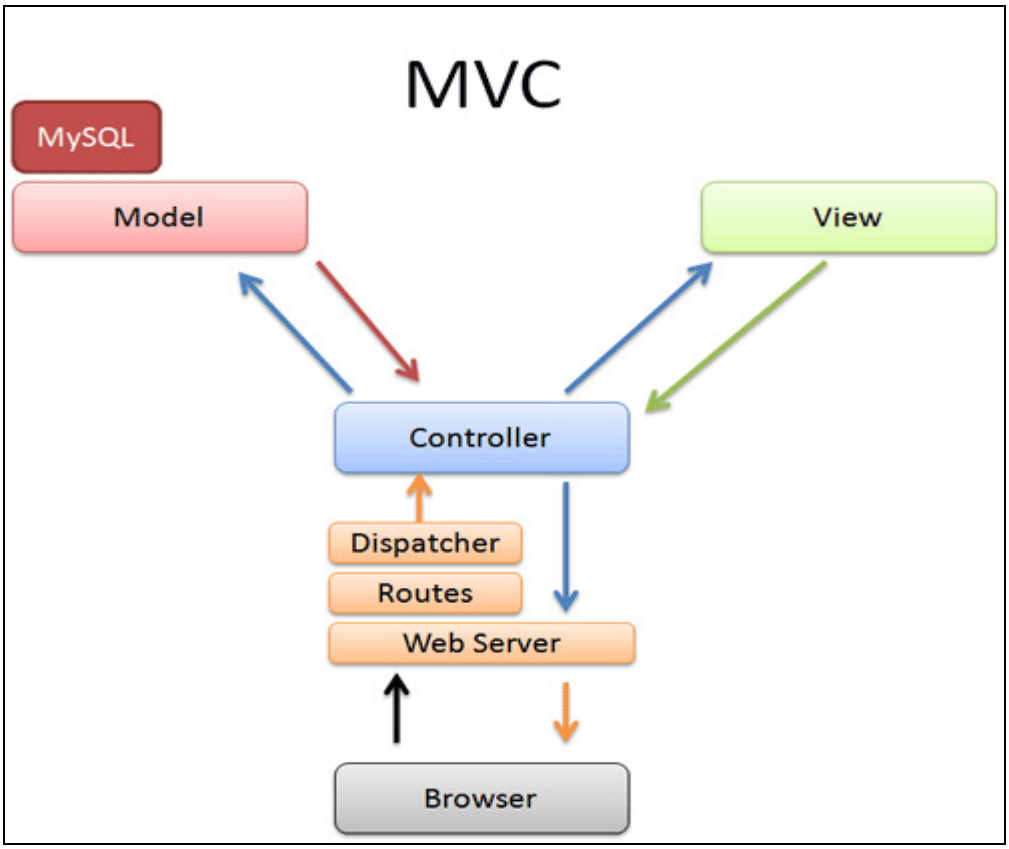
A Figura 20 apresenta o fluxograma de funcionamento do padrão MVC:

a) a *view* recebe uma interação (por exemplo, o usuário aperta um botão);
b) o *controller* manipula o evento da *view* através de uma rotina pré-escrita;

c) o *controller* acessa o m*odel*, possivelmente atualizando-o de uma maneira apropriada, baseado na interação (por exemplo, atualizando os dados de cadastro do usuário);

d) algumas implementações de *view* utilizam o *model* para gerar uma interface apropriada (por exemplo, mostrando na tela os dados que foram alterados juntamente com uma confirmação). O *view* obtém seus próprios dados do *model*. O *model* não toma conhecimento direto da *view*;

e) a interface do usuário espera por próximas interações, que iniciarão o ciclo novamente.



Fonte: BETTER EXPLAINED (2007).

Figura 20 – Fluxograma de funcionamento do MVC

Para ilustrar o desenvolvimento na Figura 21 é apresentado um trecho do código fonte da classe *model* responsável pelos quadros.

```php
<?php
class Quadro extends AppModel {
    var $name = 'Quadro';
    var $displayField = 'titulo';
    var $validate = array(
        'coluna_id' => array(
            'numeric' => array(
                'rule' => array('numeric'),
                //'message' => 'Sua mensagem de validação aqui',
                //'allowEmpty' => false,
                //'required' => false,
                //'last' => false, // Para a validação após esta regra
                //'on' => 'create', // Limitar a validação para as operações 'create' ou 'update'
            ),
        )
    );
    var $hasMany = array(
        'Cartao' => array(
            'className' => 'Cartao',
            'foreignKey' => 'quadro_id',
            'dependent' => false,
            'conditions' => '',
            'fields' => '',
            'order' => '',
            'limit' => '',
            'offset' => '',
            'exclusive' => '',
            'finderQuery' => '',
            'counterQuery' => ''
        ),
        'Coluna' => array(
            'className' => 'Coluna',
            'foreignKey' => 'quadro_id',
            'dependent' => false,
            'conditions' => '',
            'fields' => '',
            'order' => '',
            'limit' => '',
            'offset' => '',
            'exclusive' => '',
            'finderQuery' => '',
            'counterQuery' => ''
        )
    );
```

Figura 21 – Fonte classe *model* dos quadros

Na figura 22 é demonstrado um trecho do código fonte da classe *controller* dos quadros. Neste pode-se verificar os métodos *index, add e edit* que serão invocados posteriormente através de *actions* via URL.

```php
<?php
class QuadrosController extends AppController {

    var $uses = array('Quadro', 'Usuario', 'Prioridade', 'Tipocartao', 'Coluna', 'Cartao');

    function index()
    {
        $this->Quadro->recursive = 0;
        $this->set('quadros', $this->paginate());
    }

    function add()
    {
        if (!empty($this->data)) {
            $this->Quadro->create();
            if ($this->Quadro->save($this->data))
                $this->Session->setFlash('Registro gravado com sucesso.', 'default', array('class' => 'success'));
            else
                $this->Session->setFlash('Registro gravado com sucesso.', 'default', array('class' => 'error'));

            $this->redirect(array('controller' => 'quadros', 'action' => 'index'));
        }
    }

    function edit($id = null) {
        if (!$id && empty($this->data)) {
            $this->flash(sprintf(__('%s inválido.', true), 'Quadro'), array('action' => 'index'));
        }
        if (!empty($this->data)) {
            if ($this->Quadro->save($this->data)) {
                $this->flash(sprintf(__('O %s foi salvo.', true), 'quadro'), array('action' => 'index'));
            } else {
            }
        }
        if (empty($this->data)) {
            $this->data = $this->Quadro->read(null, $id);
        }
        $backlogs = $this->Quadro->Backlog->find('list');
        $raias = $this->Quadro->Raia->find('list');
        $this->set(compact('backlogs', 'raias'));
    }
```

Figura 22 – Fonte classe *controller* dos quadros

Na figura 23 pode-se visualizar um trecho do código fonte da classe *view* do quadro, responsável pela camada de visão que é apresentada ao usuário.

```
<?php echo $crumb->getHtml('Home', 'reset' );  ?>
<div id="titulo">
    <span>Quadros</span>
    <hr />
</div>
<div class="actions">
    <?php echo $html->image("menu/novo.png", array("alt" => "Nova prioridade", "title" => "Nova prioridade",'url' => array('controller' => 'quadros', 'action' => 'add'))); ?>
</div>
<div class="quadros index">
    <?php
    $i = 0;
    foreach ($quadros as $quadro):
    ?>
    <div class="qs-quadro">
        <div class="qs-quadro-in">
            <div class="qs-quadro-info">
                <strong><?php echo $this->Html->link($quadro['Quadro']['titulo'], array('action' => 'view', $quadro['Quadro']['id'])); ?></strong><br />
                <span><?php echo $quadro['Quadro']['descricao']; ?></span>
            </div>
            <div class="qs-quadro-acoes">
                <div class="qs-quadro-acoes-menu kbm-edit kb-grid-custom">
                    <?php echo $html->image("editar.png", array("alt" => "Editar quadro",'url' => array('controller' => 'quadros', 'action' => 'editName',
                    $quadro['Quadro']['id']))); ?>Editar
                </div>
                <div class="qs-quadro-acoes-menu kbm-history kb-grid-custom">
                    <a href="/Kanban/2">
                        <?php echo $html->image('historico.png', array('alt' => 'CakePHP'))?>History
                    </a>
                </div>
            </div>
        </div>
    </div>
    <?php endforeach; ?>
</div>
```

Figura 23 – Fonte classe *view* dos quadros

3.3.1.1 Biblioteca jQuery

Segundo Manor (2010), a biblioteca jQuery utiliza um seletor simples como forma de pegar e manipular elementos *Document Object Model* (DOM). Para os seletores pode-se utilizar as próprias *tags* do HTML como H2, P, SPAN entre outras ou através de atributos como ID e NAME. Quando um elemento DOM é obtido, ele se torna um objeto jQuery e qualquer método da biblioteca padrão ou de extensões do jQuery podem ser invocados desses objetos. O jQuery possibilita o uso de seletores complexos, exemplo: elementos baseados em relacionamento pai-filho, atributos, classes, filtros. O jQuery busca ser simples e intuitiva. Ela inclui:

a) navegação da árvore DOM;
b) manipulação de elementos DOM;
c) manipulação de CSS;
d) gerenciamento de Eventos (*onclick, onsubmit*);

e) interação com Ajax;
f) funções utilitárias;
g) biblioteca UI;
h) biblioteca de Efeitos.

