UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM QUALIDADE DE SOFTWARE

EDISON VANDERLEI DE LIMA

UMA NOVA ABORDAGEM DA PROTOTIPAÇÃO À LUZ DO MÉTODO ÁGIL SCRUM

São Leopoldo

2017

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUIDA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM QUALIDADE DE SOFTWARE

EDISON VANDERLEI DE LIMA

UMA NOVA ABORDAGEM DA PROTOTIPAÇÃO À LUZ DO MÉTODO ÁGIL SCRUM

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Qualidade de Software, pelo curso de Pós-Graduação Lato Sensu em Qualidade de Software da Universidade do Vale do Rio dos Sinos – UNISINOS.
Orientador: Profa. Me. Josiane Brietzke Porto

São Leopoldo

2017

Uma Nova Abordagem da Prototipação à Luz do Método Ágil SCRUM

Edison V. de Lima[1]

[1]Unidade Acadêmica de Pesquisa e Pós-Graduação – Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo – RS – Brasil

Edison.lima1988@gmail.com

Abstract. *The quest for improving software quality is currently one of the major engineering tasks primarily due to the continuous and rapid growth of software systems. This implies that the construction processes are always in continuous improvement. It is of extreme interest that these processes can be copied and extended to new organizational units or mass production lines. In response to this came the "agile methodologies" that agile delivery of the increments, focusing on the customer and change, but even with the use of these methods in software development, it is still necessary to use new techniques that can supply the Remaining gaps and that increase the continuous response to such fundamental changes in the agile study. In this way, it is of interest in this article to present qualitative and quantitative data that prove the use of the prototyping technique in the agile SCRUM methodology as a success factor in improving quality, raising its benefits and potential harms.*

Resumo. *A busca pela melhoria da qualidade de software é atualmente uma das grandes tarefas da engenharia ocasionadas principalmente pelo contínuo e rápido crescimento dos sistemas de software. Isso implica que os processos de construção estão sempre em contínuo aperfeiçoamento. É de extremo interesse que estes processos possam ser copiados e estendidos a novas unidades organizacionais ou linhas de produção em massa. Em resposta a isso surgiram as "metodologias ágeis" que visam a entrega ágil dos incrementos, focalizando o cliente e a mudança. Todavia mesmo com o uso destes métodos no desenvolvimento de software, ainda se faz a necessidade do emprego de novas técnicas, que possam suprir as lacunas remanescentes e que aumentem a resposta contínua as mudanças, tão fundamental no estudo ágil. Desta forma, este artigo através de um estudo de caso apresenta dados qualitativos e quantitativos, que comprovam o uso da técnica de prototipação na metodologia ágil SCRUM, como um fator de sucesso na melhoria da qualidade levantando-se seus benefícios e possíveis malefícios.*

1. Introdução

Hoje, na segunda década no século XXI observa-se uma dependência da tecnologia, fato confirmado pelos softwares usados em usinas nucleares, computadores de bordo, computadores autônomos de carros, aviões não tripulados, drones, bem como na construção de inteligências artificiais (PRESSMAN, 2011).

Mesmo com a constante evolução de hardware e software os processos de construção ainda enfrentam problemas metodológicos que perduram na atualidade, ainda que criadas novas metodologias para mitiga-los tais como os modelos tradicionais: cascata, modelo espiral, incremental e iterativo, prototipação e dirigidas a plano.

Embora estes modelos possam ser antigas metodologias de desenvolvimento, ainda são métodos de referência e de grande contribuição para o desenvolvimento de software no século XXI, pois muitas empresas atualmente fazem uso destes modelos adaptados e condicionados a sua realidade. Confirma Sommerville (2011, p. 19) que:

> Esses modelos genéricos não são descrições definitivas dos processos de software. Pelo contrário, são abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de software. Você pode vê-los como frameworks de processos que podem ser ampliados e adaptados para criar processos de engenharia mais específicos.

Pode-se, então, combinar vários processos e metodologias de forma a atingir maior lucratividade, produtividade e agilidade dentre uma organização. Entretanto, muitas organizações ainda se sentem confortáveis a usar modelos tradicionais de desenvolvimento aplicadas aos seus negócios.

Com o crescimento exponencial na construção de softwares para negócios, no início do ano 2000, novas abordagens na construção tiveram que ser estudadas e encontradas, motivadas, principalmente, pela mudança rápida nos requisitos do negócio. Ou seja, muitas aplicações estavam perdendo o sentido de sua construção com o passar de meses de trabalho, porque tão logo estavam concluídas, os requisitos iniciais já haviam mudado tanto que a sua razão de existir era desnecessária, levando muitos softwares a serem descartados.

Isto provocado, principalmente, segundo Sommerville (2011) ao fato de que muitas empresas estavam ainda observando paradigmas de construção utilizadas em empresas grandes de desenvolvimento, que gastavam mais tempo com o planejamento do que viria a ser construído não tendo tempo para as fases de construção e testes.

A insatisfação com estes modelos dirigidos a planos tomou conta de uma grande parcela da comunidade de Engenharia de Software, ocasionando o surgimento das chamadas “metodologias ágeis”, regidas a partir de dozes princípios ágeis de desenvolvimento encontradas no “manifesto ágil”, proposto em 2001 (PHAM, 2012).

Assim, surgiram metodologias ágeis, tais como: *Extremme Programming* (BECK, 2000), SCRUM (COHN, 2009; SCHWABER, 2004), *Crystal* (COCKBURN, 2001), *Feature Driven Development* (FDD) (PALMER, FELSING, 2002). E, os nesse contexto, integrando metodologias tradicionais de modelagem a conceitos ágeis, a modelagem ágil (AMBLER e JEFFRIES, 2002) e *Rational Unified Process* (LAEMAN, 2002). Tais modelos aceitam que as mudanças são algo normal e que não devem ser impedidas, mas devem ser fatores de adaptação e melhoria para os processos, bem como que estes processos estejam direcionados a dar suporte aos desenvolvedores e não métodos engessados e definitivos, como praticados nos métodos tradicionais de desenvolvimento.

Diferentemente das metodologias tradicionais, os métodos ágeis primam pela mínima documentação, foco na entrega e na agilidade, ocasionando problemas quanto a

carência de especificação, deixando o conhecimento muito mais legado a experiência dos *stakeholders*.

Construir softwares rapidamente e com o mínimo de informação pode ser algo complicado quando visualizamos tantas mudanças nos requisitos, sem a devida documentação ou quando não está se seguindo uma metodologia direcionada a planos, fato contraposto especialmente pelo uso da rastreabilidade de requisitos. Corrobora Sommerville (2011, p. 24) que "a engenharia de requisitos é um estágio particularmente crítico do processo de software, pois erros nessa fase inevitavelmente geram problemas no projeto e na implantação do sistema". Nisto afirmava Myers (1979) que qualquer erro quando não identificado, o seu custo é multiplicado por 10 e, por isso é muito importante detectar qualquer falha nas fases inicias de construção, já que nesta fase saem muito mais baratos.

Todavia, com o emprego de técnicas próprias do método ágil, dito o *Scrum* é possível ter uma consequente melhoria na qualidade de software, diminuição dos custos e aumento da lucratividade bem como a satisfação do cliente, porém para modelos cada vez mais modernos, maduros e exigentes o uso destes métodos ainda se torna insuficiente. Nisto supõe Sommerville (2011, p. 19) que:

> embora não exista um processo 'ideal' de software, há espaço, em muitas organizações, para melhorias no processo de software. Os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria.

Isto realça a importância de se utilizar artefatos, técnicas e/ou processos que possam elucidar os requisitos ou gerenciá-los, invariavelmente, se o sistema usa metodologias ágeis ou não para a sua construção.

Verificando-se a necessidade da constante adaptação de novas ferramentas, métodos e processos que possibilitem a qualidade de software foi observado o uso do método de prototipação na empresa "ABC", como melhoria do processo de validação dos requisitos. Portanto, este trabalho tem por motivação analisar, entender e avaliar este processo, o qual pode-se satisfazer a seguinte questão de pesquisa: Como a prototipação pode auxiliar na melhoria da qualidade de software num ambiente ágil de desenvolvimento?

