INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE SÃO PAULO

Campus São João da Boa Vista
Trabalho Final de Curso
4º ano – Curso Técnico em Informática
Prof. Breno Lisi Romano

TÉCNICAS E FERRAMENTAS DE DESENVOLVIMENTO ÁGIL DE *SOFTWARE* APLICADO AO PROJETO INDRA.

Aluno: Igor Henrique Torati Ruy
Prontuário: 1320262

São João da Boa Vista – SP
2016

Resumo

Existem basicamente dois tipos de metodologias principais, as metodologias tradicionais, que pregam que um projeto deva ter muita documentação e não pode fugir do escopo, e as metodologias ágeis, que pregam documentação mínima e adaptação do produto conforme o desejo do cliente, o que gera uma maior qualidade do produto final.

No ambiente do desenvolvimento ágil, existem algumas ferramentas para auxiliar na organização e no gerenciamento de projetos, tais como *kanban* e *Redmine* e técnicas como *Scrum* e *XP*.

Para execução do projeto do 4º Ano do curso Técnico Integrado ao Ensino Médio foram utilizadas várias práticas relacionadas a metodologias ágeis de desenvolvimento de software tais: como documentação mínima o que possibilita que futuras alterações em requisitos e casos de uso sejam mais fáceis e tenham uma agilidade maior para ocorrerem, entrega frequente de iterações, *Kanban*, *Redmine* e ideias vindas do *Scrum* como reuniões em todas as aulas de PDS (*Daily Scrum*), apresentação do que foi desenvolvido ao final de cada iteração entre outras técnicas.

Sumário

Sumário

1 Introdução

De início, fora proposto aos alunos do 4º ano que trabalhassem na criação de um site para alguma empresa, porém, devido desistência da empresa em questão, fora sugerido pelo professor Breno Lisi Romano, professor da matéria de PDS e responsável pelo projeto, que fizéssemos um portal online para consultas e comparações climáticas onde o usuário pode visualizar informações como: umidade, temperatura, nível da chuva e ter acesso a ferramentas como previsões de épocas de risco de enchentes e locais atingidos pelas mesmas. O portal irá receber as informações necessárias para cálculo dessas épocas e locais atingidos por meio das PCDs (Plataforma de Coleta de Dados), existentes nas cidades abrangidas pelo projeto, no caso de São João da Boa Vista inicialmente, porem atualmente já existem outros municípios interessados no projeto como é o caso da prefeitura de Poços de Caldas, que no início do ano sofreu grandes danos com uma enchente. Essa ideia surgiu de um trabalho de mestrado realizado pelo professor Breno.

O nome do projeto fora definido como Indra, em homenagem ao deus Hinduísta das tempestades, devido a ideia do portal de alertas de enchentes e época de risco desse tipo de desastre. Para realização do projeto, os alunos foram divididos em módulos e cada modulo possui as respectivas responsabilidades, como mostra a Figura 1:

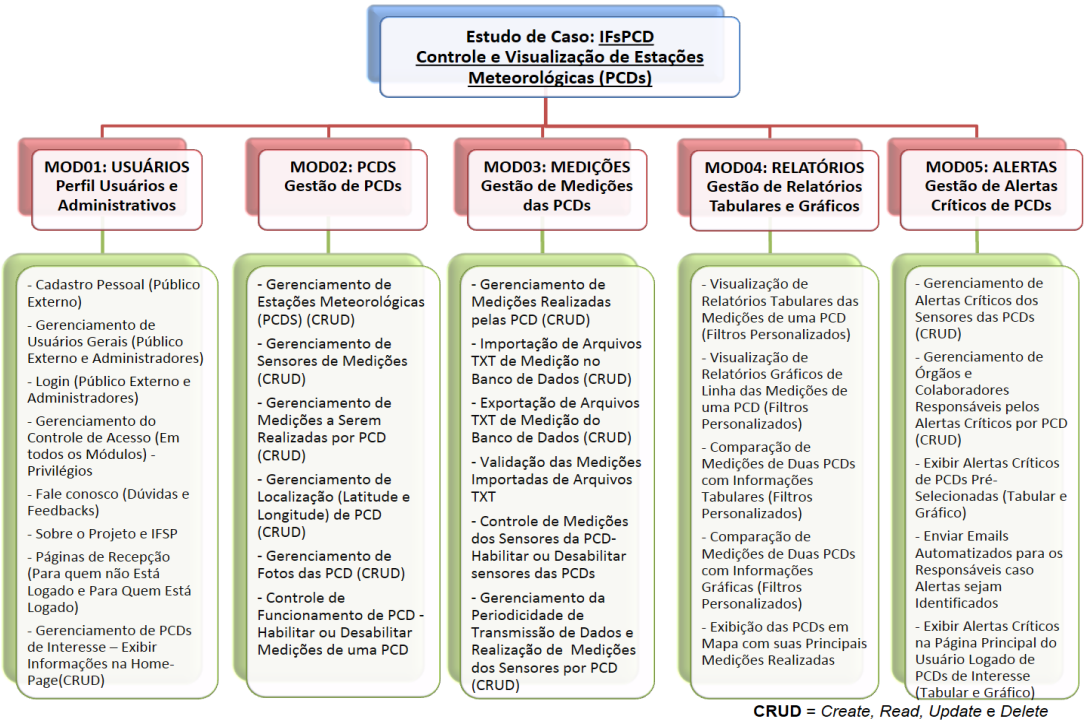

Figura 1, Distribuição dos módulos do projeto Indra. Disponível aqui.

Além da divisão por módulos os alunos também foram divididos em funções, são elas:

1. Analista: Que possui a função inicial de trabalhar mais com documentos e definições das especificações do projeto. Ao final da criação das especificações começaram a trabalhar com o banco e com o código do portal.
2. *DBA*: Inicialmente trabalhou com os analistas para definir as especificações e ajudou na consolidação dos documentos. Após a definição do documento começaram a trabalhar com os projetos de banco e sua consolidação, tal como fará as alterações necessárias no banco e trabalhara com o código do portal.
3. Desenvolvedores: Responsáveis inicialmente por definir o *template* das apresentações e do site, tal como definir as linguagens e tecnologias a serem utilizadas no desenvolvimento do portal e com a programação *back* e *front-end* do início ao fim do ano letivo.

Por meio desse trabalho pretendo demonstrar quais e como estão sendo aplicadas as práticas de desenvolvimento ágil de *Software*, tal como o surgimento de tal metodologia, o que irá englobar parte da história do desenvolvimento de *Software* e algumas metodologias tradicionais.

2 Desenvolvimento

2.1 Referencial teórico:

2.1.1 Desenvolvimento de *Software* com Processos Tradicionais:

"Um programa de computador é composto por uma sequência de instruções, que é interpretada e executada por um processador ou por uma máquina virtual. Em um programa correto e funcional, essa sequência segue padrões específicos que resultam em um comportamento desejado ". [1]

Durante a década de 70 a produção de um *Software* era feita de maneira desorganizada, desestruturada e sem nenhum tipo de planejamento, documentação insuficiente para entender o que o cliente precisava e queria, não realização de muitos testes antes da conclusão do projeto, prazo de entrega e orçamentação feitos de maneira ineficiente e errônea, esse período ficou conhecido como a "Crise do *Software*".

Esses são alguns dos motivos que tornaram necessária a criação de um processo estruturado para a criação de *Software*s, a partir disso diversas metodologias de desenvolvimento. Essas metodologias dividem, cada uma a sua maneira, a forma e o processo de criação de um *Software*. Apesar de cada uma definir de maneiras diferentes as etapas e processos alguns problemas da "Crise do *Software*" ainda permanecem até os dias de hoje, como estimativa de custos e prazos estimados de maneira errada.

2.1.2 Desenvolvimento Sistematizado de *Software*:

Uma das primeiras padronizações que surgiram para o Desenvolvimento de *Software* foi o Desenvolvimento Sistematizado de *Software*, ela utilizava-se de conceitos aplicado à engenharia a fim de tornar o processo de desenvolvimento algo econômico e eficiente que gerasse um produto final de qualidade, o que, futuramente seria o enfoque de outras metodologias mais modernas e mais eficazes para tal.

Fritz Bauer define a Engenharia de *Software* como: "A criação e utilização de sólidos princípios da Engenharia a fim de obter *Software* de maneira econômica, que seja confiável para trabalhar eficientemente em máquinas reais" ou seja: Engenharia

é a análise, o projeto, a construção (desenvolvimento), a verificação e a gestão de elementos técnicos.” [2]

A criação de documentos, realização de reuniões, levantamento de recursos, prazos, restrições entre outros aspectos, possibilita a criação de *Software* de maneira que o mesmo possua uma maior qualidade, a fim de gerar o melhor produto possível.

Para aumentar a eficiência desses processos, surgiram vários métodos de Engenharia de *Software* que visa realizar cada etapa do processo de maneira eficaz, por exemplo levantamento e analise correta de requisitos, boas práticas e maneiras de se programar com eficiência, realização profunda de testes para reduzir os problemas do produto final e também definir como deve ocorrer a manutenção do produto.

2.1.3 Metodologias mais conhecidas e suas Fases de Desenvolvimento:

2.1.3.1 Modelo Cascata:

A metodologia mais conhecida para o desenvolvimento de *Software* é a Metodologia em Cascata (*Cascade* em inglês, também conhecida como Modelo Sequencial Linear) seguida da Prototipação e da Metodologia Espiral.

A metodologia em Cascata é a mais conhecida de todas e é muito utilizada nos tempos de hoje, surgiu no ano de 1970, com a publicação de um artigo por W.W. Royce, ironicamente esse artigo foi publicado como uma forma de alerta, uma vez que Royce defendia uma abordagem mais iterativa, ele publicou esse artigo expressando que a tal metodologia “Cascata” (não nomeada por ele) era, segundo suas palavras, “*um risco e um convite para falhas*”.