A finalidade do uso de jQuery é controlar o comportamento de toda ou parte de uma página *web*. Sabe-se que uma página *web* nada mais é do que marcação HTML. Então, é lícito concluir que o princípio de funcionamento de jQuery fundamenta-se em sua capacidade de encontrar os elementos HTML que constituem a página e a estes anexar seus métodos. (SILVA, 2010)

Para ilustrar as facilidades de utilização desta biblioteca como exemplo imagina-se uma determinada página *web* que possui diversas *tags* HTML do tipo DIV escondidas e pretende-se adicionar uma classe a estas DIVS e torná-las visíveis. Na Figura 24 pode-se visualizar o código fonte necessário para realizar esta ação utilizando apenas *javascript* e na Figura 25 a mesma ação, más utilizando o jQuery. Percebe-se que com o jQuery utiliza-se apenas uma linha e a escrita do fonte se torna muito mais intuitiva.

```
var divs = document.getElementsByTagName("div");
for (var i = 0; i ++; divs.length; i++) {
        div[i].className = 'ativo';
        div[i].display = 'block';
}
```

Figura 24 – Código fonte utilizando javascript

```
$("div").addClass('ativo').show();
```

Figura 25 – Código fonte utilizando jQuery

O jQuery foi de suma importância no desenvolvimento deste trabalho, com o auxilio de *plugins* foi possível desenvolver o efeito de *drag and drop*, além de facilitar na compatibilidade da aplicação com diversos tipos de navegadores. Na Figura 26 pode-se verificar um trecho do código é construído o método de *drag and drop* das colunas. Na linha cinco pode-se notar que a atribuição do método *sortable* é feito utilizando o seletor do jQuery para todos os elementos que contenham o atributo *class* com o valor *sortable*.

```
$(document).ready(function(){
    var atualizaQuadroBloq = 0;
    var columnAnt = '';
    //atualiza sortable de cartões
    $(".sortable").sortable({
        connectWith: ".sortable",
        cancel: ".bloqueado",
        items: ".cartao",
        scroll: true,
        revert: true,
        beforeStop: function(event, ui) {
            columnAnt = $(this).last().attr("titulo"); wipAnt = $(this).last().attr("wip");
        },
        receive: function(event, ui) {
            if(ui.item.attr("id") != $(this).attr("id"))
            {
                var wip = null;
                if(($(this).find('.wip').html()) && ($(this).find('.cartao').size() > $(this).find('.wip').html()))
                {
                    wip = prompt('Executando esta ação fará com que a coluna '+$(this).attr('titulo')+' ultrapasse o seu WIP (Work
                    if(wip) {
                        historico($('#usersession').val(), ui.item.attr('id'), 'WIP', wip);
                        $(this).css({'background-color': '#FFB09D'});
                        $(this).find('.wip').css({'background-color': '#FF3300', 'color': '#FFFFFF'});
                        $(this).attr('wip','1');
                    } else {
                        location.reload();
                        return;
                    }
                } else {
                    $(this).attr('wip','0');
                    atualizaColunaWip();
                }
                $.ajax({ type: "POST", url: "/quickScreen/cartoes/editColunaCartao/"+ui.item.attr("id")+"/"+$(this).attr("id") });
                historico($("#usersession").val(), ui.item.attr("id"), "MOVE", columnAnt+' para '+$(this).attr('titulo'));
                if(wipAnt == 1 && $(this).attr("wip") == 0)
                    alert('baixou o wip');
            }
        },
    });
```

Figura 26 – Código fonte método *drag and drop*

3.3.2 Operacionalidade da implementação

O acesso ao sistema e feito através de *login* e senha, conforme o perfil do usuário o sistema disponibiliza o acesso as funcionalidades. A tela de *login* conforme a Figura 27 é comum a todos os usuários.

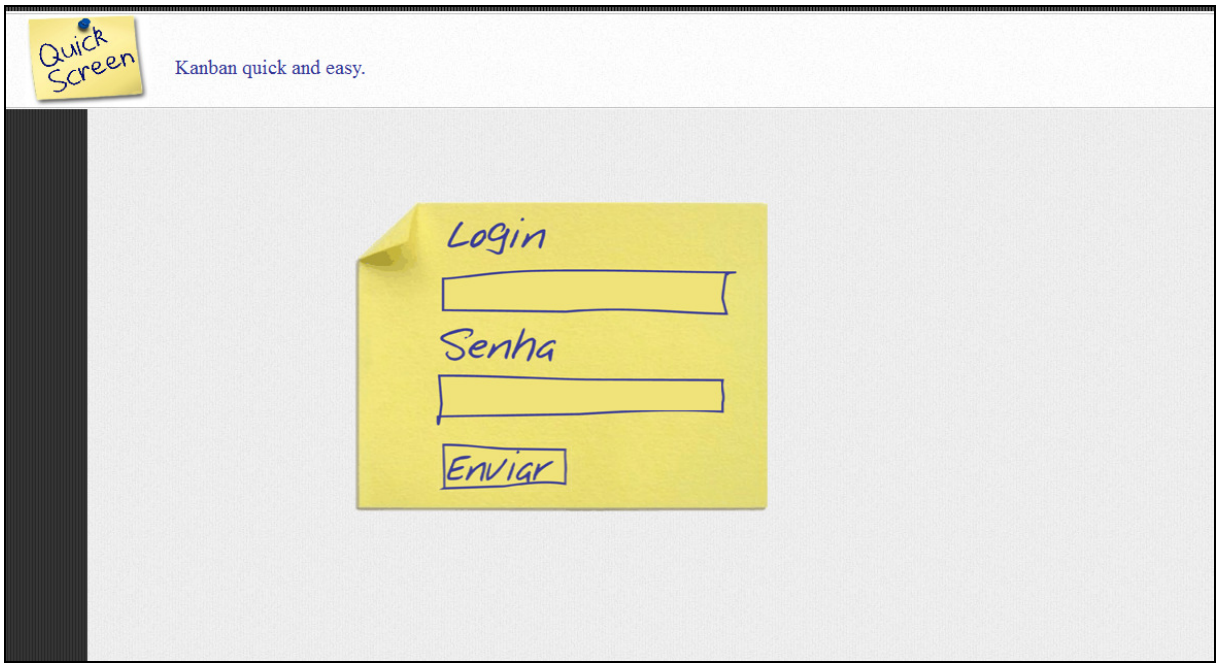


Figura 27– Tela de *login*

Caso o *login* ou senha não estejam corretos, o sistema mostra na mesma tela de *login* o erro ocorrido, como mostra a Figura 28.

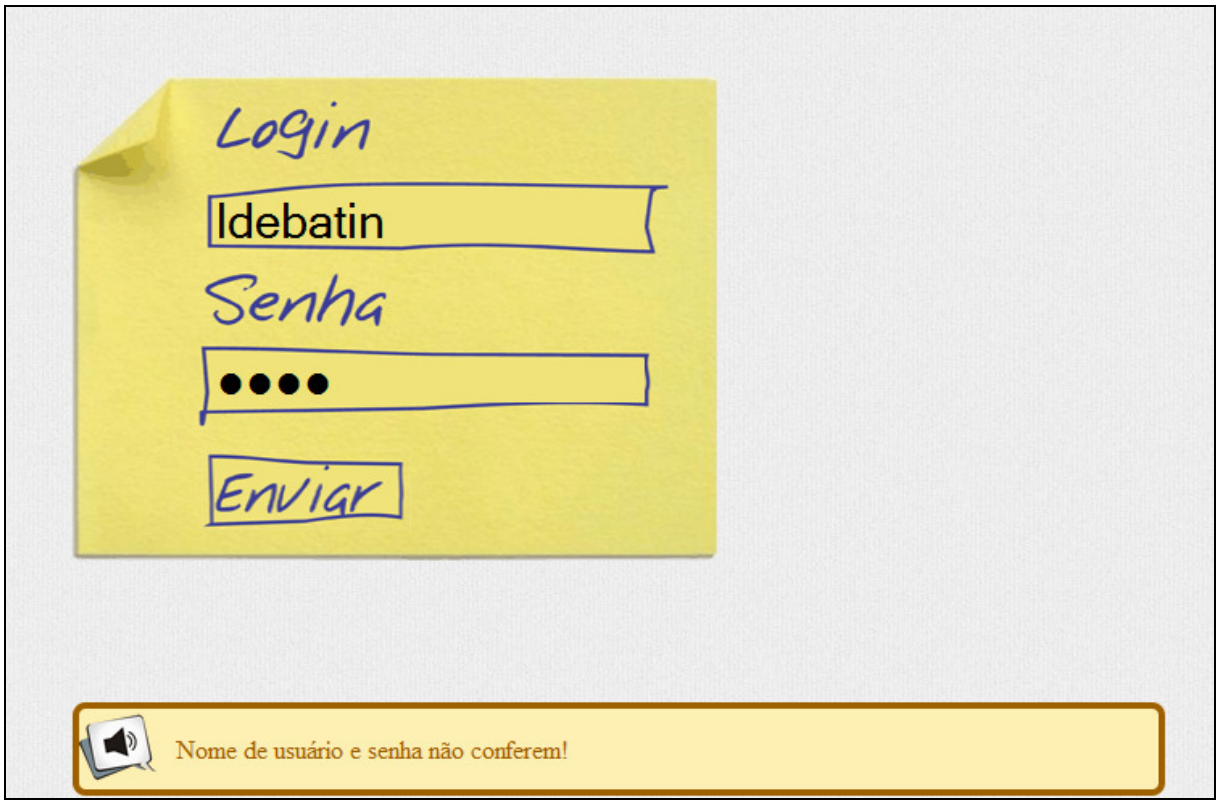


Figura 28– Falha no *login* ou senha.