Desta forma, o objetivo geral desta pesquisa é analisar um processo de demonstração ao cliente, por meio da técnica de prototipação, no contexto da empresa "ABC", com o propósito de melhoria da qualidade do produto de software. Para atingir este objetivo foram definidos os seguintes objetivos específicos: *(i)* Identificar as técnicas de verificação e validação existentes para metodologias ágeis e tradicionais, analisando comparativamente os seus atributos com o método Scrum; *(ii)* Levantar métricas e dados estatísticos da empresa "ABC", que cooperem com a sustentação da manutenibilidade da técnica como efetiva forma de melhoria da qualidade de software; *(iii)* Validar o método através do estudo de caso da empresa "ABC", no intuito de uma padronização do método no processo de desenvolvimento.

A técnica de prototipação proposta para a empresa "ABC", como uma ferramenta de melhoria pode contribuir nos seguintes aspectos, no contexto da organização: melhora na qualidade da elicitação dos requisitos, ganho no gerenciamento da mudança conforme o desenvolvimento, mitigação de erros, maior envolvimento do

cliente com o projeto, além do aumento da confiança e expectativa do cliente, maior aproximação e interação do time, dentre outros. Não obstante como melhoria da qualidade de software aplicada aos métodos ágeis é de interesse demonstrar um artefato processual eficaz e possível dentre as diferentes cerimônias do processo SCRUM e que não é documentado para processo ágeis, mas como uma técnica exclusiva de modelos prescritivos.

O artigo está organizado em 6 seções, incluindo introdução e conclusão. A seção 2 é destinada a fundamentação teórica descrevendo os conceitos, técnicas e métodos de verificação e validação, bem como o gerenciamento de requisitos, utilizados nas metodologias tradicionais e ágeis, priorizando um entendimento do framework Scrum. A seção 3 apresenta a metodologia utilizada no estudo. A seção 4 apresenta os dados na sua abordagem qualitativa e quantitativa, bem como uma comparação da pesquisa com trabalhos já realizados referentes ao uso da técnica de prototipação. A seção 5 é destinada a compilação e apresentação dos resultados do trabalho.

2. Fundamentação Teórica

Para fins de fundamentação teórica é importante verificar e revisitar como funcionam os processos de tratamento de requisitos nas diferentes abordagens de desenvolvimento de software, diferenciadas principalmente dentre os métodos tradicionais e ágeis.

Também analisando a técnica de prototipação em seu contexto aplicado a modelos prescritivos, isoladamente e/ou em conjunto e verificar o seu comportamento no ambiente ágil comparando-o também com outras técnicas de forma a ter-se uma correlação dos benefícios e malefícios. Dar-se-á atenção também ao ambiente do teste ágil e tradicional, motivando o estudo da técnica de prototipação como auxiliadora no ganho da qualidade. O método Scrum será revisitado em um tópico particular em sua completude, verificando os seus processos e cerimônias, possibilitando posteriormente que se possa efetivar uma correlação dos benefícios da inserção da prototipação neste ambiente.

2.1. Modelos prescritivos de Desenvolvimento

Os processos tradicionais, prescritivos ou direcionados a um plano são metodologias de desenvolvimento originadas para sanar o caos encontrado no desenvolvimento de software em meados do século XX, sendo o primeiro destes modelos a ser documentado a modelo cascata, visto na Figura 1, no qual as fases são dependentes da conclusão da fase anterior para serem então iniciadas (PRESSMAN, 2011).

Este modelo tinha por caraterística uma alta dependência dos requisitos e não previa a mutabilidade que pode ser acometida nestes durante todo o processo de construção, o que poderia ser fatal para a aplicação.

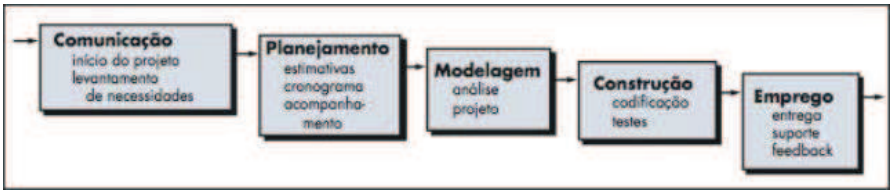


Figura 1. Representação do modelo Cascata

O levantamento dos requisitos, bem como a sua análise seriam imutáveis após a aprovação dos requisitos com o cliente e só viriam a ser implementadas mudanças necessárias na tarefa de suporte compreendida na fase de emprego, quando o cliente efetivamente teria um contato real com o software (SOMMERVILLE, 2011).

É uma variação comum da representação do modelo cascata, conforme representado na Figura 2, segundo Pressman (2011). O modelo "V", conforme Molinari (2012, p.118) "*descreve graficamente as fases individualmente em forma de "V. Neste caso, "V" vem de verificação e validação*".

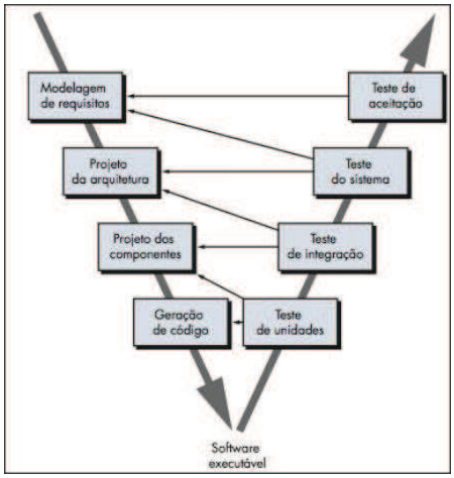


Figura 2. Representação do modelo "V"

Ressalta Pressman (2011) que em contrapartida aos modelos lineares de desenvolvimento foi criado o modelo incremental, que fornece suporte a projetos em que os requisitos iniciais estão consideravelmente definidos em parte, mas devido a questões de trabalho e necessidade de uma parte funcional pelo cliente o paradigma linear não é adequado, este modelo tem por características ter muita similaridade aos modelos evolucionários que serão abordados posteriormente.

Segundo Pressman (2011), o modelo incremental visto na Figura 3 pressupõe que cada conjunto de etapas do processo linear após concluída tenha a entrega de um incremento e este seja refinado com novas partes funcionais em cada novo ciclo. Pressman (2011) salienta que os primeiros incrementos forneçam características mais elementares da aplicação suprindo seus requisitos básicos os quais motivados com o uso do usuário tenderiam a ser refinados fornecendo assim a capacidade avaliativa ao cliente, bem como a possibilidade de mudanças dos requisitos subsequentes.

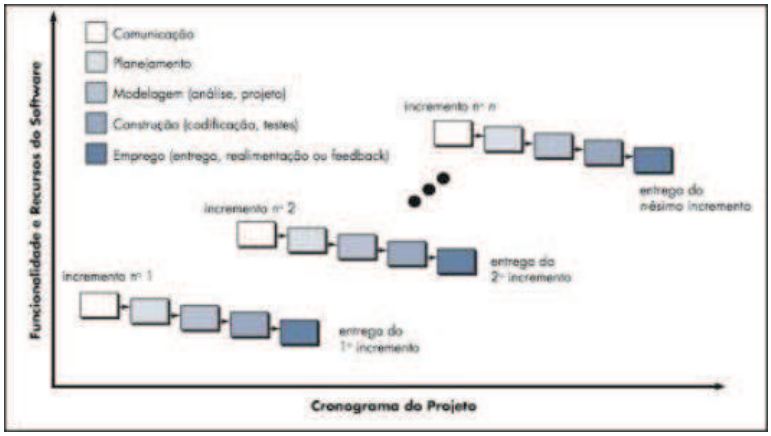


Figura 3. Representação do modelo incremental

Diz Sommerville (2011) que o modelo incremental não pressupõe a alteração de requisitos para aplicações que já estão sendo desenvolvidos, devido a caracterização de que seus processos de análise e elicitação já estejam concluídos, para tanto podem ser somente alterados nos incrementos posteriores nos quais novas alterações serão efetivadas. O autor também defende que o modelo tem por utilidade suprir os seguintes benefícios aos seus usuários:

1. Custos de acomodação das mudanças nos requisitos são minimizados, devido à baixa manutenibilidade da documentação e análise.
2. É mais fácil argumentar e levantar *feedbacks* de clientes de um artefato físico que de documentações.
3. O cliente desfruta da aplicação até então entregue (requisitos básicos) e pode usufruir monetariamente, justificando o investimento.