O modelo em Cascata é basicamente dividido nas seguintes etapas:

1. Definição de requisitos: análise do sistema onde o *Software* será desenvolvido, os requisitos do *Software* são levantados e definidos;
2. Projeto: representação dos requisitos, é subdividido em quatro atributos:

i. Estrutura de Dados, que é como os dados serão tratados.
ii. Arquitetura de *Software*, que define a base estrutural de como o sistema será desenvolvido.
iii. Caracterizações das Interfaces, representa o meio entre o usuário e o sistema, bem como entre os módulos do sistema

iv. Detalhes Procedimentais. [3]

3. Codificação/implementação: resumisse na programação do sistema, ou seja, transformar os requisitos de uma maneira que a máquina entenda;
4. Testes e implantação: identificar se as funcionalidades desenvolvidas estão funcionando de maneira correta e instalação do sistema (tal como disponibilização de um suporte técnico a fim de ensinar o cliente a utilizar o *Software*).
5. Manutenção: é basicamente a correção de erros encontrados após a entrega para o cliente, implementando melhorias ao produto – implicando em um novo ciclo de desenvolvimento.

Prosseguir de uma fase para a outra é um processo simples e linear, o único requisito é terminar a fase anterior, por exemplo, um projeto começa na fase de requisitos, após terminada essa fase seguia-se para a próxima, a fase de projeto. Nessa metodologia procura-se entender o todo de cada etapa, por exemplo, no levantamento de requisitos visava-se identificar e compreender todos os requisitos que o sistema deveria ter, isso em uma única etapa, uma etapa vital para o processo de desenvolvimento. Nas próximas etapas visava-se a criação de soluções que atendessem todos os requisitos e restrições, depois iniciava-se a implementação, em seguida os testes e por fim, após os testes, se implantava o sistema e se verificava se o projeto atendia a todas as necessidades juntamente com a manutenção do mesmo, a Figura 02 mostra o esquema de uma metodologia Cascata.

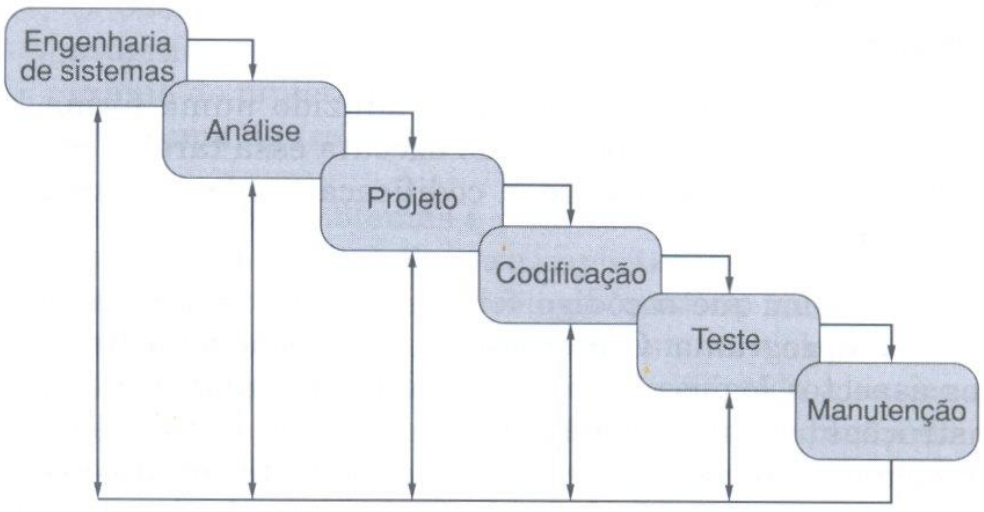

Figura 2 Esquema de Metodologia Cascata. [4]

A abordagem adotada pela metodologia cascata acaba trazendo alguns problemas. Dentre estes problemas merece destaque o fato de que os projetos reais dificilmente seguem o fluxo sequencial, o cliente quase sempre não consegue exprimir todas as suas necessidades além de ser exigida dele muita paciência visto que o *Software* só estará pronto para uso num ponto tardio do cronograma. E o maior dos problemas é que se ocorrer um erro em qualquer uma das etapas o resultado pode ser desastroso e frequentemente caro.
Além desses problemas, algumas vezes o cliente não sabe ou não expressa todos os requisitos de forma explicita, outras a codificação não atende totalmente um requisito, ou não está adaptado ao sistema operacional utilizado pelo cliente ou até mesmo dificuldade com a compreensão da interface (má interação homem-máquina).

2.1.3.2 Prototipação:

A Prototipação é uma metodologia surgida posteriormente à Cascata. Ela possibilita a equipe de desenvolvimento a criar uma aplicação protótipo que pode assumir três formas distintas. A primeira delas é um protótipo em papel ou mesmo no computador que retrate a interação homem-máquina. A segunda opção é implementar uma funcionalidade que já está no escopo do *Software* a ser desenvolvido. Por fim existe a possibilidade de utilizar-se de um *Software* já pronto que tenha parte ou todas as funcionalidades desejadas. Esta forma é mais comumente adotada em *Software*s que apesar de prontos ou parcialmente prontos possuem características que precisam ser incrementadas ou melhoradas em um novo esforço de desenvolvimento.

Geralmente o protótipo serve apenas como um mecanismo para identificar requisitos de *Software*. Isto ocorre por que na maior parte dos casos o primeiro sistema construído não é completamente usável. Normalmente ele possui uma série de problemas que só serão corrigidos em uma versão “reprojetada” na qual as deficiências sejam corrigidas. [4]

Mesmo sendo mais eficiente, essa metodologia também apresenta alguns problemas, um deles é que o cliente pode não entender corretamente o significado de protótipo, e assumir que aquele é uma versão avançada, quase pronta, do *Software*, e muitas vezes exerce pressão para que haja uma versão final

rapidamente. O problema é que essa pressão em excesso sobre a equipe pode pesar na qualidade do *Software* final. A Figura 3 mostra como é o esquema de prototipagem.

Figura 3 Ciclo da Prototipagem. Disponível em: http://jkolb.com.br/wp-content/uploads/2013/12/prototipagem.png

Apesar desses problemas, a prototipação ainda é uma eficiente metodologia de desenvolvimento de *Software*. A fim de se obter sucesso no projeto, tanto cliente quanto desenvolvedor devem chegar a um consenso de que o protótipo servirá apenas para ajudar na definição dos requisitos. [5]

2.1.3.2 Espiral:

Esse modelo surgiu unindo-se as melhores práticas da prototipagem e do modelo cascata, além de unir as práticas dos outros dois modelos trouxe também algumas inovações, como por exemplo a análise de riscos e foi a primeira metodologia ao adotar o conceito de Iterações, que é basicamente o conceito de dividir o produto final em pequenas partes como ferramentas do produto, sistemas menores dentro do produto etc.

Na primeira iteração, os objetivos, alternativas e restrições são definidos e os riscos são identificados e analisados. O cliente avalia o resultado da iteração e baseado nos apontamentos do mesmo a próxima iteração é iniciada.

O que possibilitou além de ter um prazo de entrega do projeto ter um prazo para entrega de cada Iteração, e o que também facilita a demonstração e disponibilidade de um protótipo, segue o modelo Espiral na Figura 4:

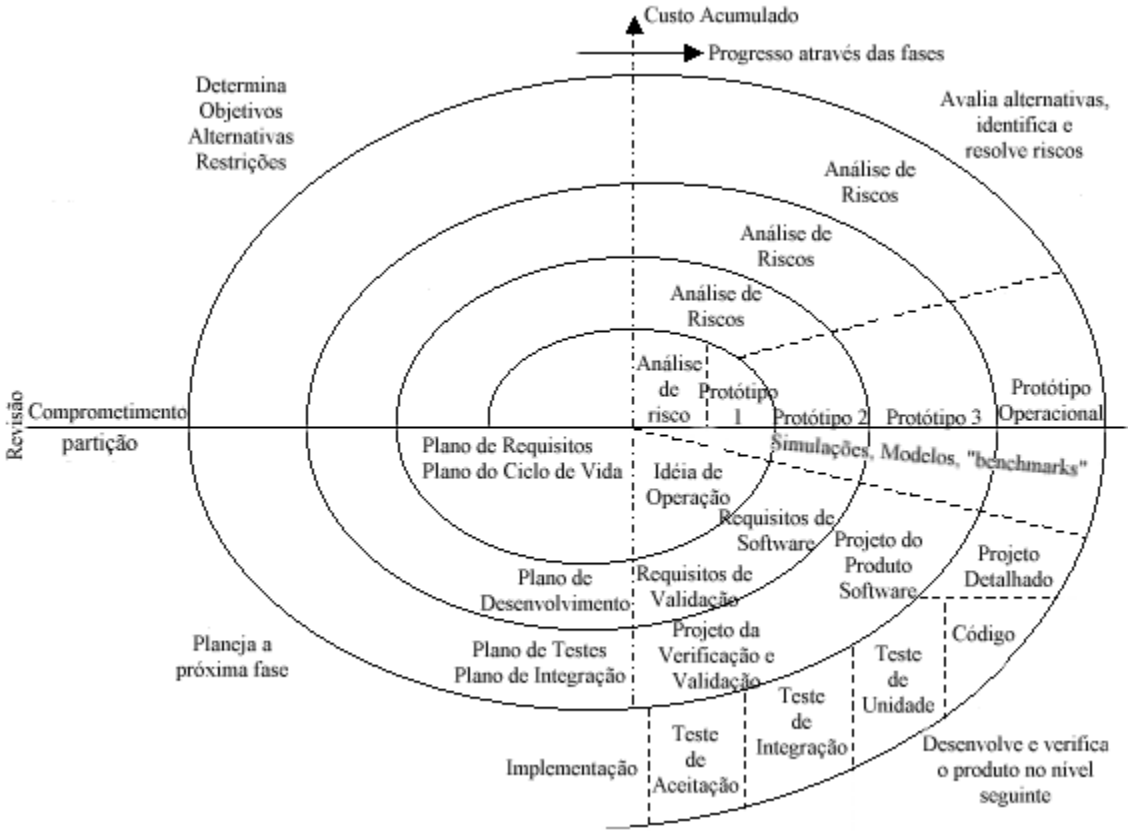

Figura 4 Ciclo do Modelo em Espiral. Disponível em: http://adsbaixarengenhariadesoftware.blogspot.com.br/2013/05/introducao-engenharia-de-software.html