Caso o *login* e senha estejam corretos o usuário acessa a página principal contendo o menu personalizado de acordo com o perfil do usuário (Administrador, Usuário). Na Figura 29 é mostrada a tela inicial do sistema.

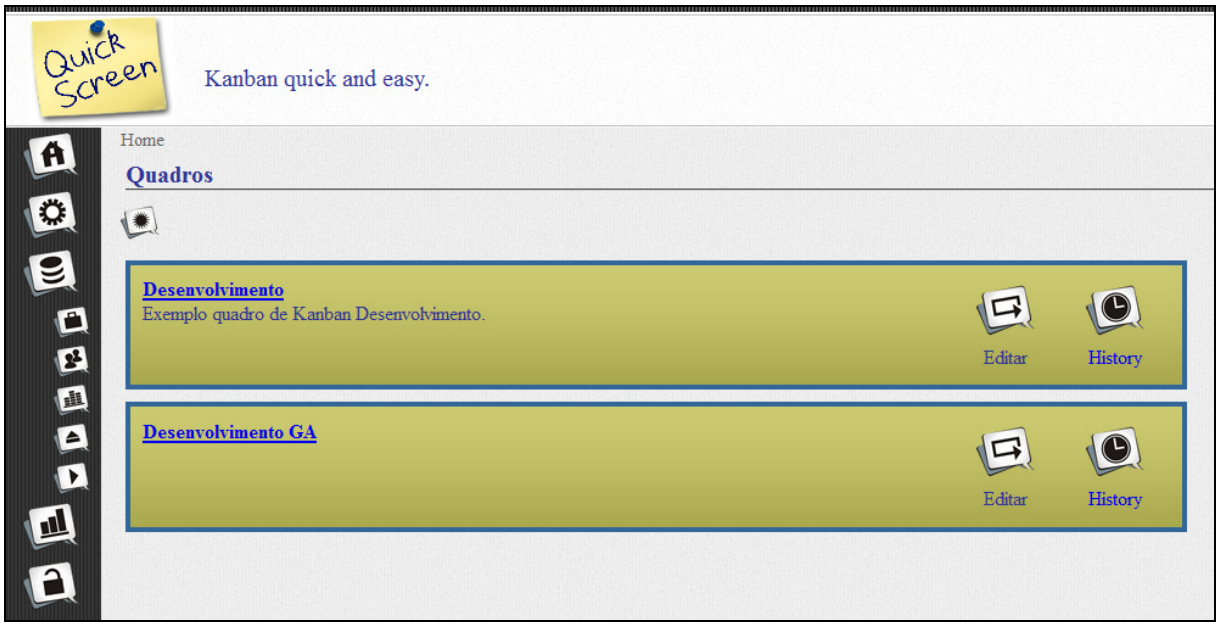


Figura 29 – Página inicial do sistema

Na Figura 30 são descritas as opções do menu do sistema.

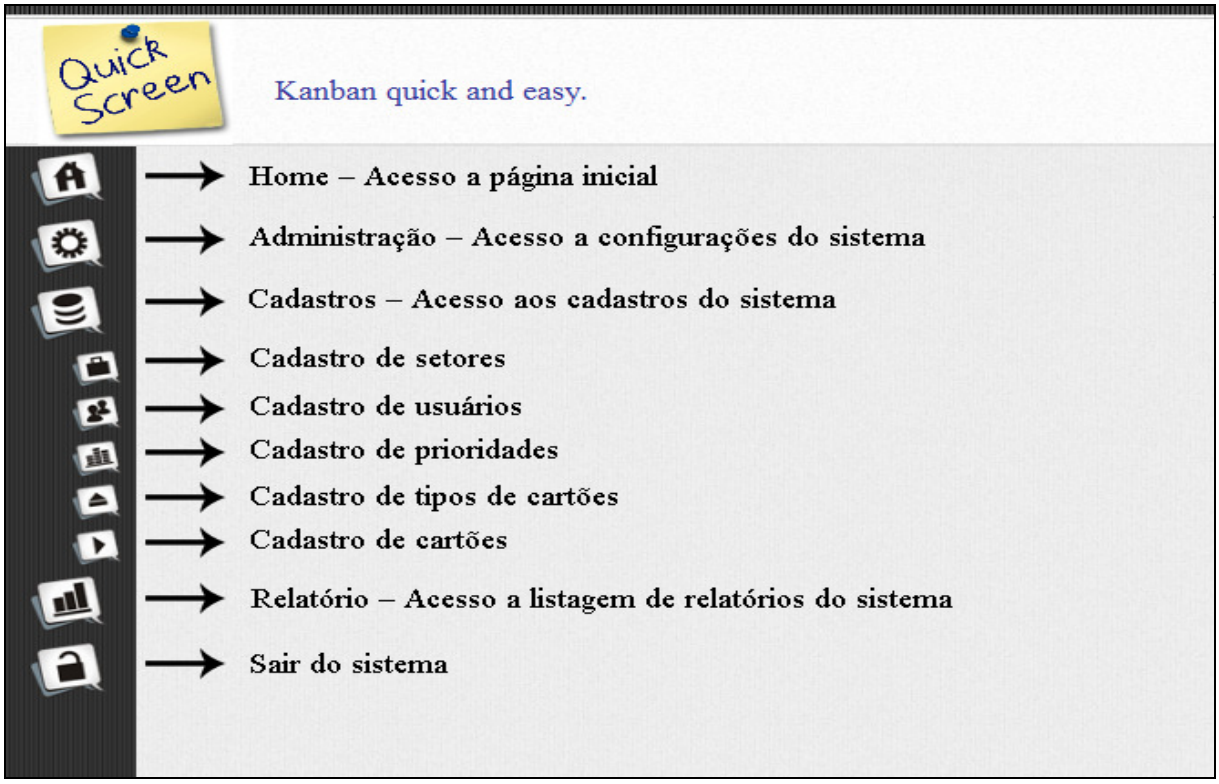


Figura 30 – Home – Acesso a página inicial

Todas as telas de cadastro do sistema seguem o mesmo padrão para listagem, cadastro, edição e exclusão de registro.

Acessando os cadastros do sistema os usuários são levados a tela de listagem de registros, como mostra a Figura 31.

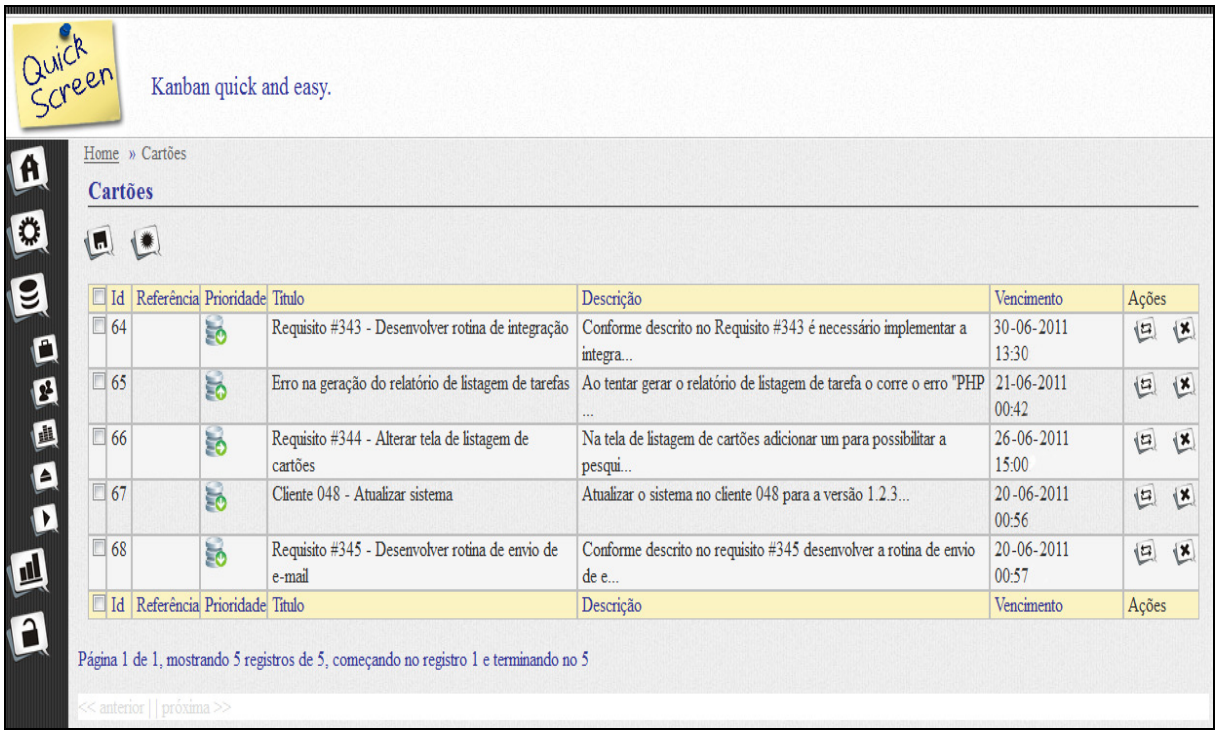


Figura 31 – Tela de listagem de cartões cadastrados

Clicando sobre o botão excluir ou selecionando as *checkbox* no canto esquerdo e clicando sobre o ícone salvar, pode-se excluir os registros a partir desta tela. Ao efetuar a exclusão o sistema apresenta a mensagem de registros excluídos conforme mostra a Figura 32.