A prototipação segundo Pressman (2011) pode ser usada dentre as fases lineares de construção do incremento como formas de auxiliar o entendimento dos requisitos não clarificados e dúbios; Sommerville (2011) discorre também que não se pode confundir uma aplicação funcional com um protótipo, visto que os incrementos são tratados como partes reais do sistema.

São também compreendidos como metodologias de cunho tradicional, os modelos evolucionários que tem por característica suprir a necessidade de sistemas que tenham ao mínimo um conjunto de requisitos entendíveis, mas que ainda não conseguiram evidenciar a sua totalidade, ou seja, existem partes do sistema que não foram mapeadas ou não estão suficientemente clarificadas. Corrobora Pressman (2011) que este modelo tem por característica aceitar a natureza evolutiva dos projetos, bem como a sua mutabilidade negando a linearidade construtiva abordada até então.

São exemplos da construção evolutiva: o paradigma Espiral e a Prototipação. Segundo Pressman (2011) a prototipação é comumente utilizada não como um modelo, mas como uma técnica para a elucidação de requisitos, entretanto, o autor defende que a mesma possa ser empregada isoladamente tendo então, uma característica modular e construtiva sendo mais que uma técnica. A mesma é empregada quando o cliente oferece uma grande gama de objetivos do sistema, entretanto, não disponibiliza detalhadamente os requisitos destas funções e recursos. Conforme mostrado na Figura 4, estes artefatos conhecidos são refinados e traduzidos em um protótipo que será apresentado ao cliente de forma que o mesmo possa fornecer uma comunicação quanto ao mesmo, retroalimentando o sistema novamente (PRESSMAN, 2011).

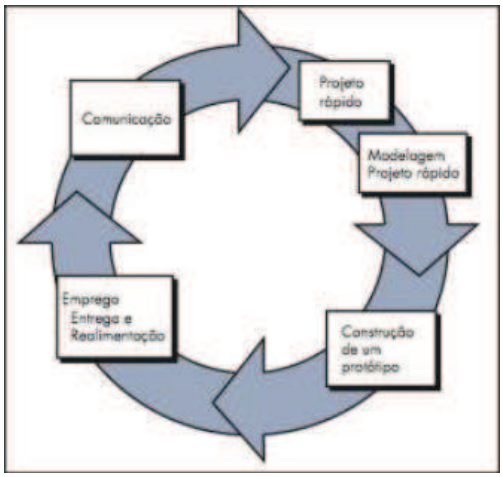


Figura 4. Modelo evolucionário - prototipação

É então possível segundo Pressman (2011) decidir quanto ao futuro desta aplicação no qual o autor menciona duas abordagens: a natureza "descartável", na qual o protótipo é totalmente negado, dando lugar a um novo, ou então a natureza "evolutiva", na qual o protótipo será reconstruído com as lições de forma a gerar novas iterações de protótipos ou até mesmo a aplicação real. O autor levanta, no entanto, desvantagens quanto ao uso do modelo:

1. Clientes podem se iludir com o protótipo e requisitarem poucas mudanças no mesmo para que tenham uma aplicação útil o mais rápido possível, porém com a mínima qualidade e não tendo atenção a características globais do mesmo.
2. Os desenvolvedores podem assumir compromissos de implementação irreais para a entrega de uma aplicação funcional rápida e que não condizem com a natureza da aplicação, não levando em conta requisitos futuros não elucidados o que pode inserir uma falha na aplicação proveniente de um protótipo anterior não explorado corretamente.

Defende Pressman (2011), entretanto, que não se pode retirar a contribuição do uso da prototipação como um processo de melhoria inegável da qualidade.

Outra abordagem do paradigma evolutivo é o modelo Espiral, figura 5, comumente direcionado a projetos de mitigação de risco, o mesmo foi proposto por BOEHM (1988). Fala Sommerville (2011, p. 33) que "*a principal diferença entre o modelo espiral e outros modelos de processo de software é seu reconhecimento explícito do risco. "*

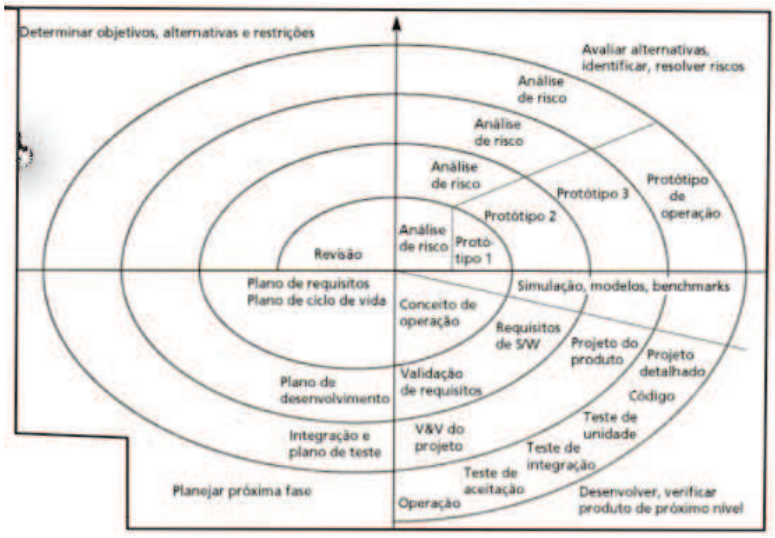


Figura 5. Modelo evolucionário - Espiral

A cada novo ciclo são identificados riscos e analisado o seu impacto. Posteriormente estes serão mitigados através do uso combinado de técnicas de prototipação, simulação ou análise mais profunda do mesmo conforme a externalização das voltas na espiral, está por sua vez dividida em 4 setores: Definição de objetivos, avaliação de riscos, desenvolvimento/validação e planejamento. O autor destaca o suporte as mudanças como o maior atributo, encarando as mesmas como resultado de riscos do sistema fornecendo atividades próprias para a solução.

O *Rational Unified Process* (RUP) é um dos mais modernos modelos tradicionais propostos, a parte das metodologias ágeis. Segundo Sommerville (2011) foi proposto em 2003 por Krutchen, unindo ideias de todos os modelos previamente citados em uma única abordagem, tendo características iterativas e incrementais, assim como fazendo uso da prototipação para elucidação de requisitos. Pode-se destacar pelo autor, a presença das seguintes boas práticas presentes no RUP:

- Gerenciamento de requisitos; acompanhando e documentando as mudanças;
- Modelar o sistema utilizando da UML;
- Atenção na qualidade de software;
- Controlar as mudanças usando sistema e ferramentas para isso;
- Desenvolver iterativamente, priorizando incrementos.

Mediante o ressaltado acima, os modelos prescritivos encarados como opositores das metodologias ágeis utilizam-se da prototipação em vários modelos, cita-se o modelo incremental e os modelos evolucionários (Espiral, prototipação e RUP), melhorando consequente a qualidade dos requisitos e da aplicação final ofertada.

A gestão dos requisitos nestes modelos tem fases bem delimitadas e exclusivas, as quais podemos citar: Especificação, Elicitação e Análise, Validação e Gerenciamento. Estas fases conforme Pressman (2011) ditam limites e tempo para a execução destas etapas, criando uma diferença quanto aos modelos ágeis nos quais as mudanças nos requisitos são realizadas durante o desenvolvimento do software, facilitadas pela não

necessidade de prévio estudo de impacto, atualização de documentos, análise de custos ou autorizações complementares.

2.2. Engenharia de Requisitos "Tradicional"

Argumenta Pressman (2011) que Engenharia de Requisitos é uma atividade da Engenharia de software iniciada na fase de comunicação e seguindo na fase de modelagem, sendo uma das etapas mais importantes do processo de construção devido aos reflexos impactantes tanto em custos como em visibilidade, em todas as fases de desenvolvimento posteriores e por se tratar da fase inicial do processo de construção.

O autor também menciona que a mesma compreende todas as tarefas e técnicas que levam ao entendimento dos requisitos. Estes últimos, confirma Sommerville (2011, p. 57) são "as descrições do que o sistema deve fazer, os serviços que oferecem e as restrições a seu funcionamento".

Ressalta Sbrocco (2012) que os requisitos podem ser taxonomizados, conforme a perspectiva em que são vistos, para tanto o mesmo pressupõe a existência de duas esferas a de usuário e de desenvolvedor. Na esfera do usuário pode-se segundo o autor haver as seguintes distinções:

- Requisitos de negócio: representam as necessidades que existem no negócio e que são acionadas pelos usuários. O autor salienta cuidado quanto a não inclusão de aspectos técnicos, pois o alvo é entender o que deverá ser feito e não como será.
- Requisitos do usuário: são funções que o sistema deve oferecer a seus usuários e as restrições em que deve operar.
- Requisitos do sistema: são descrições detalhadas dos serviços, funções e restrições operacionais do sistema de software.