2.1.4 Desenvolvimento Ágil de *Software*:

O desenvolvimento ágil de *Software* é uma técnica que visa entregar um produto com a maior qualidade possível para um cliente, ou seja, no decorrer de um projeto sua qualidade não pode ser afetada por falta de tempo ou de verba. A Figura 5 mostra o triângulo Qualidade x Custo x Benefício:

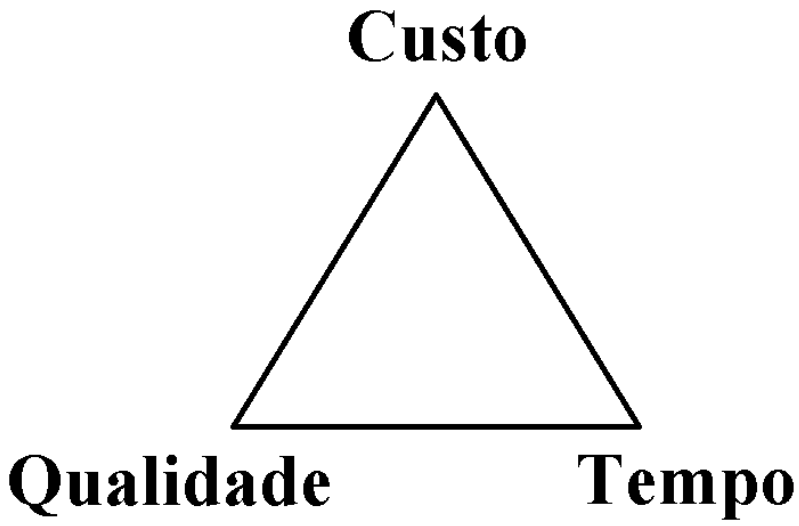

Figura 5 Triângulo Qualidade x Custo x Beneficio

Como apresentado no triangulo temos três pontos que devem ser levados em consideração na produção de um site ou *Software* ou até mesmo produto qualquer. Como dito anteriormente, na “metodologia” ágil a qualidade é sempre a maior possível, logo alterasse o tempo de produção e o custo do projeto.
Outra característica do desenvolvimento ágil de *Software* é sua carga mínima de documentação, isso ocorre para ser mais fácil mudar algo ao decorrer do projeto, caso o cliente deseje essa mudança, isso também está relacionado a qualidade do produto e a satisfação do cliente.

“As definições modernas de desenvolvimento de *Software* ágil evoluíram a partir da metade de 1990 como parte de uma reação contra métodos "pesados", caracterizados por uma pesada regulamentação, regimento e micro gerenciamento usado o modelo em cascata para desenvolvimento. O processo originou-se da visão de que o modelo em cascata era burocrático, lento e contraditório a forma usual com que os engenheiros de *Software* sempre realizaram trabalho com eficiência.
Uma visão que levou ao desenvolvimento de métodos ágeis e iterativos era retorno a prática de desenvolvimento vistas nos primórdios da história do desenvolvimento de *Software*” [6]

Peter Ferdinand Drucker (19 de novembro de 1909, em Viena, Áustria - 11 de novembro de 2005, em Claremont, Califórnia, EUA) foi um escritor, professor e consultor administrativo de origem austríaca, considerado como o pai da administração moderna, sendo o mais reconhecido dos pensadores do fenómeno dos efeitos da globalização na economia em geral e em particular nas organizações

- subentendendo-se a administração moderna como a ciência que trata sobre pessoas nas organizações, como dizia ele próprio.[7]

Drucker explica a necessidade da “metodologia” Ágil pela seguinte afirmação:

“Existem duas categorias de trabalhadores: o trabalhador manual e o trabalhador do conhecimento. Um trabalhador manual executa atividades que dependem basicamente de habilidades manuais e que não se baseiam no uso intensivo do conhecimento. São atividades relativamente fáceis de automatizar por serem altamente determinísticas e repetitivas. O trabalhador do conhecimento, em contrapartida, é aquele que produz com base no uso intensivo da criatividade e do conhecimento” [3]

Um dos principais motivos para o surgimento da “metodologia” Ágil para o desenvolvimento de *Software* deve-se pois desenvolver *Software* é uma atividade que depende de criatividade e conhecimento, porém, as metodologias existentes na época eram focadas para trabalhos manuais, e não eram eficientes para esse ramo.

Para Vinicius Teles “Inúmeros estudiosos, entre eles Drucker, DeMarco, Lister, Toffler e DeMarsi têm demonstrado que a produtividade do trabalho do conhecimento deriva de fatores completamente diferentes daqueles usados para o desenvolvimento tradicional. De fato, eles mostram que as premissas da produtividade do trabalho do conhecimento são praticamente opostas às do trabalho manual e o grande erro cometido na maioria dos projetos de *Software* é desconsiderar este fato e adotar as mesmas práticas do ambiente industrial.” [8]

2.1.4.1 Surgimento:

Em meados de 1995 começaram a surgir metodologias de desenvolvimento que se opunham as metodologias tradicionais, como o Cascata e Prototipagem. Esses novos métodos também eram chamado de métodos “leves”, uma vez que os antigos eram conhecidos como métodos “pesados”.

Os modelos tradicionais possuíam um foco maior na análise de requisitos e projeto, os novos modelos focavam principalmente no desenvolvimento e nos testes. Isso possibilitou uma liberdade maior por parte do cliente de requisitar mudanças no

Software em desenvolvimento, uma vez que para realizar mudanças não era necessário voltar à estaca zero. Estimasse que apenas 16% dos *Software*s desenvolvidos utilizando metodologias tradicionais tenham tido sucesso.

Começaram a surgir modelos como Extreme Programing (XP) e o *Scrum*, esses modelos priorizavam o *Software* em si, utilizando apenas o necessário em documentação, isso para priorizar a rapidez na entrega de um protótipo ao cliente para que o mesmo possa mostrar onde deseja alterações ou novos requisitos, isso para assegurar que o desejo do cliente com o *Software* seja atingido, assim assegurando sua qualidade.

Essa abordagem utilizasse de recursos como iterações, entrega incremental (após entregue uma versão inicial ao cliente com apenas funcionalidades básicas essa será expandida e melhorada), envolvimento do cliente com o projeto (repassando o feedback dos usuários do *Software* que fora desenvolvido e sua própria perspectiva também), busca pela qualidade do *Software* e sua simplicidade e pôr fim a união da equipe de desenvolvimento.

Em fevereiro de 2001, dezessete desenvolvedores se reuniram e publicaram o manifesto Ágil (Manifesto for Agile *Software* Development). Para fins de entendimento, ser ágil não é ser veloz, um Guepardo é veloz, se desloca rapidamente, mas uma Lebre é ágil por conseguir mudar sua direção rapidamente. Esse conceito define exatamente o método ágil, não é desenvolver rapidamente, e sim ter a capacidade de mudar sua direção rapidamente, aceitar e modificar o projeto quando necessário.

Principais valores:

> “Estamos descobrindo maneiras melhores de desenvolver *Software*, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo.
>
> Através deste trabalho, passamos a valorizar:
>
> Indivíduos e interações mais que processos e ferramentas;
> *Software* em funcionamento mais que documentação abrangente;
> Colaboração com o cliente mais que negociação de contratos;
> Responder a mudanças mais que seguir um plano;
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.”. [9]

1- Indivíduos e interações mais que processos e ferramentas:

Indivíduos e interações são essenciais para equipes de alto rendimento, estudos sobre falta de comunicação nas empresas mostraram que quando não existe nenhum problema de comunicação as equipes podem atingir um rendimento de até 50 vezes mais que a média da indústria. Para facilitar a comunicação, as metodologias ágeis contam com frequentes ciclos de inspeção e adaptação. Esses ciclos podem variar de alguns minutos de programação em par, muitas horas com interação continua, reuniões rápidas todos os dias, à até toda interação com revisão e retrospecção.

Apenas aumentar a frequência de feedback e comunicação não é o suficiente para se livrar de problemas de comunicação, para que isso funcione é preciso que cada funcionário possuam alguns comportamentos:

- Respeitar o valor de cada pessoa.
- Confiança em cada comunicação.
- Transparência de toda informação, ação e decisão.
- Confiar que cada pessoa vai ajudar a equipe.
- Comprometer-se com a equipe e com suas metas.

Como forma de promover esse tipo de comportamento a gestão ágil deve fornecer um ambiente de suporte e os treinadores/monitores das equipes devem facilitar a inclusão de cada membro da equipe.

2- *Software* em funcionamento mais que documentação abrangente:

Software funcional é um dos grandes diferenciais da metodologia ágil. Todas as metodologias ágeis que representam o manifesto ágil entrega pequenos pedaços do *Software* ao cliente em intervalos de tempo.

As equipes de desenvolvimento ágil devem estabelecer o que querem dizer com "*Software* funcional". De maneira geral, uma parte da "funcionalidade" é concluída quando suas ferramentas passam todos os testes e podem ser utilizadas pelo usuário final. As melhores equipes, incluem também em sua visão de "funcional" testes de integração, desempenho e também levam em consideração a aceitação do cliente.

Alguns passos que podem ser tomados para reduzir grande parte de defeitos são:

- Definir testes de aceitação quando se definir o recurso.
- Implementar recursos em série e em ordem de prioridade.

- Rodar testes de aceitação em cada recurso/ferramenta o quando antes quando forem implementadas
- Corrigir problemas (bugs) de alta prioridade o quanto antes.

3- Colaboração com o cliente mais que negociação de contratos:

Nas duas últimas décadas a taxa de sucesso de projetos mais que dobrou no mundo. Esses aumentos ocorreram como resultado de pequenos projeto e entregas mais frequentes de produtos de *Software*, o que permite o cliente a prover feedback no *Software* que está sendo trabalhado em intervalos regulares. Os escritores do manifesto ágil eram claramente a favor da participação do cliente no processo de desenvolvimento pois entendiam que isso era essencial para o sucesso do projeto.