Figura 32 – Mensagem de exclusão de registro

Clicando sobre o ícone novo registro o usuário é direcionado a tela de cadastro onde deve informar os campos e clicar sobre o botão salvar, conforme a Figura 33.

Figura 33 – Tela de cadastro de cartões

Após clicar sobre o botão salvar o usuário direcionado novamente para a tela de listagem de registros e o sistema apresenta a mensagem de registro gravado, conforme a Figura 34.

Figura 34 – Mensagem registro gravado

Na Figura 35 é apresentada a tela de edição onde pode-se editar as informações dos registros cadastrados e salvar as alterações. Após salva a alteração o usuário é direcionado

para a tela de listagem de registro.

Figura 35 – Tela de edição de cartões

Na Figura 36 é apresentada a tela do quadro de *Kanban*, principal funcionalidade do sistema. Nesta tela o usuário tem acesso ao cadastro e edição de cartões, *Backlog* do quadro, colunas do quadro, registro de providências do cartão, histórico de movimentação do cartão e movimentar os cartões sobre as colunas. Cada cartão recebe uma referente ao seu tipo que pode ser customizado no cadastro de tipos de cartões. Quando um usuário assume a responsabilidade sobre um cartão o mesmo é marcado com o avatar do usuário, customizado no cadastro de usuários.

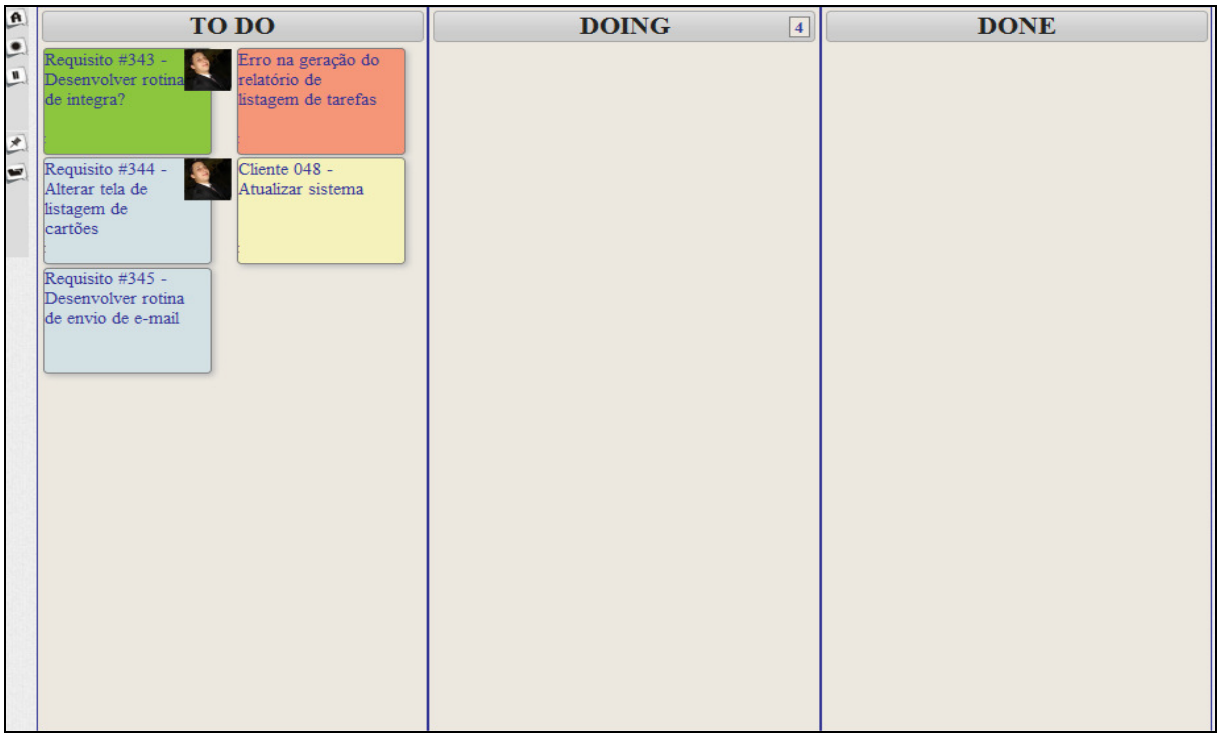


Figura 36 - Tela principal do quadro de *Kanban*

Na Figura 37 é apresentada a tela de *backlog* do quadro, onde ficam os cartões enquanto não estão em uma etapa do quadro.

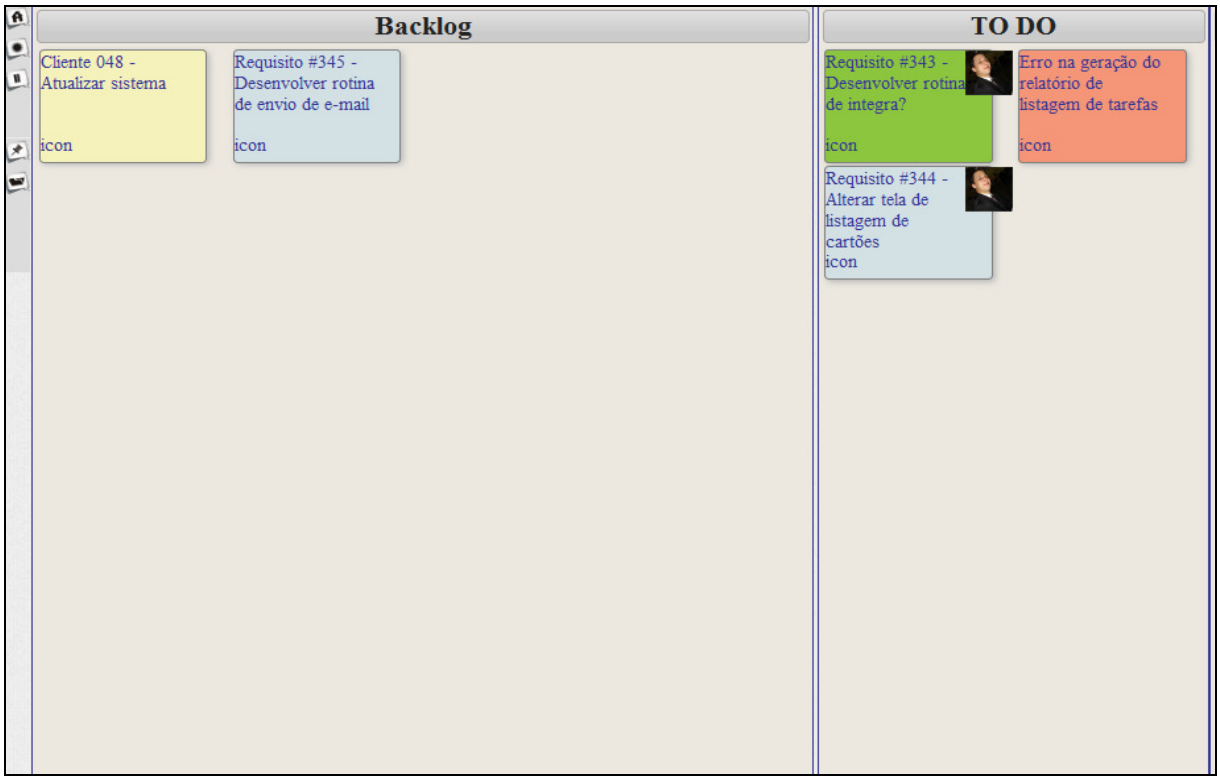


Figura 37 - Tela de visualização do *backlog* do quadro

O cadastro e edição de cartões também podem ser feitos através do quadro de *Kanban,* através das telas apresentadas nas Figuras 38 e 39. Os campos de cadastro não se alteram nessas telas.