Na esfera do desenvolvedor Sbrocco (2012) salienta as seguintes esferas:

- Requisitos funcionais: são descrições das funções do sistema deve fornecer ao usuário representando as regras de negócio, as funcionalidades que o sistema deve fornecer. Podem também compreender o que o sistema não deve fazer segundo Sommerville (2011)
- Requisitos não funcionais: são restrições impostas pelas funções ofertadas pelo sistema. Podem ser divididos em:
    - Requisitos do produto: requisitos ligados ao comportamento do sistema, tais como desempenho, confiança e segurança.
    - Requisitos organizacionais: são provenientes de políticas e procedimentos da organização;
    - Requisitos externos: são advindos quando se observa o ambiente em que será executado o sistema.

A Engenharia de Requisitos conforme Pressman (2011) engloba sete tarefas distintas para o tratamento eficaz destes requisitos: a concepção, levantamento de requisitos, elaboração, negociação, especificação, validação e gestão de requisitos. Esta última sendo o grande diferencial, quando vista a diferenciação ágil e tradicional.

A concepção é a fase da Engenharia de Requisitos responsável pelo entendimento do problema, as pessoas que querem a resolução deste problema, a natureza da solução pretendida e a eficácia da comunicação entre os *stakeholders*.

A fase de levantamento tem por objetivo extrair dos usuários, cliente e interessados quais são os objetivos que devem ser alcançados com a construção do sistema, onde se objetiva chegar, o que deve ser feito. Sommerville (2011) menciona uma série de artifícios que podem ser utilizados no intendo de extrair os requisitos dos usuários dentre eles pode-se mencionar as entrevistas formais e informais, criação de cenários reais de uso, casos de uso, prototipação, brainstorming, entre outras.

A fase de elaboração é destinada a refinar os cenários de usuários obtidas durante as fases de concepção e elaboração mapeando as relações entre as entidades de domínio do sistema.

A fase de negociação é usada para conciliar requisitos que não poderão ser alcançados devido a limitações próprias do sistema, mudança de requisitos que são conflitantes, e /ou eliminação de requisitos duplicados ou desnecessários.

A fase de especificação é o translado físico de todos os aspectos dos requisitos no qual também se faz uso de artefatos gráficos, entrevistas, cenários de caso de uso, prototipação e casos de uso, bem como diagramas UML.

A fase de validação corresponde a análise dos artefatos produzidos durante a especificação fazendo a verificação das mesmas, nesta etapa podem ser realizadas revisões dos documentos produzidos, ou solicitar-se a sua modificação. Segundo Sommerville (2011) podem se usar artifícios de revisões através da prototipação, geração de casos de teste e verificações reais dos requisitos levando-se em conta o conhecimento do usuário.

A fase de gestão de requisitos entende que os sistemas mudam conforme o tempo, assim como os desejos dos usuários e as tecnologias aos quais são submetidos. Sommerville (2011), esboça 3 razões para estas mudanças:

1. O ambiente técnico, de negócios do sistema muda após a instalação. Ou seja, tecnologias novas podem ser implementadas, novas prioridades e ou legislações.
2. O usuário final do sistema muda e consequentemente novos desejos e requisições são formadas.
3. O usuário final entra em conflitos existentes entre diversas nuances havendo a necessidade também da mudança dos requisitos.

Sommerville (2011) elicita três estágios, que devem ser seguidos para o gerenciamento da mudança de requisitos:

1. Análise do problema e especificação da mudança: Um problema é encontrado em um requisito os interessados devem analisar a mudança e se é realmente necessária afim de determinar a sua validade. Munido das informações passa ao requisitante para que o mesmo aprove ou não a mudança.
2. Análise de mudanças e custos: O efeito da mudança é medido com base nas informações existentes como matriz de rastreabilidade e conhecimentos do sistema. O custo desta mudança é medido e determina-se a execução ou não da mudança.

3. Implementação da mudança: O documento de requisitos, o projeto e a implementação do sistema são alterados e modificados.

Segundo Sommerville (2011) devem ser evitados alterar requisitos e partes do sistema sem que se faça previamente a atualização da base de requisitos, pois muitas partes podem ser esquecidas o que pode tornar o documento defasado.

Quanto as etapas acimas citadas menciona Sommerville (2011, p. 58) que:

> "Para a maioria dos sistemas de grande porte, ainda é o caso de existir uma fase de engenharia de requisitos claramente identificável antes de se iniciar a implementação do sistema. Certamente, haverá mudanças nos requisitos e os requisitos de usuário poderão ser ampliados em requisitos de sistema mais detalhados. No entanto, a abordagem ágil de simultaneamente elicitar os requisitos enquanto o sistema é desenvolvido é raramente utilizada para desenvolvimento de sistemas de grande porte. "

Então, conforme o autor, os modelos dirigidos a planos tem por características possibilitar que os requisitos possam ser mais trabalhados, motivados principalmente pela presença de fases mais definidas, planejadas e delimitadas dentre o processo de construção de software fazendo com que ao final de cada fase a posterior venha a ser retroalimentada com as saídas da anterior, possibilitando a continuação do ciclos, primando também pela documentação destas saídas que são a comunicação entre os estágios. Em contrapartida as metodologias ágeis não utilizam destes processos devido a limitação de tempo, ausência de documentação e uso de uma comunicação real entre todas as fases de construção.

2.3. Tradicional X Ágil

Na atualidade, em um mercado global, novas necessidade e oportunidades de negócio despontam a cada dia, as necessidades dos usuários finais mudam, as legislações mudam, as condições políticas e econômicas mudam e isto muito rapidamente. E todas estas operações envolvem pessoas e negócios que utilizam algum software capaz de tornar suas ações possíveis.

Salienta Sommerville (2011) que estes softwares ou serão substituídos por novos que serão criados para atender essas novas exigências ou serão adaptados de forma a satisfazê-las e tudo isso rapidamente devido à natureza competitiva presente nas aplicações. O autor também defende que estes ambientes têm por característica requisitos instáveis, portanto, seguir uma abordagem dirigida a planos não seria uma solução plausível, visto que ao serem implementados esses sistemas certamente apresentariam problemas em seus requisitos, não satisfazendo mais as necessidades que deveriam satisfazer, tornando-os falhos, pois não levaram em conta o aspecto da mudança (resultado da flutuação nos âmbitos do negócio). Neste fato confirma Pressman (2011, p. 82) que "fluidez implica mudanças, e mudanças são caras. Particularmente se forem sem controle e mal gerenciadas".

Pressupõe-se então que para tratar esta mutabilidade presente nos requisitos, apoiada também pela rapidez na criação de novas aplicações, que surgiram as metodologias ágeis, advindas originalmente do famoso manifesto ágil criado em 2001, a partir de uma reunião entre renomados profissionais de software da época (Alistair Cockburn, Jeff Sutherland, Dave Thomas, Kent Beck, Martin Fowler, entre outros) e que já vinham trabalhando com métodos inovadores, insatisfeitos com os chamados

métodos “pesados” existentes até a época. Relata Pressman (2011, p. 83) quanto à natureza dos princípios que regem grande parte das metodologias ágeis:

> “Ela incentiva a estruturação e as atitudes em equipe que tornam a comunicação mais fácil (entre membros da equipe, entre o pessoal ligado à tecnologia e o pessoal da área comercial, entre os engenheiros de software e seus gerentes). Enfatiza a entrega rápida do software operacional e diminui a importância dos artefatos intermediários (nem sempre um bom negócio); assume o cliente como parte da equipe de desenvolvimento e trabalha para eliminara atitude de “nós e eles”, que continua a invadir muitos projetos de software; reconhece que o planejamento em um mundo incerto tem seus limites e que o plano (roteiro) de projeto deve ser flexível”

Sommerville (2011) ressalta pontos acerca das diferenças entre as metodologias tradicionais e as ágeis. Como visto na Figura 6, as metodologias ágeis têm por característica englobar elicitação de requisitos e testes juntamente no projeto e na implementação havendo uma comunicação constante entre as fases, ou seja, todas as etapas do processo de desenvolvimento fornecem subsídios para todas as etapas iterativamente sem que haja um elo isolante entre as mesmas.