As metodologias ágeis promoveram esse valor por ter um cliente advogando trabalho lado a lado com a equipe de desenvolvimento. A primeira equipe de *SCRUM* tinham milhares de clientes. Por causa disso não era possível envolve-los todos no desenvolvimento do projeto, por isso, definiram uma única pessoa e o nomearam de “proprietário do produto” (*Product Owner* em inglês). O proprietário do produto não representava apenas todos os clientes, mas também gerenciava as vendas, suporte, serviços ao cliente dentre outras tarefas. O proprietário era responsável por atualizar a lista de requerimentos a cada 4 semanas quando a equipe lançasse alguma iteração, levando em conta principalmente o feedback dos clientes e principais envolvidos. Dessa forma os clientes ativamente ajudavam a dar forma ao *Software* em desenvolvimento.

É devido a colaboração do cliente que a indústria tem mostrado que projetos ágeis tem duas vezes mais taxa de sucesso que os projetos tradicionais na média mundial. Por causa das metodologias ágeis reconhecerem o valor do engajamento do cliente com o projeto foi criado um logar nas equipes de desenvolvimento especificamente para representar o cliente.

4- Responder a mudanças mais do que seguir um plano:

Para as equipes criarem produtos que vão satisfazer clientes e fornecer valor de negócio, elas devem responder as mudanças. Os dados da indústria mostram que 60% dos produtos ou projetos requerem mudanças durante seu desenvolvimento. Mesmo projetos que utilizam metodologia tradicional são entregues dentro do cronograma, dentro do orçamento, com todos os recursos planejados, os clientes geralmente ficam infelizes pois o produto final não era exatamente o que estavam

esperando. A “Lei de Humphrey” diz que o cliente nunca sabe o que querem até que vejam o *Software* funcionando. Se os clientes não verem o *Software* funcionando até o fim do projeto, já será tarde demais para incorporar suas novas necessidades. As metodologias ágeis procuram o retorno (*feedback*) do cliente durante o desenvolvimento do projeto para que possam incorporar as mudanças e as novas informações ao produto final.

Todas as metodologias ágeis tem um processo interno de mudar seus planos em intervalos regulares, baseados no feedback do cliente ou do proprietário do produto. Seus planos são feitos para sempre entregar o produto com o maior valor agregado possível. Por que 80% de valor estão em 20% dos recursos, os projetos ágeis bem executados tendem a serem concluídos mais cedo, enquanto projetos tradicionais tendem a serem concluídos tardiamente. Como resultado, clientes felizes e desenvolvedores apreciando cada vez mais o seu trabalho. Metodologias ágeis são baseadas no conhecimento de que para se atingir o sucesso, deve prever mudanças. É por isso que estabeleceram processos, como revisões e retrospectivas, que são criadas especificamente para priorizar o feedback do cliente e o valor de negócio.

2.1.4.2 Principais Ideologias:

Os principais princípios do manifesto Ágil são:

1. Nossa maior prioridade é satisfazer o cliente
   através da entrega contínua e adiantada
   de *Software* com valor agregado.
2. Mudanças nos requisitos são bem-vindas,
   mesmo tardiamente no desenvolvimento.
   Processos ágeis tiram vantagem das
   mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente *Software* funcionando,
   de poucas semanas a poucos meses,
   com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar
   diariamente em conjunto por todo o projeto.

5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. *Software* funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade--a arte de maximizar a quantidade de trabalho não realizado--é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo. [10]

2.1.4.3 Criadores do Manifesto Ágil:

O Manifesto Ágil foi assinado por Kent Beck e mais dezesseis renomados desenvolvedores, que compartilhavam da ideia formalizar e definir o que era o desenvolvimento ágil, são eles:

1. Kent Beck
2. Mike Beedle
3. Arie van Bennekum
4. Alistair Cockburn
5. Ward Cunningham
6. Martin Fowler
7. James Grenning
8. Jim Highsmith

9. Andrew Hunt
10. Ron Jeffries
11. Jon Kern
12. Brian Marick
13. Robert C. Martin
14. Steve Mellor
15. Ken Schwaber
16. Jeff Sutherland
17. Dave Thomas

Um pouco sobre Kent Beck (um dos mais importantes da lista):
Kent Beck é um engenheiro de *Software* americano criador do Extreme Programming e Test Driven Development. Beck foi um dos 17 signatários originais do Agile Manifesto em 2001. [11]

Frequentou a Universidade do Estado do Oregon entre 1979 e 1987, recebendo o Bacharelado em Ciências da Computação e Mestre em Ciências da Computação.

Beck vive em Medford, Oregon e trabalha no Facebook.[12]

2.1.4.4 Empresas que utilizam a “metodologia” ágil:

Como exemplo de empresas que utilizam-se das “metodologias” ágeis no desenvolvimento de *Software* no Brasil podemos citar:

- “Objective Solutions: Empresa pioneira no uso de XP no Brasil. Sediada em São Paulo, utilizando métodos ágeis, desenvolve sistemas de *Software* em Smalltalk e Java para várias empresas de médio e grande porte como, por exemplo, a SKY Brasil.
- Improve It: Empresa sediada no Rio de Janeiro especializada em consultoria e treinamento em XP.
- Paggo: Uma jovem empresa que desenvolve um produto inovador: pagamentos com cartão de crédito através de telefone celular; o cartão de crédito é operado pela própria operadora de telefonia móvel. O interessante é que não só o sistema de *Software* é desenvolvido usando XP mas também todo o funcionamento da empresa é fortemente influenciado por métodos ágeis. XP foi introduzido na empresa pelo AgilCooper Alexandre Freire em 2003.

- Caelum: A Caelum oferece treinamentos em desenvolvimento ágil para Web 2.0 e também desenvolve soluções de negócio agilmente. Desenvolvem o VRaptor, um arcabouço livre para desenvolvimento ágil de aplicações Web em Java.
- Teamware: Empresa de consultoria, mentoring e treinamento pioneira na organização de cursos abertos de *Scrum* Master e *Scrum* Leader no Brasil.
- LocaWeb: Maior empresa de hospedagens de sites da América Latina, passou a utilizar *Scrum* e XP a partir de 2007.
- Neurobox: A Neurobox oferece treinamentos, consultoria e mentoring para a implantação de métodos ágeis de desenvolvimento de *Software*, tais como *Scrum*, XP e Lean.” [13]

2.1.4.5 Principais Vantagens:

A metodologias ágeis de desenvolvimento de *Software* tem sido muito requisitadas por empresas, tanto grandes quando pequenas, um dos principais motivos para isso é por ser uma maneira diferenciada de se desenvolver *Software* com qualidade e em um período de tempo normalmente mais curto que o normal. Mas quais são as principais vantagens de se utilizar métodos ágeis?

Envolvimento dos participante: Esse tipo de metodologia é focado na interação entre os envolvidos, tanto clientes quanto patrocinadores e principalmente os membros da equipe. Por esse motivo elas causam uma colaboração muito forte entre todos, gerando maior entendimento tanto de requisitos do projeto e também maior motivação para levar o projeto adiante.

Transparência: Como o cliente está em constante contato com a equipe do projeto isso gera um processo mais transparente, principalmente, quando o cliente sabe quais são os custos envolvidos em cada iteração de desenvolvimento, quais são as funcionalidades que determinam o sucesso do produto final e qual é o status do projeto a cada nova fase.

Flexibilidade: Ao contrário de modelos como o cascata, os Métodos Ágeis sempre estão sempre abertos a mudanças tal como estas são facilmente implementadas, uma vez que o planejamento acontece a cada nova Iteração. Essa flexibilidade permite que o produto final possua a maior qualidade possível e contenha todas as funcionalidades que o cliente necessite, sem incorrer a riscos que possam comprometer a finalização do projeto e a satisfação do cliente.

Agilidade: Como o nome já diz, uma das principais vantagens das Metodologias Ágeis é a agilidade com que as funcionalidades do *Software* são desenvolvidas. Cada funcionalidade é desenvolvidas dentro de uma Iteração, o tempo de criação para cada Iteração pode variar e é determinado pela equipe do projeto e, assim, o cliente pode fazer o acompanhamento em temo real de quanto tempo ainda falta para a finalização do produto, além é claro de receber algumas demonstrações ou protótipos do *Software*. Isso, além de facilitar a tomada de decisões no quesito priorização das atividades, ajuda também a uma maior rapidez quando é necessário realizar alguma alteração em parte do *Software*.

Gestão de riscos: Como o planejamento é feito para cada iteração, é possível prever e prevenir possíveis riscos com maior eficácia, impedindo que o projeto sofra um grande impacto por falta de previsibilidade. Além disso, torna-se possível detectar falhas com antecedência, corrigindo-as antes da entrega final do *Software*, ocasionalmente reduzindo as manutenções realizadas.

Controle de custos: A segmentação do projeto em iterações é estratégica para o controle de custos por parte do cliente. Cada nova iteração tem seus custos calculados com maior precisão, o que possibilita ao cliente priorizar as ferramentas com maior valor de mercado para sua empresa a um custo mais baixo, decidindo quais serão as ferramentas priorizadas e não priorizadas a cada etapa que é desenvolvida pela equipe do projeto.

Qualidade do Produto: Por fim, mas não menos importante, a qualidade do produto final é garantida pela correta aplicação das melhores práticas do Manifesto Ágil, bem como por meio da experiência da equipe no desenvolvimento de projetos Ágeis. Teste frequentes devem ser realizados para afirmar o êxito de cada funcionalidade, o que possibilita a identificação de qualquer distúrbio com a antecedência necessária para que a entrega final do produto seja feita dentro do prazo das especificações acordadas com o cliente.

2.2 Metodologia do trabalho:

Este trabalho possui o intuito de expor como foi e está sendo a aplicação de ferramentas e “métodos” de desenvolvimento ágil de *Software* no projeto Indra, desenvolvido pelos alunos do 4º ano do ensino médio integrado ao ensino técnico em informática, para isso, primeiramente apresentaremos algumas das principais

Metodologias Tradicionais de desenvolvimento tal como as Metodologias Ágeis, seus criadores, vantagens, empresas que as utilizam no Brasil dentre outros assuntos.