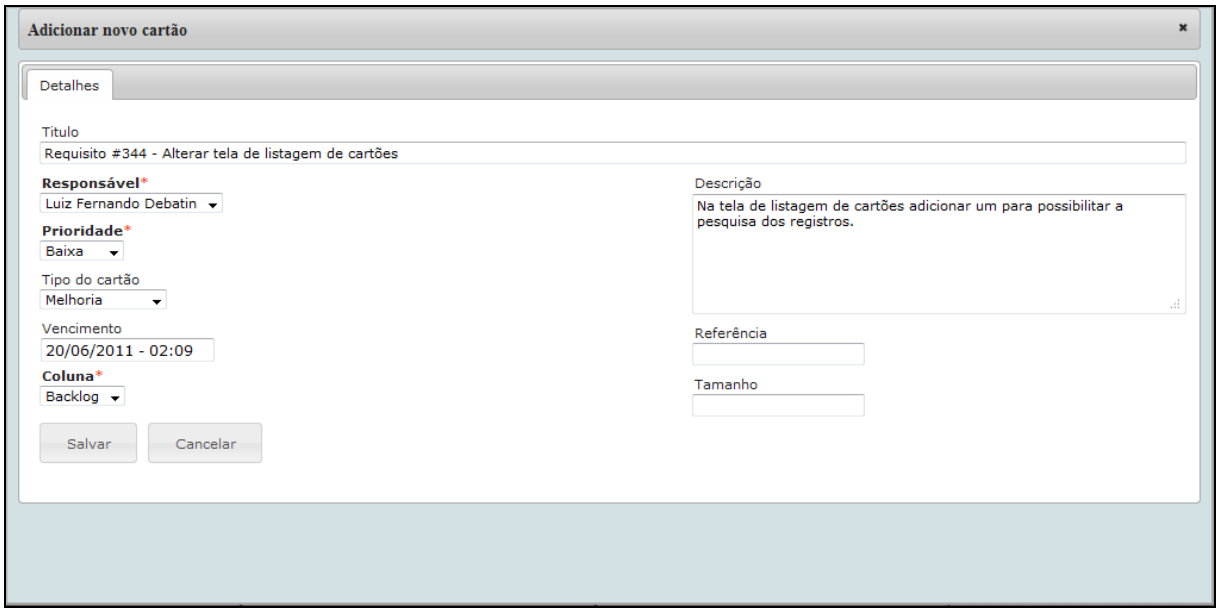


Figura 38 - Tela de cadastro de cartão através do quadro de *Kanban*

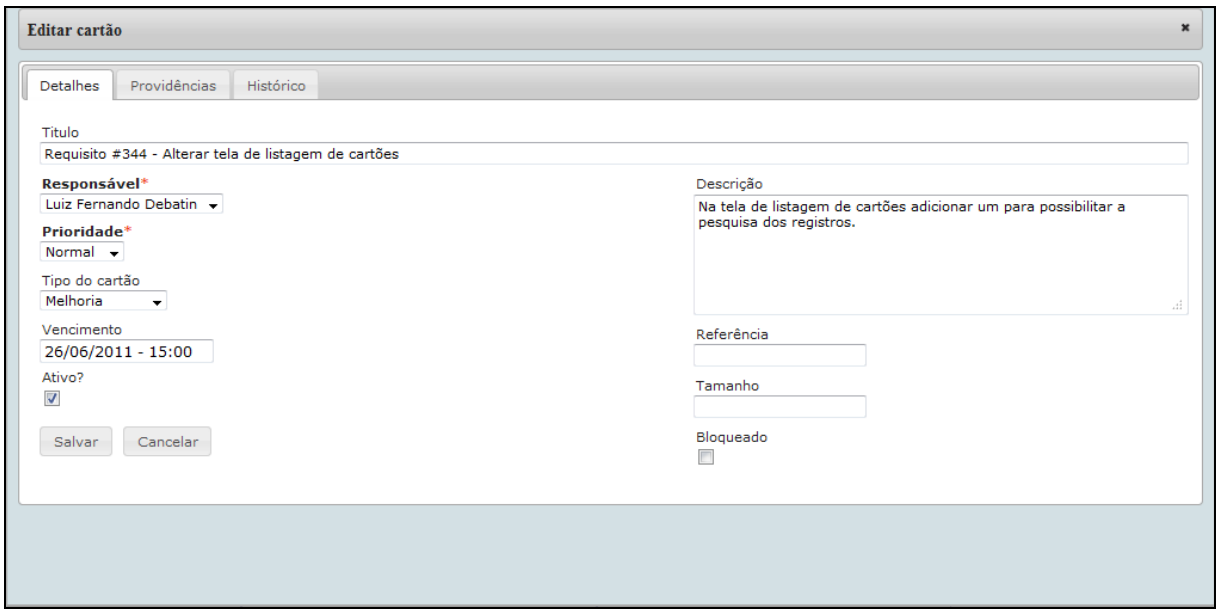


Figura 39 - Tela de edição de cartão através do quadro de *Kanban*

Na Figura 40 é apresentada a tela de registro de providências do cartão, ou seja, a descrição dos trabalhos realizados.

Figura 40 - Tela de registro de providências do cartão

Na Figura 41 é apresentada a tela de visualização do histórico de movimentação e registros do cartão.

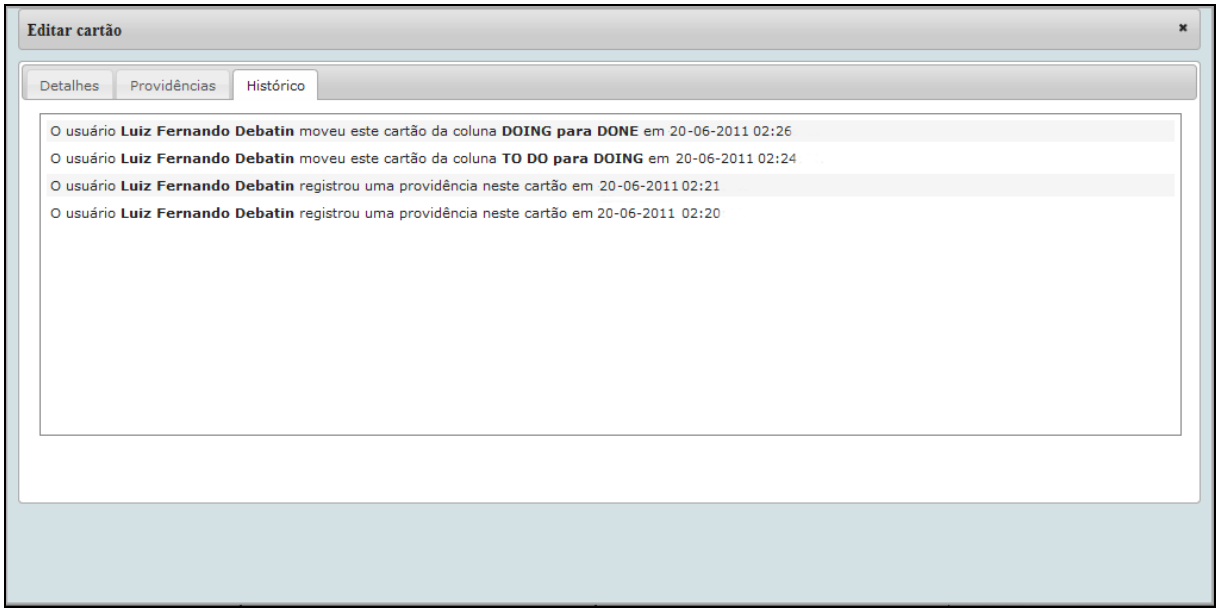


Figura 41 - Tela de visualização do histórico de movimentação do cartão

Na Figura 42 é apresentado o relatório de listagem de cartões por quadro.

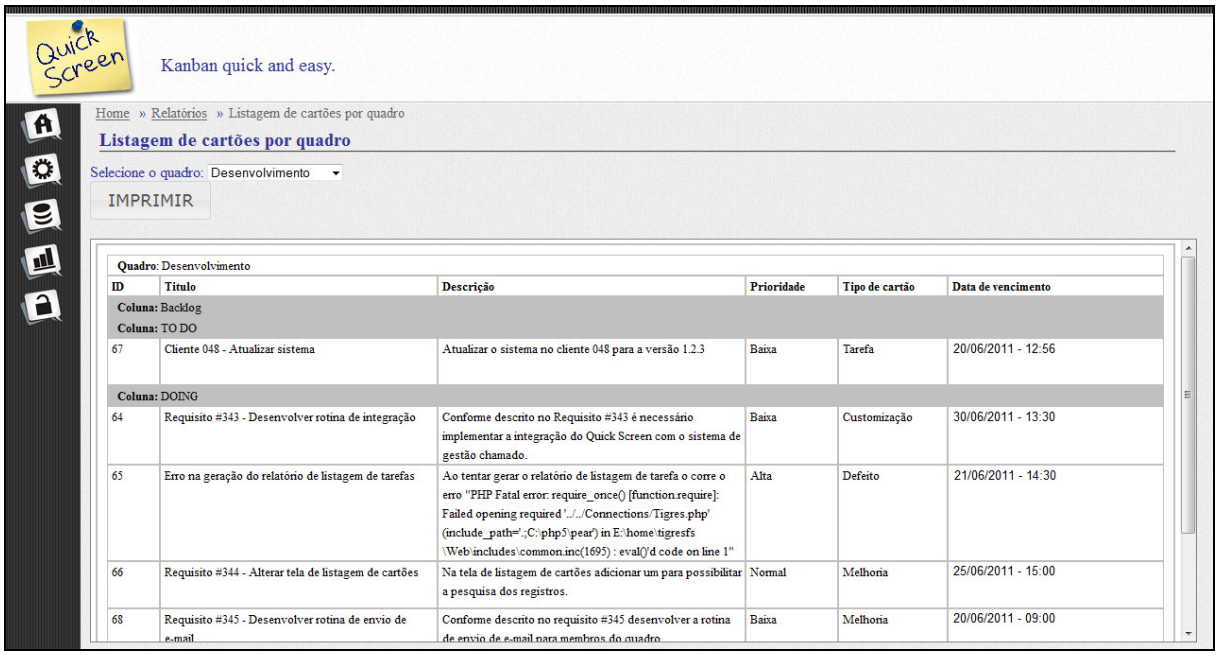


Figura 42 – Relatório de listagem de cartões por quadro

Na Figura 43 é apresentado o relatório de listagem de usuários por setor.