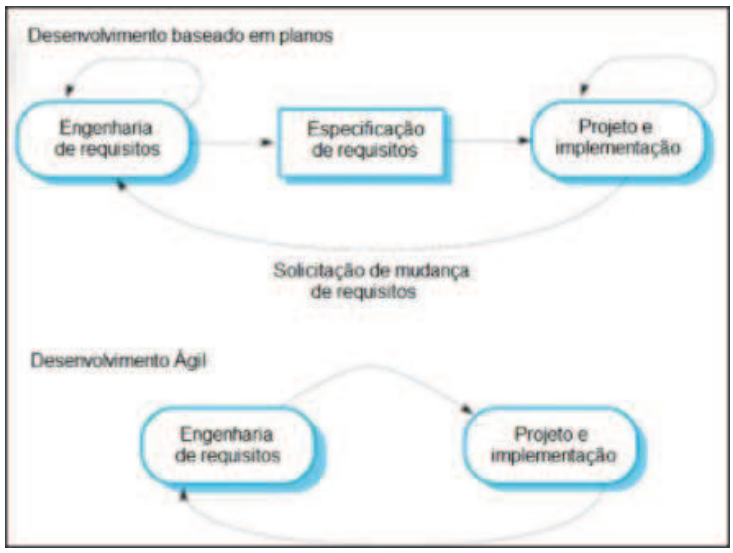


Figura 6. Tradicional x Ágil

Já no paradigma dirigido a planos, estas saídas e entradas são unidirecionais retroalimentando as fases posteriores para o seu prosseguimento. Outro fator são que devido a realidade “isolada e distinta” de cada fase, no processo dirigido a planos as saídas são convertidas em documentação, que servem de comunicação e entrada para a fase posterior.

Salientam Sommerville (2011) e Pressman (2011) que as metodologias tradicionais são altamente recomendáveis para sistemas que:

- Necessitam de um alto detalhamento do projeto e especificação antes de irem para a implementação;
- Pretendem trabalhar com equipes de desenvolvimento grandes;
- Nível de habilidade baixo da equipe (Abordagem dirigida a planos pressupõe o uso da documentação para suporte a equipes com baixo nível de habilidade);

- Necessitem de documentação específica definida por regulamentação externa cita-se o ramo aeroespacial, automobilístico e aeronáutico.

Defende Sommerville (2011) que indiferentemente da abordagem escolhida os compradores destes sistemas não estão interessados na abordagem de construção, mas sim em ter uma aplicação executável, que atenda a seus requisitos e que seja útil para os usuários individuais e/ou vistos como organização.

2.4. SCRUM

Foi proposto em 1995 no artigo "*Scrum and the perfect storm*", contendo o resumo de suas experiências e aprendizados, a pedido da empresa *Object Management Group* (OMG) em conjunto por Jeff Sutherland e Ken Schwaber (PHAM, 2011).

Conforme Schwaber (2009), o *Scrum* não é um processo padronizado, com etapas metódicas e rigorosas e que na certeza de execução vá gerar garantia de resultados, mas é um conjunto de valores, princípios e práticas que fornecem base para que uma organização de forma pessoal possa moldar a seus próprios fins, criando assim a sua própria versão.

Na abstração impessoal podemos imaginar que o *Scrum*, conforme o autor, seja um prédio, os valores, prática e princípios seriam as fundações e as paredes, que não podem ser alteradas, mas em via de regras toda a parte restante é alterada, criando a sua própria abstração e se tornando impessoal, ou seja cada pessoa monta o seu prédio da forma que almejar, porém as práticas de fundação são as mesmas.

Conforme Sbrocco (2014), o *Scrum* pode ser dividido em 4 grandes fundamentos:

1. **Papéis:** que nada mais são as pessoas (*Product Owner, Scrum Master* e *Team*) que estão envolvidas no processo bem como seus papéis correspondentes;
2. **Artefatos:** São todos os artigos produzidos pelo *Scrum* durante o seu processo através das reuniões (*Product Backlog*, *Sprint Backlog*, incremento, objetivo do sprint, lista de bloqueios);
3. **Reuniões:** São uma das estruturas mais importantes dentro do *Scrum*, pois modelam e dão continuidade ao processo (*Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Retrospective*);
4. **O processo:** Que nada mais é que a união de todos os pontos como reuniões, pessoas, artefatos para a entrega a entrega dos documentos, dentro das regras previstas.

Para Schwaber (2009), os integrantes e participantes do *Scrum* podem ser divididos em partes quanto à sua participação no processo de entrega do incremento. Segundo o autor, o *Scrum* é composto por equipes basicamente formadas por um corpo único, com três papéis bem definidos, podendo existir em uma única corporação diversas equipes com a mesma estrutura, trabalhando de forma independente das demais, supõe-se que não passem de 9 pessoas por grupos, para manter uma comunicação eficiente, onde temos: o *Scrum Master*, o Proprietário do Produto, ou *Product Owner* e a Equipe, ou *Team*.

O *Product Owner segundo* Schwaber (2009) é o dono do produto ou projeto sendo responsável por ditar as regras e direção a seguir para que se faça a entrega, definindo os requisitos que vão formar o *product backlog* e qual a prioridade que deve ser abordado pelo time. O time por usa vez deve obedecer somente às regras e prioridades impostas pelo *Product Owner*. Cabe a ele conversar com as áreas pertinentes para decidir que abordagem tomar quanto ao incremento, seja para alterações quanto para exclusões, podendo o time levar subsídios e opiniões, mas nunca serem aptos a decidir por si mesmos.

Ainda, segundo o autor, o *Scrum Master* é o elemento que faz a interação entre o *Product Owner* e a equipe tem a responsabilidade de gerenciar, organizar e pleitear as reuniões, cuidar e assegurar que o processo do *Scrum* seja seguido, acompanhar o trabalho e certificar que a equipe está tendo as devidas condições, ferramentas e utensílios para trabalharem adequadamente, destroçando as barreiras que possam impedir o andamento saudável do processo. Outra tarefa atribuída ao *Scrum Master* é a de manter o foco da equipe de forma a atingir os máximos resultados possíveis.

O Time é a equipe que trabalha para o desenvolvimento do projeto e produto (analisar, projetar, desenvolver, testar técnicas de comunicação, documentos, etc.), recomenda-se que seja uma equipe auto organizada e auto-conduzida. Segundo Schwaber (2009) dentro do time não deve haver menção de títulos e nem interesses voltados para desenvolvimento próprio, mas sim um compartilhamento de conhecimentos entre as partes mais especializadas para que o incremento possa ser entregue.

Outro fator, segundo o autor, que deve ser visto são a auto-organização do time, pois ninguém deve dizer como fazer para transformar os requisitos em funcionalidades, nem mesmo o *Scrum Master*, o time dentre sua capacidade e interação deve encontrar o caminho para este translado.

É interesse que por questão de organização e gerenciamento, os times não ultrapassem o limite de 9 pessoas e que também não sejam inferiores a 5 pessoas, pois há uma menor interação, o que acarreta em menos conhecimento sendo trocado e possivelmente assim menos produtividade.

Argumenta Schwaber (2009, p. 16), que o *Scrum* utiliza artefatos para posicionamento dos incrementos dentro do processo: *"...* o *Backlog* do Produto, o *Burndown* da Versão para Entrega, o *Backlog* da *Sprint* e o *Burndown* da *Sprint*".

O *Product Backlog* é uma lista priorizada de tudo que pode ser necessário para lançamento do produto, nunca está completo, mostra somente uma versão inicial do que são os requisitos, é dinâmico, pois muda constantemente no sentido de estar em concordância com o que o produto necessita. Argumenta Schwaber (2009, p. 17) que "enquanto existir um produto, o *Backlog* do produto também existirá".

Em seu artigo publicado na *Scrum Alliance*, Borghi (2015) ressalta a importância do uso de técnicas de elicitação de requisitos combinadas, aplicadas à metodologia *Scrum*, no início da construção do *Product Backlog*, conforme Quadro 1.