2.3 ***Kanban*:**

Kanban é um termo de origem japonesa e significa literalmente “cartão”.

É um conceito que se utiliza de cartões (como post-it ou outros tipos) para indicar o andamento de uma determinada tarefa.

Nesses cartões são colocadas indicações sobre uma determinada tarefa, por exemplo, “Para ser feito”, “Em andamento” ou “Finalizado” por exemplo.

A utilização do *Kanban* permite um controle de produção com informações como quando deve ser produzido, o que deve ser feito (por exemplo um sistema de login) e também pode delegar alguém para realizar determinada tarefa.

O método *Kanban* fora inicialmente utilizado em empresas japonesas de Fabricação em série como montadoras de veículos entre outras e a empresa responsável pela introdução desse método foi a Toyota, devido a sua necessidade de manter um funcionamento eficaz de seu sistema de produção em série.

O *Kanban* eletrônico (e-*Kanban*) é utilizado em substituição ao modelo físico, evitando alguns problemas como a perda de cartões e proporcionando mais rapidez na atualização do quadro de tarefas.

Hoje em dia o *Kanban* é muito utilizado em conjunto com o *Scrum*, por serem duas metodologias utilizadas no desenvolvimento ágil de *Software*.

O *Kanban* está sendo utilizado no projeto Indra, fora imposto sua utilização no começo do projeto pelo responsável pelo projeto, o professor Breno Lisi Romano.

Em nosso projeto fora definido ao menos um responsável em cada modulo para que esse cobrasse a atualização do e-*Kanban* da equipe, ficando em cima para realizarem atualizações em cada cartão.

O e-*Kanban* de nosso projeto está dividido em 6 módulos, são eles: Projeto Geral – IFsPCD, Módulo 01: USUÁRIOS, Modulo 02: PCDs, Módulo 03: MEDIÇÕES, Módulo 04: RELATÓRIOS e Módulo 05: ALERTAS. Cada um desses módulos possuem as mesmas quatro colunas de progresso, são elas: Backlog (ou seja, o que ainda precisa ser feito), Em Andamento, Validação/Testes e Finalizado.

Abaixo segue algumas imagens (Figura 6 e Figura 7) mostrando como é o *Kanban* utilizado.

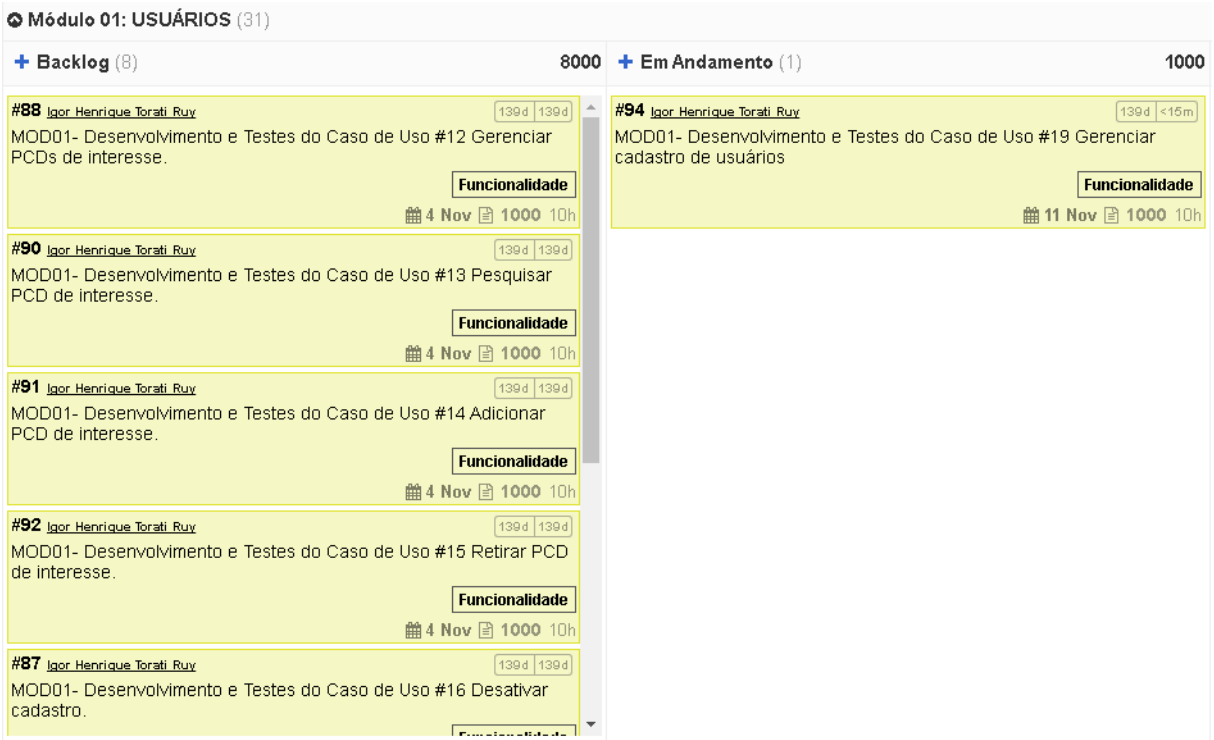

Figura 6 *Kanban* Módulo 1 parte 1 (“Backlog” e “Em andamento”).

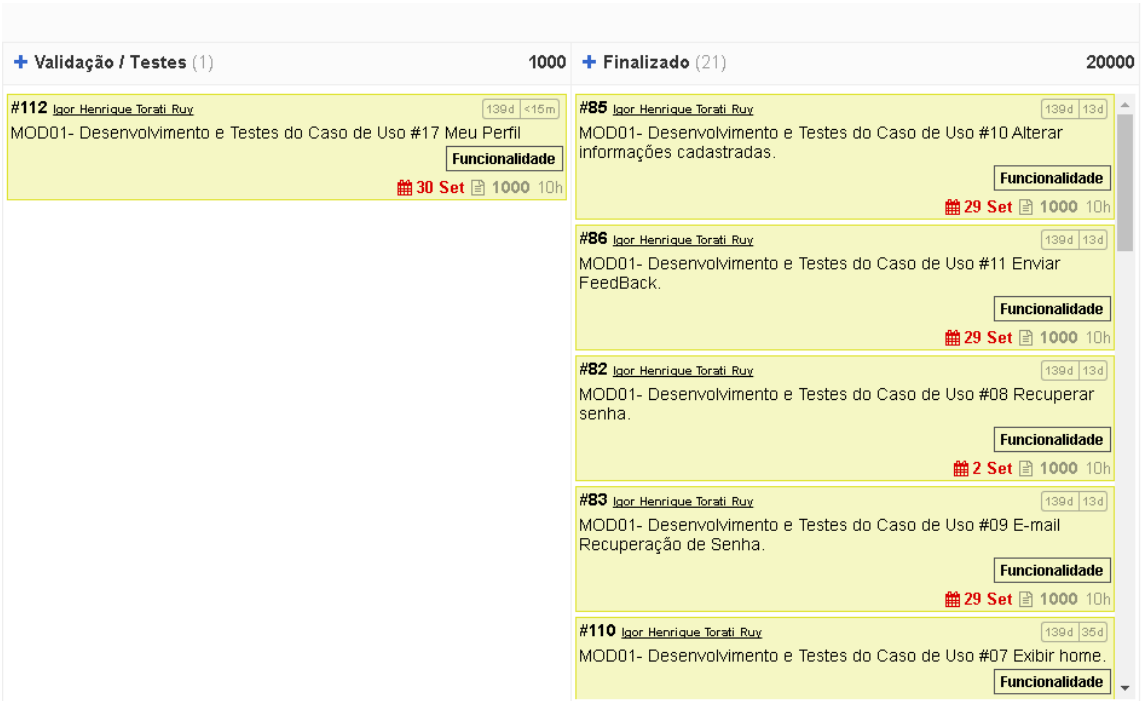

Figura 7 *Kanban* Módulo 1 parte 2 (“Validação" e "Finalizado")

Para facilitar a navegação e a realização de alterações foi definido uma cor de cartão para cada módulo, na cores definidas foram: Projeto Geral (Rosa), Módulo 01 (Amarelo), Módulo 02 (Verde), Módulo 03 (Cinza), Módulo 04 (Azul) e Módulo 05 (Violeta). Segue uma visão geral da *Board* do nosso projeto (Figura 8):