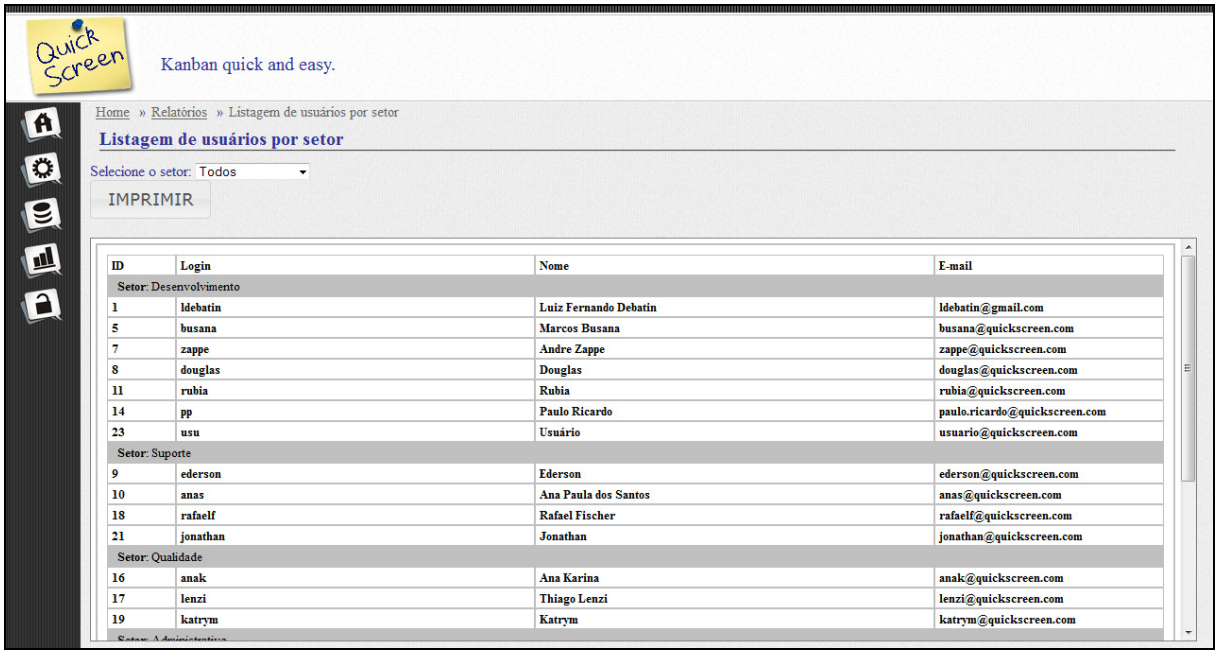


Figura 43 – Relatório de listagem de usuário por setor

Na Figura 44 é apresentado o relatório gráfico de número de cartões por coluna.

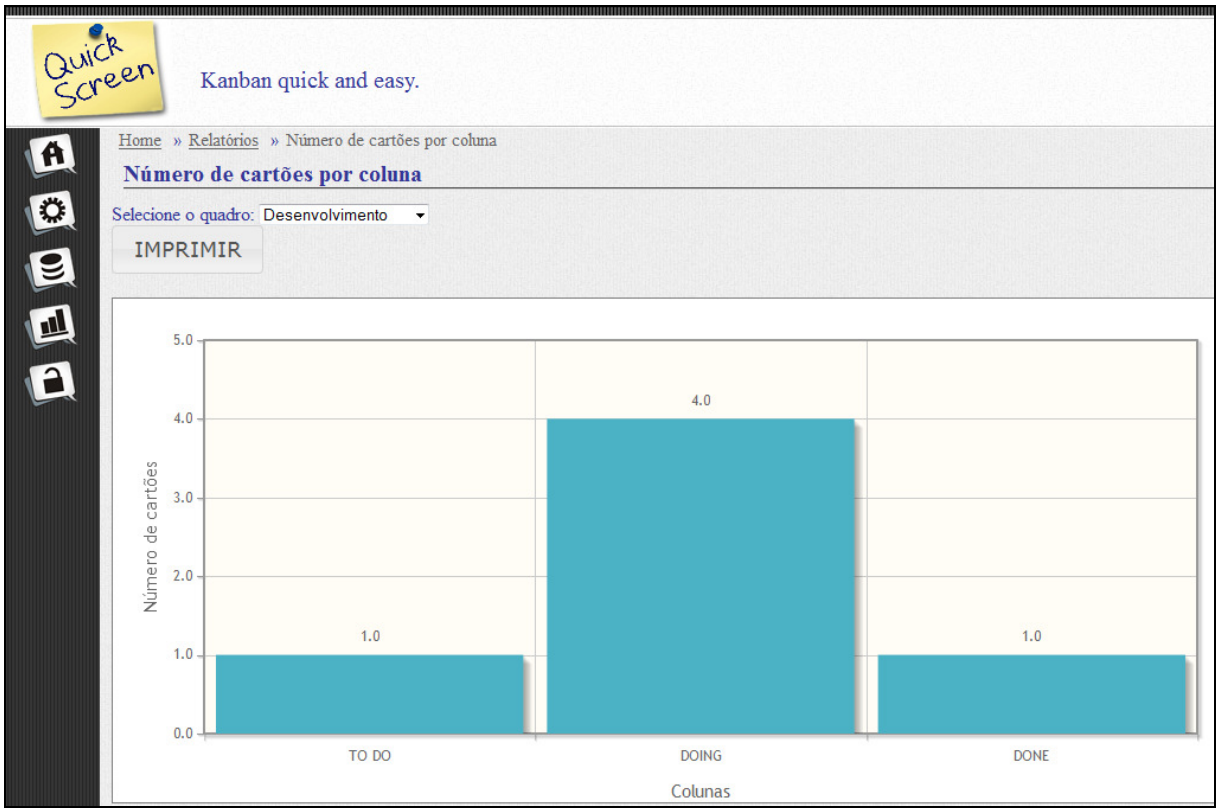


Figura 44 – Relatório gráfico de número de cartões por coluna

3.4 RESULTADOS E DISCUSSÃO

Com o desenvolvimento deste trabalho foi possível criar uma aplicação *web* que proporciona uma visão geral do trabalho em progresso no projeto, facilitando assim o acompanhamento e identificação de possíveis problemas nas etapas do processo.

Em seu trabalho de conclusão de curso Wagner (2003), desenvolveu um sistema de controle da produção que faz uso do *Kanban* para facilitar a visualização das máquinas que estão sendo utilizadas e qual pedido esta sendo produzido. O sistema é limitado ao número máximo de sete máquinas por quadro e pode ser utilizado apenas para produção industrial.

O LeanKit Kanban, desenvolvido pela empresa Bandit Software oferece suporte as principais funcionalidade do *Kanban* e conta diversos relatórios gráficos de fácil entendimento (BANDIT SOFTWARE, 2010).

O Pronto é um sistema desenvolvido em Java por Gomes e Faias Junior (2009), para o cadastro, gerenciamento e acompanhamento de solicitações com objetivo de atingir a satisfação dos clientes, demonstrando maior transparência do que será entregue nas versões do

produto. Possui o quadro de *Kanban* para visualização da atividade, mais todas as atividades devem passar pelos controles do *Scrum* antes de chegar ao quadro, tornando dependente do uso desta metodologia de desenvolvimento.

O Quadro 3 apresenta uma comparação do sistema desenvolvido com os trabalhos correlatos, baseado na fundamentação teórica e nas principais necessidades do desenvolvimento de software.

Funcionalidades/ características	**Aplicação criada**	**WAGNER (2003)**	**LeanKit Kanban**	**Pronto**
plataforma *web*	sim	não	sim	Sim
relatórios estatísticos	sim	sim	sim	Sim
relatório de listagem cartões / atividades pendentes	sim	sim	não	Não
flexibilidade na escolha do processo	sim	não	sim	Não
controle de histórico de movimentação dos cartões	sim	não	sim	Sim
Tecnologias utilizadas	PHP, MySQL, jQuery, Eclipse IDE	Pascal, MySQL, Delphi	C# e Python, Oracle	Java, MySQL, jQuery, PostgreSQL, Eclipse IDE

Quadro 3 – Relação entre trabalhos correlatos

4 CONCLUSÕES

Neste trabalho se propôs o desenvolvimento de uma aplicação *web* para o acompanhamento de projetos utilizando o método *Kanban*. Os principais objetivos do *Kanban* são evitar trabalho dobrado ou desnecessário e identificar problemas e falhas no processo durante a iteração, isso é feito através da simples e eficiente visualização do status do projeto e de cada atividade em quadro, o quadro. O *Kanban* não define um processo ou metodologia de desenvolvimento de sistema em si, na verdade ele se adapta ao utilizado, sendo assim muito adaptativo, ele prescreve apenas três regras:

a) visualize o fluxo de trabalho;
b) limite o trabalho em progresso (WIP);
c) acompanhe a execução da atividade.

Estas três regras, ao longo de um processo de experimentações, que deve ser executado todo o tempo através de *feedbacks* e avaliações constantes do próprio sistema de desenvolvimento, podem ser aliadas a novas regras construídas com o time, transformando o *Kanban* em um processo totalmente voltado para suas necessidades.

Para se conseguir atingir a dinâmica do método *Kanban* a aplicação desenvolvida neste trabalho utiliza a plataforma *web* a fim de disponibilizar o fácil acesso as informações a todos os envolvidos.

A aplicação desenvolvida tem funcionalidades diferenciadas conforme o perfil do usuário, permitindo restrições de acesso a determinadas funcionalidades. A aplicação conseguiu atingir o objetivo do trabalho. A implementação de uma aplicação que possibilite identificar visualmente as atividades que estão sendo realizadas pela equipe assim como em qual etapa se encontra, evidenciando os problemas durante o processo.

Através do desenvolvimento de rotinas de cadastro de usuários, prioridades, tipos de cartões e cartões, obtêm-se um maior controle sobre as atividades desenvolvidas permitindo gerenciá-las e possibilitando a extração de dados através de relatórios.

Como maior dificuldade deste trabalho, destaca-se a questão do desenvolvimento do quadro totalmente *web*, tornando a experiência do usuário o mais próximo da realidade do quadro físico foram utilizadas técnicas de *drag and drop* (arrastar e largar), tornando a aplicação intuitiva e de fácil utilização.

Pode-se dizer que o trabalho permitiu a ampliação dos conhecimentos pessoais sobre o método *Kanban*, e sobre cultura de produção desenvolvida no Sistema Toyota de Produção,

além de novas ferramentas e aplicativos que auxiliam no desenvolvimento de sistemas. Em particular ao CSS, o qual tinha pouca familiaridade.