Técnicas Tradicionais	**Técnicas Colaborativas**
Entrevistas	Protótipos

Cenários	Reuniões facilitadas
Observação	Estória de usuário
Análise de documentos	*Brainstorming*
Análise de interfaces	Grupo Focal
Engenharia Reversa	*Workshop* de Requerimentos
Pesquisas e questionários	

Quadro 1. Técnicas para elucidação de requisitos

Borghi (2015) menciona também que é importante o uso de técnicas combinadas levando-se em conta o conhecimento dos *Stakeholder*, a criticidade dos requisitos e a complexidade dos incrementos. O autor dá maior importância no seu monólogo as técnicas de prototipação, *brainstorming* e *workshop* de requerimentos, levantando a destaque os seguintes pontos, vistos no Quadro 2.

Benefícios	**Malefícios**
Protótipos descartáveis são rápidos e baratos	Número muito grande de participantes podem atrapalhar a coleta de dados
Eficácia das seções de brainstorm e workshop	Os protótipos podem iludir os usuários e dar falsa realidade
Feedbacks imediatos	O conhecimento dos participantes é limitado e as reuniões de brainstorm se tornam não frutíferas
Protótipos dão visibilidade das tecnologias e lacunas	
Imagens projetam o futuro do sistema	

Quadro 2. Benefícios e malefícios do uso das técnicas supracitadas

É importante notar que estas técnicas são usadas em requisitos do *Product Backlog* e não no *Sprint Backlog,* devido ao argumento de Sbrocco (2014) de que somente estórias de usuário com requisitos básicos claros possam entrar no *sprint* corrente, e que incrementos falhos deveriam ser reenviados para o *Product Backlog* para a correta elucidação. Não é aceitável, segundo o autor, que incrementos tenham seus requisitos alterados dentro do *sprint* corrente, após as reuniões de *Sprint Planning*.

As histórias de usuário do *backlog* possuem atributos, tais como: descrição, prioridade e estimativa, sendo que a prioridade é determinada pelo risco, valor e necessidade definidos pelo *Product Owner*. Quanto mais alta a prioridade delegada à história do usuário dentro da lista do *Product Backlog*, mais detalhadas estão as suas informações, bem como seus requisitos, pois estão em ordem imediata para trabalho.

O *Sprint Backlog* é uma lista de histórias de usuário, retiradas do *Product Backlog*, para uma *Sprint* (período de tempo), representando um ou mais incrementos do produto potencialmente entregável, somente estórias de usuário que tenham sido elucidadas adequadamente pelo Time e *Scrum Master* podem ser trazidas para o *Sprint Backlog*.

Durante a reunião de planejamento da *Sprint* (*Sprint Planning*), o *Product Owner* prioriza os itens do *Product Backlog* e os descreve para a equipe. Então, a equipe se encarrega de argumentar o que poderá completar durante o *Sprint* que vai começar. Conforme Schwaber (2009), o *Sprint Backlog*, através do quadro *Kanban (*Figura 7*)* é a lista de tarefas que o Time se compromete a fazer durante o *Sprint*. Os itens do *Sprint Backlog* são extraídos diretamente do *Product Backlog*, pela equipe com base na priorização definida pelo *Product Owner* e a percepção da equipe sobre o tempo que é necessário para completar as várias funcionalidades.

Durante o *Sprint* corrente, o *Scrum Master* mantém o *Sprint Backlog* atualizado para refletir de forma apurada e correta o andamento das tarefas que são completadas (*Done*), o que está em progresso (*In Progress*), o que resta e tem-se a fazer (*To do*) e os itens pais (*Stories*) assim como o tempo gasto para cada uma. É comum dividir as *User Stories* em tempo e tarefas (*Tasks*) para melhor visualização do tempo transcorrido, tudo isso alimenta diariamente a mudança do *Sprint Burndown Chart*.

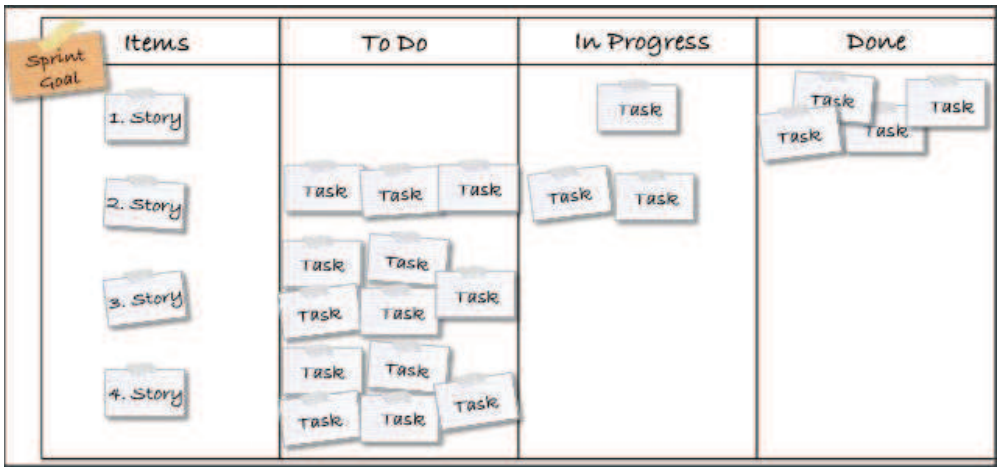


Figura 7 – *Sprint Backlog*/Quadro *Kanban*

Conforme Schwaber (2009), o *Sprint Burndown Chart* (Figura 8) é a ferramenta que monitora o andamento do *Sprint*, consiste em uma tabela que faz o mapeamento do que foi feito e o que precisa ser feito a tempo do final do *Sprint*, assim como visualizar como ocorreu a interação do *Sprint*. O eixo vertical mostra a quantidade de trabalho que ainda precisa ser feita no início de cada *Sprint*. O trabalho que ainda resta pode ser mostrado na unidade preferencial da equipe: Pontos de estória (*Story Points*), dias (*team days*) e assim por diante.

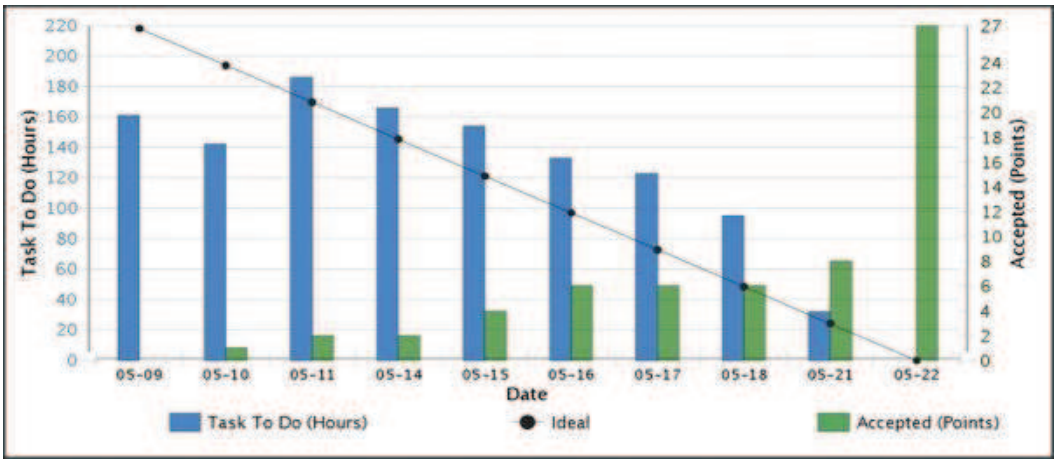


Figura 8 – *Sprint Burndown Chart*

O *Sprint* é a medida do tempo de trabalho do *Scrum*, podendo ser medido de 2 a 4 semanas sendo programados para que sempre tenham uma data de início e fim fixa e com mesma duração, seguindo um novo *Sprint* ao final do outro. Ressalta Sbrocco (2014) que é no tempo da *Sprint* que o produto deverá ter seus requisitos negociados e o produto projetado, codificado e testado, com o fim de uma *sprint* novamente esses processos são reexecutados na subsequente. O autor ressalta, no entanto, que todas as etapas acontecem obedecendo o fluxo das cerimônias existentes.

Outra questão importante no *Scrum* é a determinação do estado de "pronto" que segundo Pham (2014, p. 133) significa "*pronto para implantar no ambiente de produção*". O autor argumenta três estados dentro de sua experiência, em que o pronto possa ocorrer:

1. Requisitos>Projeto> Codificação e Teste unitário então "pronto": Neste estilo de pronto o autor ressalta que devidos a várias equipes estarem trabalhando com o Sprint é mais rentável declarar cada incremento pronto individualmente, porém o autor alerta a necessidade de que os testes e a integração tenham suas etapas cumpridas.