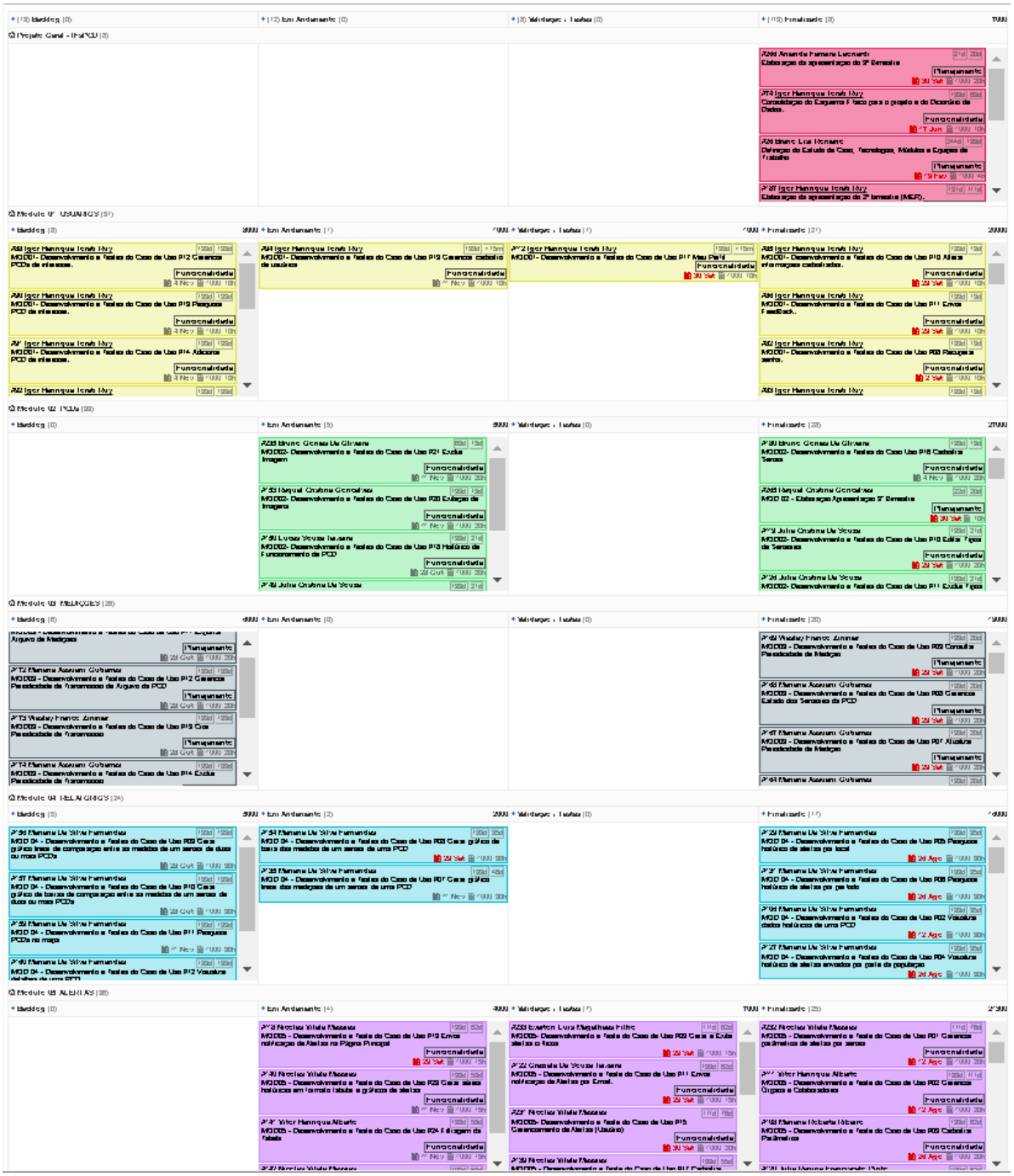

Figura 8 Visão Geral *Kanban* Projeto Indra

2.4 *Redmine*:

Segundo a Wikipédia: “***Redmine*** é um *Software* livre, gerenciador de projetos baseados na web e ferramenta de gerenciamento de bugs. Ele contém calendário e gráficos de Gantt para ajudar na representação visual dos projetos e seus deadlines (prazos de entrega). Ele pode também trabalhar com múltiplos projetos.” [14]

De maneira geral, o *Redmine* é também um *Kanban*, acrescido de muitas outras ferramentas para facilitar a visualização do andamento de seus projetos, como o gráfico de Gantt e o calendário.

No projeto uma ferramenta que foi muito utilizada foi o gráfico de Gantt para visualizar o andamento do projeto juntamente com sua data de entrega, segue imagem do gráfico (Figura 9):

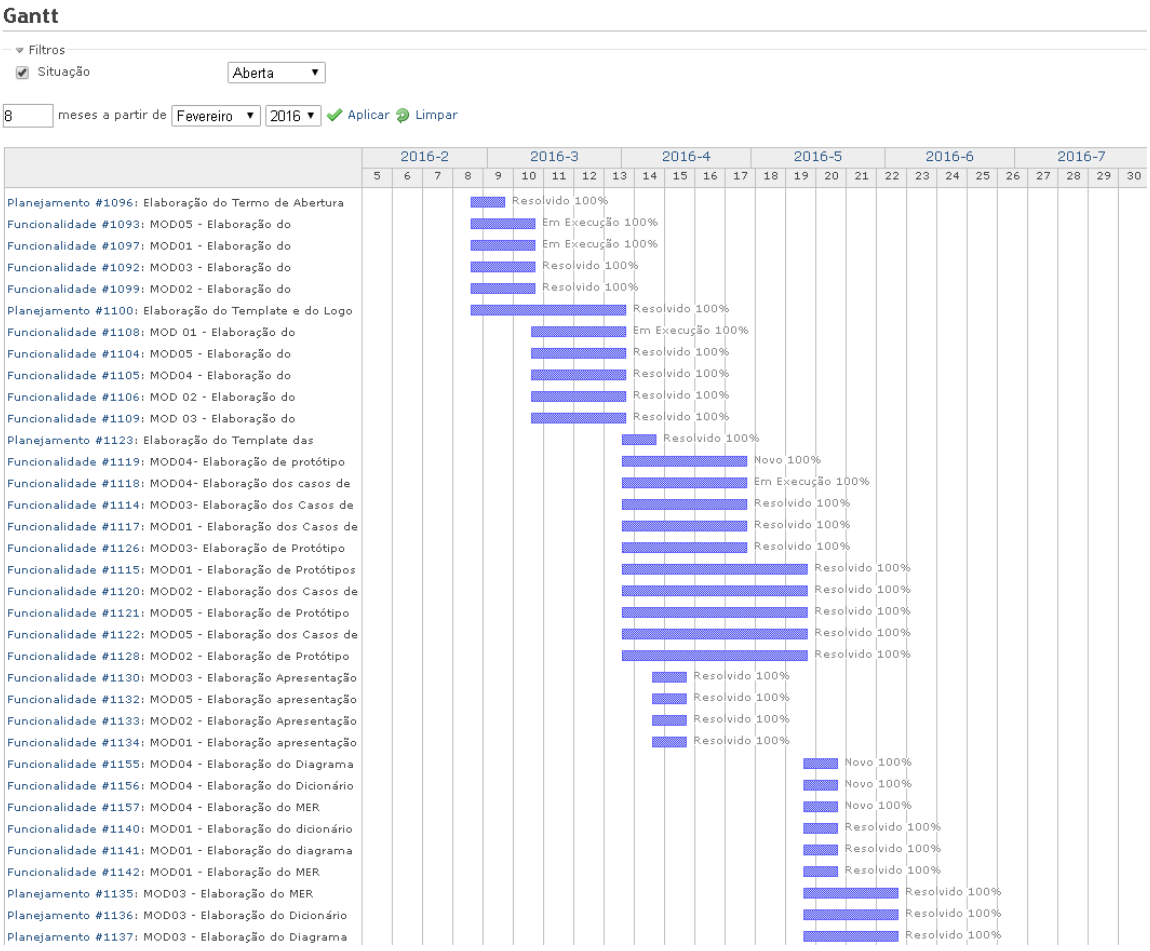

Figura 9 *Redmine*, visualização do Gráfico de Gantt

Outra funcionalidade interessante do *Redmine* é a visualização de prazos pelo Calendário, como mostra a imagem abaixo na Figura 10:

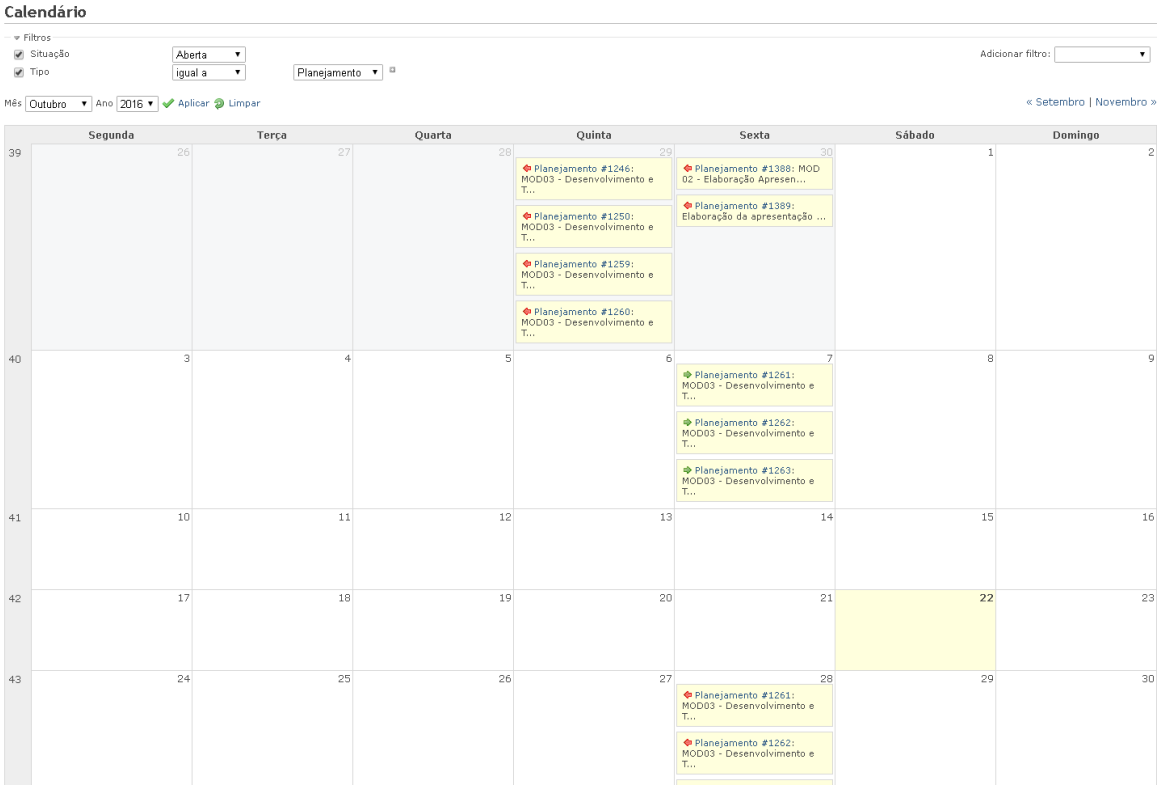

Figura 10 *Redmine*, Calendário do Projeto

Outras funcionalidades interessantes do *Redmine* são: a criação de equipes para realizar alguma atividade (ferramenta que no *Kanban* está disponível apenas para definição de um único responsável), a possibilidade de criar uma Wiki, como mostra a Figura 11, a possibilidade de criar versões para o projeto em questão, e também a possibilidade de se criar um fórum para discussões dentro do próprio *Redmine*, como mostra a Figura 12.