Por fim, pode-se dizer que a utilização da aplicação desenvolvida junto a alguma metodologia de desenvolvimento ágil, mostrou-se uma grande aliada, tornando o processo visual e facilitando a identificação de problemas e falhas no processo.

4.1 EXTENSÕES

Como sugestão de implementação de funcionalidade para trabalhos futuros pode-se destacar:

a) o desenvolvimento de um *webserver* para facilitar a integração da aplicação desenvolvida com softwares de controle de chamados e tarefa;
b) desenvolver plugins para dar suporte completo as práticas de metodologias ágeis;
c) desenvolver um módulo para acesso através de dispositivos portáteis;
d) o desenvolvimento de mais relatório estatístico para auxiliar na obtenção de informações do processo.

REFERÊNCIAS BIBLIOGRÁFICAS

BANDIT SOFTWARE. **Lean Process Management.** New York, 2010. Disponível em: http://www.leankitkanban.com. Acesso em: 20/05/2010.

BETTER EXPLAINED. **Intermediate Rails: Understanding Models, Views and Controllers.** [S.1.], 2007. Disponível em: http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/. Acesso em: 09 abr. 2011.

CASTRO, André Dourado. **Um dia na terra do Kanban.** Curitiba, 2009. Disponível em: http://blog.adsystems.com.br/2009/07/01/um-dia-na-terra-do-kanban/. Acesso em: 21/06/2010.

CAKEPHP BRASIL. **CakePHP framework.** [S.l.], 2007. Disponível em: http://cakephp.com.br/modules/news/index.php?storytopic=2. Acesso em: 22 set. 2010.

FOWLER, Martin. **Using an Agile Software Process with Offshore Development.** [S.l.]**,** 2006. Disponível em: http://www.martinfowler.com/articles/agileOffshore.html. Acesso em: 11 maio 2010.

FURTADO, Vasco. **Tecnologia e gestão da informação na segurança pública**. Rio de Janeiro: Garamond, 2002.

GOMES, André Faria; FAIAS JUNIOR, Luiz dos Santos. **Pronto! - Software para gestão de projetos ágeis**. 2009. 66f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Faculdade de Informática e Administração Paulista, São Paulo, 2009.

IMAI, Masaaki. Kaizen: **A estratégia para o sucesso competitivo.** 5 ed. São Paulo: Iman, 1994.

INFOQ. **Tracking change and innovation in the enterprise software development community.** [S.1.], 2010. Disponível em: http://www.infoq.com. Acesso em: 12 maio 2011.

LINKER, Jeffrey K. e MEIER, David. **O modelo toyota - um guia prático para a implementação dos 4ps da Toyota.** Porto Alegre: ARTMED, 2006.

MAGEE, D. **O segredo da Toyota: lições de liderança da maior fabricante de automóveis do mundo.** São Paulo: Campus, 2008.

MARTINS, José Carlos C. **Técnicas para gerenciamento de projetos de software.** Rio de Janeiro: Brasport, 2007.

MCADAM, Rodney, FULTON, Francs. The impacto of the ISSO 9000:2000 quality standars in smal software firms. **Managing Service Quality.** V. 12, n.5, p. 336-345, 2002.

MANOR, Gilad. **JQuery, um Framework JavaScript.** [S.l.]**,** 2010. Disponível em: http://www.infoq.com/br/news/2010/05/jquery_framework. Acesso em: 10 maio 2010.

MONDEN, Yasuhiro. **Sistema Toyota de produção.** São Paulo: IMAM, 1984

MOURA, Reinaldo A. **Kanban – A simplicidade do controle de produção.** São Paulo: IMAM, 1989.

ORTH, Afonso I.; PRIKLADNOCKI, Rafael. **Planejamento e gerência de projetos.** Porto Alegre: EDIPUCRS, 2009.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software: Fundamentos, Métodos e Padrões.** Rio de Janeiro: LTC, 2003.

PACE, João H. **O Kanban na prática.** Rio de Janeiro: Qualitymark, 2003.

PROJECT MANAGEMENT INSTITUTE. **A Guide to the Project Management Body of Knowledge - PMBOK Guide.** Newtown Square, Pennsylvania, USA: Paperback, 2001.

REZENDE, Denis A. **Engenharia de software e sistemas de informação.** 3. ed. Rio de Janeiro: Brasport, 2005

SILVA, Maurício Samy. **jQuey – A Biblioteca do Programador JavaScript**. 2. ed. São Paulo: Novatec, 2010.

SPYER, Juliano. **Para Entender a Internet - Noções, práticas e desafios da comunicação em rede.** [S.l.], 2009. Disponível em: http://www.openinnovatio.org/wp-content/Para%20entender%20a%20Internet.pdf. Acesso em: 15 mar. 2011.

TOMAYKO, James E.; HALLMAN, Harvey K. **Software Project Management.** Pennsylvania: Software Engineering Institute, 1999.

VARGAS, Ricardo. **Gerenciamento de projetos**. 6. ed. Rio de Janeiro: Brasport, 2005.

WAGNER, Alzir. **Sistema de informação da produção utilizando o método Kanban**. 2008. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Expansão dos casos de uso

Os quadros de 4 a 8 apresentam a descrição dos principais casos de uso do sistema.

Cadastra quadro	
Ator	Administrador
Descrição	Este caso de uso tem como objetivo cadastrar, editar e excluir quadros
Pré-condições	Efetuar *login* no sistema
Pós-condições	Quadro foi cadastrado, editado ou excluído
Cenário Principal	1. O Administrador acessa a tela inicial do sistema 2. O Sistema apresenta listagem de quadros cadastrados 3. O Administrador seleciona a opção novo quadro 4. O Sistema direciona o usuário para a tela de cadastro de quadros 5. O Administrador preenche os dados do formulário 6. O Administrador seleciona a opção salvar 7. O Sistema grava quadro
Cenário de Exceção	No passo 3 o Administrador seleciona a opção editar registro 4. O Sistema direciona o Administrador para a tela de edição de quadro 5. O Administrador realiza as alterações necessárias 6. O Administrador seleciona a opção salvar 7. O Sistema grava alterações
Cenário de Exceção	No passo 3 o Administrador seleciona a opção excluir registro 4. O Sistema exclui o quadro

Quadro 4 **–** Detalhamento do caso de uso Cadastra quadro

Cadastra coluna	
Ator	Administrador
Descrição	Este caso de uso tem como objetivo cadastrar, editar e excluir colunas
Pré-condições	1. Efetuar *login* no sistema 2. Deve existir pelo menos um quadro cadastro
Pós-condições	Coluna foi cadastrada, editada ou excluída
Cenário principal	1. O Administrador acessa a tela inicial da aplicação 2. O Sistema apresenta os quadros cadastrados 3. O Administrador seleciona um quadro 4. O Sistema direciona o usuário para a tela de edição de quadro e lista as colunas do quadro logo abaixo 5. O Administrador seleciona a opção nova coluna 6. O Sistema direciona Administrador para tela de cadastro de coluna 7. O Administrador preenche os dados do formulário

	8. O Administrador seleciona opção salvar 9. O Sistema grava coluna
Cenário de Exceção	No passo 5 o Administrador seleciona a opção editar registro 6. O Sistema direciona o usuário para a tela de edição de quadro e lista as colunas do quadro logo abaixo 7. O Administrador seleciona uma coluna 8. O Sistema direciona Administrador para tela de edição de coluna 9. O Administrador edita as informações da coluna 10. O Administrador seleciona opção salvar 11. O Sistema grava alterações da coluna
Cenário de Exceção	No passo 5 o Administrador seleciona a opção excluir coluna 6. O Sistema exclui coluna

Quadro 5 – Detalhamento do caso de uso Cadastra coluna

Movimenta cartão	
Ator	Usuário
Descrição	Este caso de uso tem como objetivo movimentar os cartões sobre o quadro
Pré-condições	1. Efetuar *login* no sistema 2. Deve existir pelo menos um quadro cadastrado 3. Deve existir pelo menos uma coluna no quadro 4. Deve existir pelo menos um cartão no quadro
Pós-condições	O cartão foi assumido e movimentado sobre o quadro
Cenário Principal	1. O Usuário acessa a tela inicial do sistema 2. O Sistema apresenta listagem de quadros cadastrados 3. O Usuário seleciona um quadro 4. O Sistema direciona o Usuário para o quadro 5. O Usuário seleciona um cartão, assume sua responsabilidade e movimenta o cartão para a coluna onde será realizada a atividade 6. O Sistema registra a movimentação no histórico de movimentação do cartão 7. O Usuário movimenta o cartão para a próxima coluna do quadro e remove sua responsabilidade sobre o cartão 8. O Sistema registra a movimentação no histórico de movimentação do cartão
Cenário de Exceção	No passo 5 o Sistema consulta o cartão e verifica o mesmo está bloqueado por outro usuário 6. O Sistema apresenta uma mensagem informando que o cartão foi bloqueado por outro usuário 7. O sistema retornar o cartão para a posição original
Cenário de Exceção	No passo 5 o Sistema verifica que o número de cartões na coluna mais o cartão que esta sendo movimentado pelo Usuário ultrapassa o WIP máximo da coluna 6. O Sistema apresenta uma mensagem informando o Usuário que o mesmo está ultrapassando o limite de WIP da coluna apresenta um campo para registro do motiva dessa ação