2. Requisitos>Projeto> Codificação e Teste unitário> Testes de sistema/Integração: Neste somente após todas as estórias de usuário terem sido concluídas é que o produto pode der declarado como pronto.

3. Requisitos>Projeto> Codificação e Teste unitário> Testes de sistema/Integração> Testes de aceitação do usuário: Este segundo PHAN (2014) é o cenário ideal, no qual usuários do negócio fazem parte da equipe e conseguem executar os testes de aceitação antes da entrega, neste caso o pronto somente é dado após está fase e os incrementos podem ser destinados as iterações de release.

Detalha-se a seguir um pouco do entendimento das cerimônias realizadas durante o *Sprint* (Eventos com duração fixa) e antes dele que moldam o processo *Scrum* e o fazem acontecer, são como as artérias para o coração aqui representado como a *Sprint*, são divididas fundamentalmente em: *Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Sprint Retrospective*.

Segundo Schwaber (2009), a reunião de *Sprint Planning* (planejamento da Sprint) é quando a Sprint é planejada, de forma a ocupar pelo menos 5% do tamanho total do *Sprint* (para 1 mês 8hs, para 2 semanas 4 horas) que é dividida em duas etapas, a primeira com duração fixa e tempo de 4 – 2 horas é quando será decidido o que será feito no *Sprint*, provendo um diálogo para priorizar o *Product Backlog*.

A segunda parte também definida igualmente a primeira parte em tempo serve para visualizar como a equipe desenvolverá a funcionalidade para o *Sprint*, decompondo as atividades em forma de tarefas menores, estimando horas e votando pesos para poder organizar melhor as atividades e melhorar o aproveitamento do tempo que será utilizado (reunião conhecida como *Planning Poker*), nesta reunião é possível ter a presença do *Product Owner* para suprir dúvidas necessárias, a partir daqui se monta o *Sprint Backlog*. O Time pode decidir que não consiga entregar a tempos as atividades, para tanto deve negociar com o *Product Owner* as prioridades.

O *Daily Scrum* (Reunião Diária) é a reunião realizada diariamente, idealmente no mesmo horário, preferencialmente atingindo no máximo 15 minutos e com todos os participantes de pé. É uma reunião muito importante, pois melhora a comunicação, elimina reuniões desnecessárias, identifica e remove impedimentos, ressalta e promove a tomada de decisão e melhora o nível de conhecimento da equipe em relação ao projeto, nesta reunião somente são permitidos a equipe.

A reunião diária é uma inspeção do progresso na direção da entrega da meta, materializadas pelas respostas das três questões fundamentais, segundo Schwaber (2009, p. 15):

> "O que fiz ontem que ajudou o time a atingir a meta do Sprint?
> O que vou fazer hoje para ajudar o time a atingir a meta do Sprint?
> Existe algum impedimento que não permita a mim ou ao time atingir a meta do Sprint? "

O *Sprint Review (Revisão da Sprint)*, segundo o autor é uma das reuniões que acontece ao final do *Sprint*, tem como participantes todos os envolvidos e interessados com os incrementos, geralmente utiliza-se demonstrações funcionais dos incrementos em ação, esta reunião destina-se a motivar a equipe, obter comentários acerca do que se está finalizando, promover a colaboração entre todos, alinha o *Product Backlog* para as novas Sprints. Idealmente tarefas que não forem completadas não devem ser mostradas.

Confirma Sbrocco (2014), que esta etapa tende a ser a etapa pertinente aos testes de aceitação das metodologias tradicionais havendo-se a possibilidade que ao final da apresentação o *Product Owner* decida não aceitar o incremento, neste caso as mudanças necessárias devem ser traduzidas em novas estórias de usuários e incluídas no *Product Backlog* para priorização. É interessante verificar neste ponto que os incrementos não aceitos podem tomar a perspectiva de protótipos "descartáveis" ou "evolutivos", no entanto gerando novos ciclos de trabalho comprovando a existência de retrabalho.

O *Sprint Retrospective* tem como objetivo avaliar a *Sprint* anterior com o intuito de verificar as necessidades de adaptações no produto e no processo de trabalho, ocorre após o *Sprint Review*, geralmente, compreende a participação do *Scrum Master* e a equipe. Idealmente nesta reunião se responde também as seguintes perguntas: O que deu certo? O que deu errado? e O que pode ser melhorado?

Observa-se que *Scrum* é uma metodologia importante para empresas que almejam ter um controle de seus processos, com uma abordagem iterativa e incremental para otimizar, controlar e prever riscos além de propiciar uma inspeção e adaptação constante.

2.5. Prototipação na Qualidade de Software

Relata Sbrocco (2014) e Pressman (2011) que o uso da prototipagem como artefato de elucidação de requisitos é iniciado com um levantamento das funcionalidades básicos que o sistema deve oferecer com os usuários, após a equipe desenvolvimento se empenha em criar uma aplicação denominada protótipo, sendo este de baixa ou alta fidelidade. Protótipos de baixa fidelidade podem ser expressos por rabiscos em papel, um desenho no quadro negro ou até mesmo um desenho no computador, já representações de alto nível são destinadas a mostrar as relações, páginas da aplicação codificada ou uma aplicação realmente funcional ao usuário.

Estes protótipos, segundo Sbrocco (2014) serão refinados após uma demonstração com os usuários que adicionaram novos requisitos e/ou eliminarão outros, em sequência estas mudanças serão adicionadas a este protótipo ou um novo protótipo será criado para satisfazer os usuários, retroalimentando novamente o ciclo. Esta natureza segundo Pressman (2011) define dois tipos de prototipação (descartável e evolutiva) como vista no capitulo 2.1 que demonstrava a prototipação como metodologia.

A prototipação descartável leva em consideração que o protótipo após a fase de engenharia de requisitos será descartado dando sequência a sua real implementação. A prototipação evolutiva prega que partes deste modelo serão reutilizados ou mesmo toda a aplicação será reusada e que novos incrementos serão adicionados para satisfazer os requisitos faltantes. Recomenda Sbrocco (2014) que não é aconselhável reutilizar " o primeiro sistema" criado, ao contrário o autor recomenda o seu descarte.

O autor menciona que a prototipação atual evoluiu de forma a ser dividida em *wireframes*, *mockup e* protótipo, cadê um com objetivos distintos. Nisto confirma Zemel (2011) a respeito destes modelos:

- *Wireframes*: São abstrações simples em papel e/ou em desenho computacional da estrutura e componente de uma página web. Demonstram ações estáticas no modelo representando uma primeira abordagem no processo de design e coleta de requisitos;
- Modelos: Focam na realidade visual do design do site. São idênticos ou semelhantes ao produto final, porém não evidenciam dinamismo entre telas tendo característica estática.
- Protótipos: são representações dinâmicas, semi-funcionais e servem para dar uma amostra da real funcionalidade da aplicação, podem ou não incluir elementos de design finalizado.

Segundo Zemel (2011), estes modelos seguem a mesma realidade modular apresentada na prototipação como metodologia, devido a perspectiva de que todos são sinônimos da prototipagem.

A prototipação segundo Sommerville (2011) também é uma ferramenta usual na validação dos requisitos, não somente para a coleta. Os protótipos segundo o autor são usados em conjunto em várias metodologias de forma a apoiar o processo de elicitação de requisitos, podemos citar o *Rational Unified Process* (RUP), o paradigma Espiral e a prototipação como metodologia (incremental e iterativa).

Não obstante, a prototipação conforme Borghi (2015) é usada nas metodologias ágeis, cita-se o *Scrum* para a criação do *Product Backlog*, também com o intuito de elucidação dos requisitos. Entretanto, a técnica de prototipagem por ser altamente moldável pode ser inclusa em outras etapas do desenvolvimento de software.

No entanto qual a real contribuição da técnica de prototipação para a qualidade dos produtos de software, levando-se em conta aspectos implícitos na interação homem-máquina? Nisto confirma Pressman (2011) que estes benefícios podem ser encontrados quando analisados os fatores básicos da qualidade, através destas interações.