Apesar de possuir todas essas ferramentas o *Redmine* é um pouco mais complexo que o *Kanban* e requer um pouco mais de paciência de seus usuários, porém, mesmo com esse problema ainda sim é altamente recomendado e foi muito útil para a realização do projeto Indra.

Vale salientar que o principal motivo do uso do *Redmine* no projeto foi justamente pelo gráfico de Gantt.

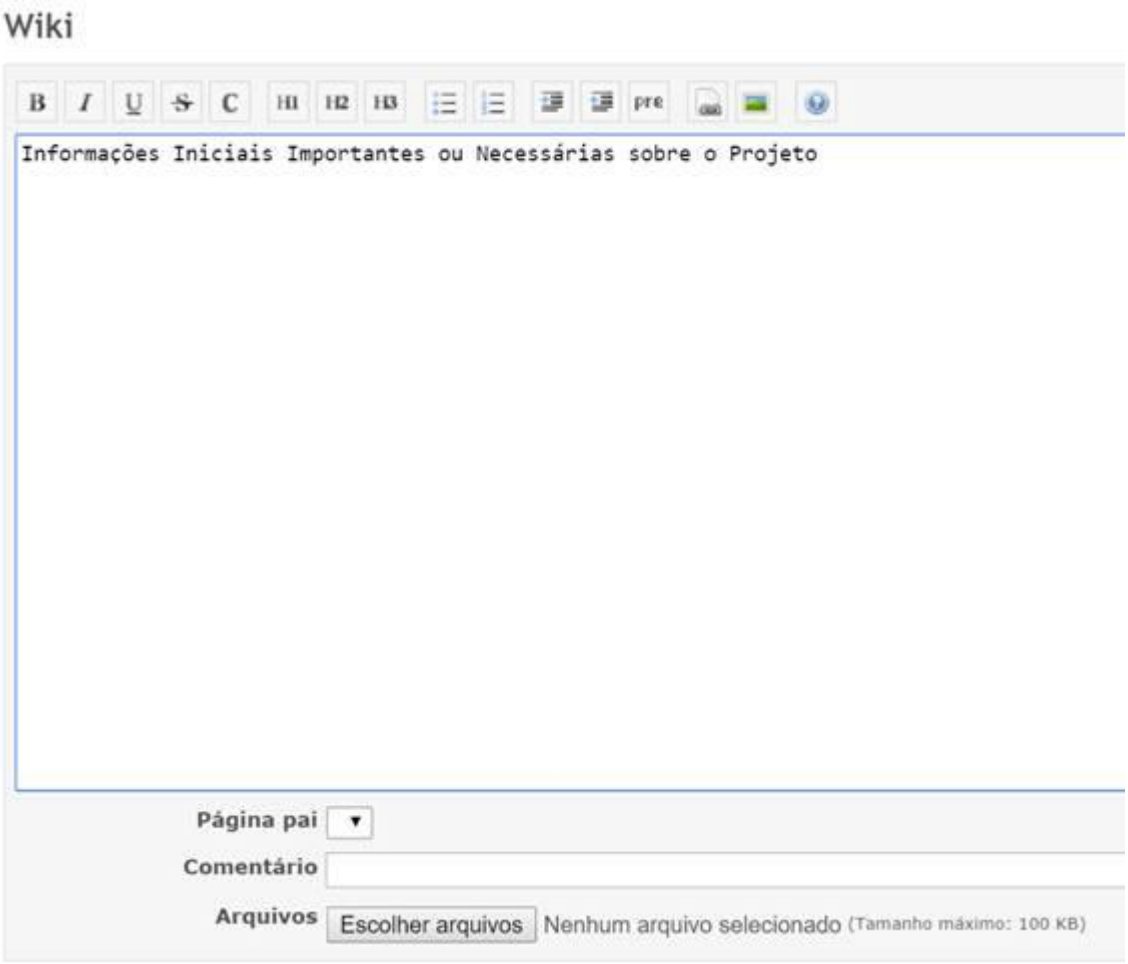

Figura 11 Ferramenta Wiki do *Redmine*

Figura 12 Ferramenta de Versões do *Redmine*

2.5 *Scrum*:

Scrum é uma **metodologia ágil** de gestão e planejamento de projetos de *Software*.

No *Scrum* os projetos são divididos em ciclos, mais conhecidos como *Sprints*. Os *Sprints* são como iterações do projeto, em outras palavras, são um conjunto de atividades que devem ser executadas em um prazo de tempo determinado, normalmente um mês.

O *Scrum* possui uma lista conhecida como *Product Backlog*, no início de cada Sprint é feita uma reunião de planejamento onde o Proprietário do Produto define as atividades que devem ser priorizadas no *Product Backlog* daquele aquele Sprint.

A cada dia de uma Sprint é realizada uma reunião (*Daily Scrum*) no início do expediente, essa reunião possui o intuito de disseminar o conhecimento do que fora feito no dia anterior, identificar as dificuldades e impedimentos e definir quais tarefas deveram ser priorizadas naquele dia.

Ao se terminar um Sprint é feita uma reunião onde serão apresentadas as funcionalidades implementadas. Após a apresentação é realizada uma nova reunião de planejamento para o próximo *Sprint*, e esse ciclo se repetirá até a finalização do projeto. O ciclo do *Scrum* pode ser visualizado na Figura 13.

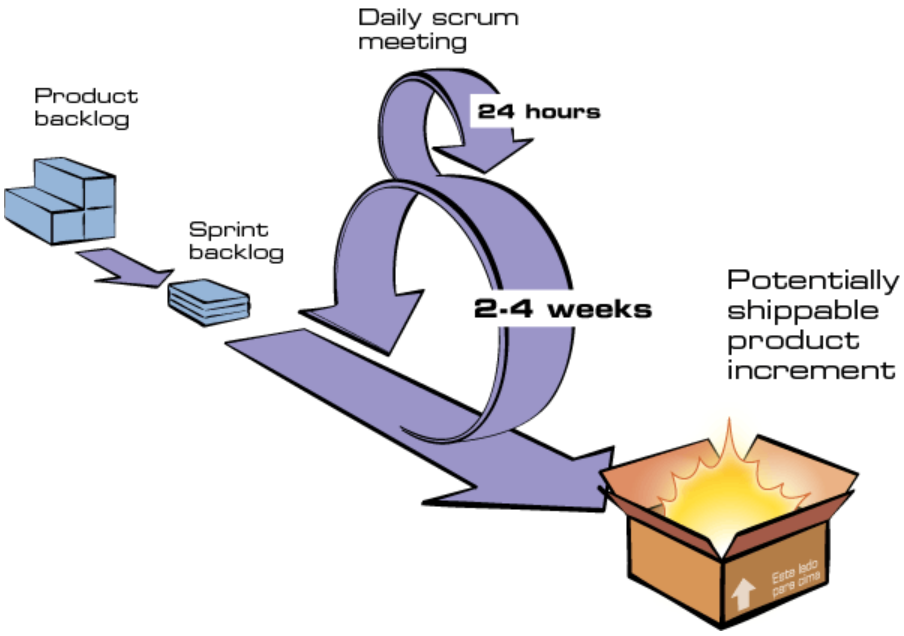

Figura 13 Ciclo de vida de um Projeto com *SCRUM*

As técnicas do *Scrum* utilizadas no projeto foram, principalmente, a *Daily Scrum*, a ideia de *Sprints*, no caso iterações em si realizadas em um período de um

mês aproximadamente, porém as atividades a serem realizadas foram definidas no início do projeto, a lista de *Product Backlog*, criada no início do projeto e alterada durante a análise de requisitos, casos de uso, desenvolvimento e criação do banco de dados e também foram realizadas reuniões e apresentações ao final de cada *Sprint*(Iteração).

As Figuras 14 e 15 mostram a lista de *Product Backlog* geral do projeto Indra:

	Cronograma de Atividades do Projeto PDS 2016 Vespertino				
	Status	**Data Real de Entrega**	**Data Inicialmente Prevista**	**Título da Atividade**	**Descrição**
1º Bimestre	#01		19/fev	Estudo de Caso	Decidir qual será o Contexto do Projeto Acadêmico??
	#02		19/fev	Definição da Plataforma de Desenvolvimento	Linguagem de Programação e Banco de Dados
	#03		19/fev	Definição das Equipes	Analistas de Sistemas, Desenvolvedores, DBAs, Testadores e Equipe de Documentação
	#04		19/fev	Definição dos Módulos (Macrorequisitos)	Definição dos Módulos do Estudo de Caso escolhido e os respectivos macro-requisitos a serem contemplados
	#05		26/fev	Definição das Responsabilidades de Gerência	Divisão das responsabilidades de gerência entre os integrantes de cada um dos módulos do projeto.
	#06		04/mar	Termo de Abertura do Projeto	Definir o Contrato Inicial do Projeto
	#07	18/mar	11/mar	Identificação dos Stakeholders	Nome, Informações de Contato e Classificação do Stakeholder (Cliente, Acionista, Governo, Parceiro, Patrocinador, ...)
	#08	18/mar	11/mar	Documento de Visão para Cada Módulo	Utilizar o Template do RUP para definir Problema e Solução Proposta no Documento de Visão.
	#09		01/abr	Definir o Template do Projeto	Os desenvolvedores de todos os módulos devem se reunir e propor um template para o projeto. Definir o logo da empresa. Já deve ser implementado este template em HTML+CSS+PHP. Deve ser definido o padrão de pastas para o desenvolvimento (imagens, css, javascript, etc...)
	#10	08/abr	01/abr	Documento de Especificação dos Principais Envolvidos para cada Módulo	Utilizar o Template do RUP para Especificação dos Principais Envolvidos (Stakeholders + Requisitos Funcionais)
	#11		15/abr	Apresentação do Primeiro Bimestre	Apresentação das atividades realizadas no Primeiro Bimestre por cada um dos Módulos
	#12	18/mai	29/abr	Casos de Uso para cada Módulo e Estimativa de Esforços	Elaborar o Diagrama de Casos de Uso para cada um dos Módulos (Preencher o Documento de MCU do RUP). Elaborar uma estimativa de esforços (custo/tempo) para cada um dos módulos.
	#13	18/mai	29/abr	Elaboração dos Protótipos de IHMs	Elaborar, baseado no template definido, os protótipos de IHMs para cada caso de uso do Projeto.
	#14	Não Necessário	Não Necessário	Refinamento das Atividades do Primeiro Bimestre - Por Módulo	Nesta atividade, os alunos devem finalizar todas as outras atividades que estão pendentes até o momento, principalmente elaboração de casos de uso e seus respectivos protótipos.
2º Bimestre	#15	03/jun	20/mai	Elaboração do Diagrama Conceitual de cada Módulo	Os DBAs serão responsáveis em elaborar o Diagrama Entidade-Relacionamento (MER) para cada um dos módulos (Entidades, Relacionamentos e Atributos) e Definir o Dicionário de Dados.
	#16	03/jun	20/mai	Elaboração do Dicionário de Dados do BD de cada Módulo	Os DBAs serão responsáveis em elaborar o Diagrama Entidade-Relacionamento (MER) para cada um dos módulos (Entidades, Relacionamentos e Atributos) e Definir o Dicionário de Dados.
	#17	03/jun	20/mai	Elaboração do Diagrama de Classes de cada Módulo.	O módulo deve ficar responsável em elaborar o Diagrama de Classes, a partir do MER definido, utilizando uma ferramenta CASE para auxiliá-la. Deve entregar esta atividade na aula de AW2, pois será utilizada para iniciar o desenvolvimento do Projeto.
	#18		10/jun	Planejamento: Definir as Entregar de Cada Módulo para Implementação e Testes	Agrupar casos de uso diretamente relacionados e definir as entregas de cada módulo, ou seja, o que será implementado e testado em conjunto. Considerar as datas do terceiro e quarto bimestre e também a Semana da Tecnologia.
	#19		17/jun	Elaboração dos Casos de Teste para cada Módulo	Os Testadores deverão elaborar, juntamente com a Equipe, os casos de teste para cada um dos Módulos (Preencher o Documento do RUP).
	#20		17/jun	Consolidação do Esquema Físico para o Projeto e do Dicionário de Dados	Os DBAs ficarão responsáveis para consolidar todos os esquemas físicos e dicionários de dados dos módulos em um único para ser utilizado por todos. Os DBAs ficarão responsáveis pelas seguintes atividades: Definir o Dicionário de Dados, Transformar o MER em Modelo Relacional, Conferir a Transformação proposta pelo brModelo e Gerar o Modelo Físico.
	#21		01/jul	Entrega da Iteração #01 de Casos de Uso	Entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#22		01/jul	Apresentação do Segundo Bimestre (Incremental)	Apresentação das atividades realizadas no Primeiro e Segundo Bimestres por cada um dos Módulos.

Figura 14 Planejamento Geral do Projeto, parte 1

3º Bimestre	#23	19/ago	12/ago	Revisão da Iteração #01 de Casos de Uso	Revisão e Refinamento da entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#24		12/ago	Definição do Tema e Apresentação do Objetivo do Relatório Técnico (Vulgo TFC)	Definição do Tema e Apresentação do Objetivo do Relatório Técnico (Vulgo TFC). Valerá parte da nota do 3 e do 4 bimestre. Para os alunos que não fizeram estágio, servirá de equivalência.
	#25	09/set	26/ago	Entrega da Iteração #02 de Casos de Uso	Entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#26		02/set	Término do Capítulo 01 do Relatório Técnico (Vulgo TFC)	Elaboração do Capítulo 01 do TFC, contendo Contextualização, Motivação da Pesquisa e Objetivo.
	#27		29/set	Entrega da Iteração #03 de Casos de Uso	Entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#28		30/set	Entrega da Versão Parcial do Capítulo 02 do Relatório Técnico (TFC)	Elaboração de uma versão parcial do Capítulo 02 do TFC, detalhando a Metodologia do Trabalho.
	#29		30/set	Apresentação do Terceiro Bimestre (Incremental)	Apresentação das atividades realizadas no Terceiro Bimestre por cada um dos Módulos.
4º Bimestre	#30		04/nov	Entrega da Iteração #04 de Casos de Uso	Entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#31		28/out	Entrega do Relatório Técnico do Projeto (Vulgo TFC)	
	#32		24/nov	Entrega da Iteração #05 de Casos de Uso	Entrega dos casos de uso da primeira iteração de Casos de Uso de Cada Módulo já implementados e testados (Terceiro Bimestre).
	#33		24/nov	Manual do Usuário	Seguir todas as recomendações do conjunto de slide a parte no Portal da Disciplina.
	#34		24/nov	Portal de Documentações	
	#35		24/nov	Manual de Implantação	
	#36		24/nov	Integração entre todos os Módulos	
	#37		25/nov	Apresentação do Quarto Bimestre (Incremental)	Apresentação das atividades realizadas no Quarto Bimestre por cada um dos Módulos.

Figura 15 Planejamento Geral do Projeto, parte 2

A tabela abaixo mostra o planejamento feito para os Sprints/Iterações do Modulo 01:

Tabela 1 Planejamento de Iterações Modulo 1

Planejamento das Iterações para o Desenvolvimento dos Casos de Uso - Módulo de Cadastros			
Iteração	**Data Prevista**	**Casos de Uso**	**Equipe Responsável *2**
#01	12/08/2016	Cadastrar Usuários.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Logout.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Logar no sistema.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
#02	26/08/2016	Verificar acessos necessários de usuários	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Confirmar cadastro de usuário	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Enviar e-mail de confirmação de cadastro.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Exibir home.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
#03	29/09/2019	Recuperar senha.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Enviar E-mail Recuperação de Senha.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Alterar informações cadastradas.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Enviar *FeedBack*.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
#04	04/11/2016	Gerenciar PCDs de interesse.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Pesquisar PCD de interesse.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Adicionar PCD de interesse.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Retirar PCD de interesse.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)

		Desativar cadastro.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Meu perfil.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
#05	11/11/2016	Exibir informações sobre o projeto.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Gerenciar cadastro de usuários.	Giovanna (Analista), Anderson (Desenvolvedor) e Amanda (DBA)
		Pesquisar cadastros existentes.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)
		Editar cadastros existentes.	Leonardo (Analista), Aleccheévina (Desenvolvedor) e Igor (DBA)

3 Conclusões e Recomendações

As metodologias tradicionais possuem como característica uma grande quantidade de documentação gerada no processo de desenvolvimento de *software* que, muitas vezes, é apontada como a causa de atraso em muitos projetos. E as metodologias ágeis que focam no código, na qualidade do produto, no cliente e evitam, se possível, a documentação.

De modo geral, o projeto Indra, que está sendo desenvolvido pelos alunos do 4º ano do curso Técnico em informática integrado ao ensino médio, possui muitas bases relacionadas ao desenvolvimento Ágil de *Software*, é possível notar essas bases pelo fato de que estão sendo utilizadas ferramentas de gerenciamento Ágil como é o caso do *Kanban* e *Redmine* e também grande parte da análise e gestão do projeto foi baseado na Metodologia de Desenvolvimento Ágil *Scrum*.

Apesar desses esforços, técnicas como XP (*eXtreme Programing*) não puderam ser utilizadas devido ao número relativamente pequeno de alunos disponível para criação do projeto.

Embora tenha ocorrido alguns problemas de comunicação, tempo necessário para adaptação às ferramentas e as técnicas utilizadas no projeto é notável que o projeto assim como a matéria de PDS (Prática de Desenvolvimento de Sistemas) foi de grande aprendizado a todos os alunos, não limitado apenas no quesito de métodos ágeis, mas também para criação de relações entre a classe e amadurecimento mutuo da maioria dos envolvidos.

4 Referências Bibliográficas

[1] – Flavia Reisswitz, Livro Análise de Sistemas Vol. 1, 2008

[2]- Marcel Horner e Rafael De Andreis Pires, Resumos Projeto 1, 2004. Disponível em: http://projetos.inf.ufsc.br/arquivos_projetos/projeto_14/Resumo%20dos%20Artigos.doc. Acessado em 01/10/2016.

[3] - Roger Pressman, Engenharia de *Software* 7° edição, 2011.

[4] – Leandro Costa, As Metodologias Tradicionais de Desenvolvimento de Software, 2011. Disponível em: http://www.semeru.com.br/blog/as-metodologias-tradicionais-de-desenvolvimento-de-software/. Acessado em 05/10/2016

[5] - BROOKS, apud PRESSMAN, 2006.

[6] – Wikipédia, Desenvolvimento Ágil de Software. Disponível em: https://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software. Acessado em 08/10/2016

[7] – Wikipédia, Peter Drucker. Disponível em: https://pt.wikipedia.org/wiki/Peter_Drucker. Acessado em 08/10/2016

[8] – Vinicius AC., Introdução ao Desenvolvimento Ágil. Disponivel em: www.devmedia.com.br/introducao-ao-desenvolvimento-agil/5916. Acessado em 03/10/2016

[9] – Agile Manifesto, Manifesto for Agile Software Development, 2001. Disponível em: http://agilemanifesto.org/. Acessado em 12/10/2016

[10] - Manifesto Ágil, Princípios por Trás do Manifesto Ágil, 2001. Disponível em: http://www.manifestoagil.com.br/principios.html. Acessado em 13/10/2016

[11] – Conhecimento Geral, Kent. Disponível e: http://www.conhecimentogeral.inf.br/kent_beck/. Acessado em 15/10/2016

[12] – Kent Beck, Kent Beck. Disponível em: https://www.linkedin.com/in/kentbeck. Acessado em 15/10/2016

[13] – AgilCoop. Disponível em: http://ccsl.ime.usp.br/agilcoop/empresas_ageis. Acessado em 15/10/2016

[14] – Wikipédia, Redmine. Disponível em: https://pt.wikipedia.org/wiki/*Redmine*. Acessado em 22/10/2016