	7. O Usuário registra o motivo da ação 8. O Sistema registra a movimentação no histórico de movimentação do cartão 8. O Usuário movimenta o cartão para a próxima coluna do quadro e remove sua responsabilidade sobre o cartão 9. O Sistema registra a movimentação no histórico de movimentação do cartão

Quadro 6 – Detalhamento do caso de uso Movimenta cartão

Registra providência	
Ator	Usuário
Descrição	Este caso de uso tem como objetivo registrar uma providência sobre as atividades realizadas no cartão
Pré-condições	1. Efetuar *login* no sistema 2. Deve existir pelo menos um quadro cadastrado 3. Deve existir pelo menos uma coluna no quadro 4. Deve existir pelo menos um cartão no quadro 5. Assumir a responsabilidade de uma cartão
Pós-condições	Usuário registrou uma providência sobre as atividades realizadas no cartão
Cenário Principal	1. O Usuário seleciona cartão 2. O Usuário seleciona a opção providências do cartão 3. O Usuário registrar uma providência 4. O Sistema grava a providência 5. O Sistema grava a ação no histórico de movimentação do cartão

Quadro 7 – Detalhamento do caso de uso Registra providência

Visualiza histórico de movimentação do cartão	
Ator	Usuário
Descrição	Este caso de uso tem como objetivo visualizar o histórico de todas as movimentações e ações do cartão no quadro
Pré-condições	1. Efetuar *login* no sistema 2. Deve existir pelo menos um quadro cadastrado 3. Deve existir pelo menos uma coluna no quadro 4. Deve existir pelo menos um cartão no quadro
Pós-condições	Usuário consultou o histórico das movimentações e ações do cartão no quadro
Cenário Principal	1. O Usuário seleciona cartão 2. O Usuário seleciona a opção histórico do cartão 3. O Sistema lista todas as movimentações e ações do cartão sobre o quadro assim com a data, hora e usuário que executou as mesmas.

Quadro 8 – Detalhamento do caso de uso Visualiza histórico de movimentação do cartão

APÊNDICE B – Dicionário de dados

O dicionário de dados descreve em mais detalhes as entidades da modelagem do banco de dados utilizada no trabalho desenvolvido. Os tipos de dados de cada campo são descritos a seguir:

a) *varchar*: tipo de campo para armazenamento de strings de caracteres e seu tamanho é definido em *bytes* com largura variável, os valores entre parênteses definem o comprimento máximo em *bytes* de caracteres;
b) *int*: tipo de campo para armazenamento de números inteiros;
c) *smallint*: tipo de campo para armazenamento de números inteiros, fornece 2 bytes de armazenamento numérico;
d) *datetime*: tipo de campo para armazenamento de datas e horas;
e) *text*: tipo de campo para armazenamento de grandes strings ou binários.

O dicionário de dados apresenta o nome da tabela, o nome do campo, o tipo do campo, a descrição do campo e a observação para o caso do campo fazer parte de chave primária e/ou chave estrangeira.

O Quadro 9 contém o dicionário de dados da tabela “setores”.

Tabela: setores			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro do setor.	Chave primária.
descricao	*varchar(255)*	Descrição do setor.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se o setor está ativo ou não.	
created	*datetime*	Data de cadastro do setor.	
modified	*datetime*	Data da última modificação do setor.	

Quadro 9 – Dicionário de dados da tabela “setores”

O Quadro 10 contém o dicionário de dados da tabela “usuarios”.

Tabela: usuarios			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro do usuário.	Chave primária.
setor_id	*int*	Código do setor que o usuário pertence.	Chave estrangeira.
nome	*varchar(255)*	Nome do usuário.	
email	*varchar(255)*	Email do usuário.	
username	*varchar(100)*	*Login* do usuário	
password_2	*varchar(40)*	*Hash* da senha do usuário	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se o usuário está ativo ou não.	
regra	*smallint*	Inteiro de valor 1 ou 0 informando se o usuário é um Administrador ou Usuário normal.	
avatar	*varchar(100)*	Caminha da imagem do usuário.	
created	*datetime*	Data de cadastro do usuário.	
modified	*datetime*	Data da última modificação do usuário.	

Quadro 10 – Dicionário de dados da tabela “usuarios”

O Quadro 11 contém o dicionário de dados da tabela “quadros”.

Tabela: quadros			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro do quadro.	Chave primária.
titulo	*varchar(255)*	Titulo do quadro.	
descricao	*text*	Descrição do quadro.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se o quadro está ativo ou não.	
created	*datetime*	Data de cadastro do quadro.	
modified	*datetime*	Data da última modificação do quadro.	

Quadro 11 – Dicionário de dados da tabela “quadros”

O Quadro 12 contém o dicionário de dados da tabela “colunas”.

Tabela: colunas			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro da coluna.	Chave primária.
quadro_id	*int*	Código do quadro.	Chave estrangeira.
titulo	*varchar(255)*	Titulo da coluna.	
wip	*smallint*	Número máximo de tarefas sendo executadas na coluna.	
ordem	*smallint*	Ordem de exibição da coluna no quadro.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se a coluna está ativo ou não.	
created	*datetime*	Data de cadastro da coluna.	
modified	*datetime*	Data da última modificação da coluna.	

Quadro 12 – Dicionário de dados da tabela “colunas”

O Quadro 13 contém o dicionário de dados da tabela “prioridades”.

Tabela: prioridades			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro da coluna.	Chave primária.
descricao	*varchar(255)*	Descrição da prioridade.	
sequencia	*Int*	Sequência numérica que define a criticidade da prioridade, quando menor o número maior sua criticidade.	
icone	*varchar(255)*		
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se a prioridade está ativo ou não.	
created	*datetime*	Data de cadastro da	

		prioridade.	
modified	*datetime*	Data da última modificação da prioridade.	

Quadro 13 – Dicionário de dados da tabela “prioridade”

O Quadro 14 contém o dicionário de dados da tabela “tipocartoes”.

Tabela: tipocartoes			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
Id	*int*	Código de registro do tipo de cartão.	Chave primária.
Descricao	*varchar(255)*	Descrição do tipo de cartão.	
color	*varchar(100)*	Cor do tipo de cartão, esta cor será utilizada pelo cartão no quadro.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se o tipo de cartão está ativo ou não.	
created	*datetime*	Data de cadastro do tipo de cartão.	
modified	*datetime*	Data da última modificação do tipo de cartão.	

Quadro 14 – Dicionário de dados da tabela “tipocartoes”

O Quadro 15 contém o dicionário de dados da tabela “permissoes”.

Tabela: permissoes			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
Id	*int*	Código de registro do tipo de cartão.	Chave primária.
quadro_id	*int*	Código do quadro.	Chave estrangeira.
usuario_id	*int*	Código do usuário.	Chave estrangeira.
usuariocriacao_id	*int*	Código do usuário que definiu a permissão de acesso.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se a permissão está ativo ou não.	
created	*datetime*	Data de cadastro da	

		permissão.	
modified	*datetime*	Data da última modificação da permissão.	

Quadro 15 – Dicionário de dados da tabela “permissoes”

O Quadro 16 contém o dicionário de dados da tabela “cartoes”.

Tabela: cartões			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro do cartão.	Chave primária.
coluna_id	*int*	Código da coluna.	Chave estrangeira.
quadro_id	*int*	Código do quadro.	Chave estrangeira.
prioridade_id	*int*	Código da prioridade.	Chave estrangeira.
tipocartao_id	*int*	Código do tipo de cartão.	Chave estrangeira.
usuário_id	*int*	Código do usuário responsável pelo cartão.	Chave estrangeira.
usuariocriacao_id	*int*	Código do usuário que cadastrou o cartão.	Chave estrangeira.
titulo	*varchar(255)*	Titulo do cartão.	
descrição	*text*	Descrição das atividades do cartão.	
vencimento	*datetime*	Data de vencimento do cartão	
bloqueado	*smallint*	Inteiro de valor 1 ou 0 informando se o cartão esta bloqueado para movimentação no quadro.	
referencia	*varchar(100)*	Campo aberto preparado para futuras integrações.	
tamanho	*int*	Campo inteiro para mensurar o tamanho do cartão dentro do quadro.	
ativo	*smallint*	Inteiro de valor 1 ou 0 informando se o cartão está ativo ou não.	
created	*datetime*	Data de cadastro do cartão.	

modified	*datetime*	Data da última modificação do cartão.	

Quadro 16 – Dicionário de dados da tabela “cartoes”

O Quadro 17 contém o dicionário de dados da tabela “providencias”.

Tabela: providencias			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro da providência.	Chave primária.
usuario_id	*int*	Código do usuário que registrou a providência.	Chave estrangeira.
cartao_id	*int*	Código do cartão da providência.	Chave estrangeira.
descricao	*text*	Descrição da providência.	
created	*datetime*	Data de cadastro do cartão.	
modified	*datetime*	Data da última modificação do cartão.	

Quadro 17 – Dicionário de dados da tabela “providencias”

O Quadro 18 contém o dicionário de dados da tabela “historicos”.

Tabela: historicos			
Nome da coluna	**Tipo**	**Descrição**	**Observação**
id	*int*	Código de registro de histórico.	Chave primária.
usuario_id	*int*	Código do usuário que executou a ação.	Chave estrangeira.
cartao_id	*int*	Código do cartão da ação.	Chave estrangeira.
acao	*varchar(255)*	Tipo de ação executada.	

observacao	*text*	Observações e informações extras da ação.	
created	*datetime*	Data de cadastro do histórico.	
modified	*datetime*	Data da última modificação do histórico.	

Quadro 18 – Dicionário de dados da tabela “historicos”