Estes fatores podem ser categorizados, conforme (PRESSMAN,2011 apud MCCAL, 1977, p. 362) em três aspectos: *as características operacionais, as habilidades de suportar mudanças e a adaptabilidade a ambientes.*

Figura 9 – Fatores da qualidade de MCCALL

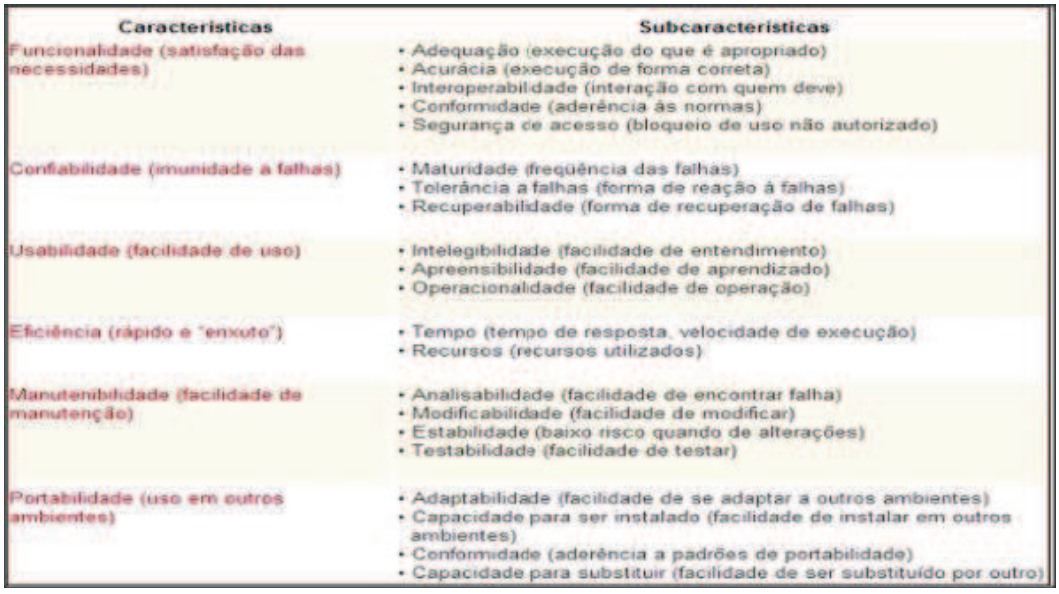
Com base na Figura 9 pode-se verificar a presença dos seguintes fatores, relata Pressman (2011, p. 363):

> 1.Correção: o quanto o programa satisfaz a sua especificação e atende ao cliente.
> 2.Confiabilidade: Refere-se ao quanto pode se esperar que o programa possa executar determinada tarefa.
> 3.Eficiência: A quantidade de código e recursos que são usados pela aplicação para sua função.
> 4.Integridade: Quanto ao acesso ao controle do software;
> 5.Usabilidade: Esforço condizente a interação homem-máquina;
> 6.Facilidade de manutenção: Esforço de correção dos problemas do sistema;
> 7.Flexibilidade: Esforço para modificar um sistema em operação;
> 8.Testabilidade: Esforço necessário para testar o programa;
> 9.Portabilidade: Esforço para transferência do programa aplicação entre hardwares/softwares;
> 10.Reusabilidade: O quanto o programa pode ser reutilizado;

11.Interoperabilidade: Esforço necessário para integrar um sistema a outro.

Segundo Pressman (2011, p. 363) é possível através das métricas de MCCAL (1977) definir uma " *sólida indicação da qualidade de software".*

O conjunto de normas ISO/IEC 25000 – SQuaRE, segundo ISO 25010 – Modelo da qualidade (2012) identifica os atributos fundamentais da qualidade de *software*. Na qual se tem uma visão geral do que trata cada atributo delegado à qualidade interna e externa do produto de *software*.



Características	Subcaracterísticas
Funcionalidade (satisfação das necessidades)	• Adequação (execução do que é apropriado) • Acurácia (execução de forma correta) • Interoperabilidade (interação com quem deve) • Conformidade (aderência às normas) • Segurança de acesso (bloqueio de uso não autorizado)
Confiabilidade (imunidade a falhas)	• Maturidade (freqüência das falhas) • Tolerância a falhas (forma de reação à falhas) • Recuperabilidade (forma de recuperação de falhas)
Usabilidade (facilidade de uso)	• Intelegibilidade (facilidade de entendimento) • Apreensibilidade (facilidade de aprendizado) • Operacionalidade (facilidade de operação)
Eficiência (rápido e "enxuto")	• Tempo (tempo de resposta, velocidade de execução) • Recursos (recursos utilizados)
Manutenibilidade (facilidade de manutenção)	• Analisabilidade (facilidade de encontrar falha) • Modificabilidade (facilidade de modificar) • Estabilidade (baixo risco quando de alterações) • Testabilidade (facilidade de testar)
Portabilidade (uso em outros ambientes)	• Adaptabilidade (facilidade de se adaptar a outros ambientes) • Capacidade para ser instalado (facilidade de instalar em outros ambientes) • Conformidade (aderência a padrões de portabilidade) • Capacidade para substituir (facilidade de ser substituído por outro)

Figura 10 – ISO/IEC 25000 – SQuaRE

Junto à Figura 10, têm-se ainda mais quatro fatores contidos na ISO 25024 – Qualidade em uso (2012) integrantes da ISO 25000, que reflete características observáveis próprias da qualidade de uso do produto de *software*. Pressman (2011, p. 363) fala a respeitos destas:

> •Efetividade: capacidade do produto de *software* de permitir que usuários atinjam metas específicas precisão e completude, em um contexto de uso específico.
> •Produtividade: capacidade do produto de *software* de permitir que seus usuários empreguem quantidade apropriada de recursos em relação à eficácia obtida, em um contexto de uso especificado.
> •Segurança: capacidade do produto de *software* de apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, *software*, propriedades ou ambiente, em um contexto de uso especificado.
> •Satisfação: capacidade do produto de software de satisfazer usuários, em um contexto de uso especificado. (PRESSMAN, 2011, p.363)

Conforme mencionado acima pela ISO/IEC 25000 e Pressman (2011), estes são atributos próprios da qualidade que podem ser melhorados quando utilizadas técnicas de prototipagem.

2.6. Trabalhos relacionados

Como forma de dar consistência diversos trabalhos acadêmicos dão atenção ao melhoramento das técnicas de obtenção de requisitos e validação dos mesmo com os clientes de forma a ter uma consequente melhoria na qualidade de software aplicada ao produto final. Nesta vertente ressaltam Silva e Silveira (2016) que a prototipagem pode ser encarada como uma técnica ágil de captura de requisitos integrante da engenharia de requisitos ágil, sendo está acompanhada de outras técnicas comuns como entrevistas, *brainstorms* e casos de uso.

Os autores reforçam a realidade de que essas técnicas são utilizadas na metodologia *Scrum* no momento da formulação do *Product Backlog* pelo *Product Owner*, juntamente com as partes interessadas, bem como são usuais quando nas reuniões de refinamento de requisitos entre a equipe como um todo, incluindo-se aqui todos os *stakeholders p*resente no método *Scrum*.

Afirma ainda Silva e Silveira (2016) que a prototipação auxilia para que a metodologia ágil não deixa de efetivar as atividades implícitas na engenharia de requisitos, tais como: Elicitação, Análise, documentação e validação de requisitos. Finalizando os autores defendem que "*quanto maior o entendimento do projeto a nível de requisitos, menor será o risco de mudança e maior será a confiança da equipe em atender as necessidades do projeto de software solicitado*. " (SILVA; SILVEIRA, 2016, p. 9).

Em seu artigo, Dolci (2011) relata que a prototipação é uma prática relacionada a interatividade, é comumente utilizada na metodologia ágil *Dynamic Systems Development Method* (DSDM) e também junto ao desenvolvimento rápido de aplicações (RAD) que tem por características fundamentais ter rapidamente programas em operação.

O autor também salienta que a modelagem ágil (AM) é outra utilizadora do conceito de interatividade, esta, por sua vez, faz uso dos *feedbacks* recebidos pelos clientes como retro alimentadora para as aplicações futuras, primando pela rapidez na correção de erros e agilidade no planejamento, de forma a ter aplicações funcionando o mais rápido possível em menor tempo.

Confirma Dolci (2011) que o maior objetivo da modelagem ágil é evitar processos que não contribuam para a entrega do software operacional. O autor ressalta como exemplos a extensa documentação, técnicas falhas, modelos e artefatos desnecessários, frisando que os mesmos podem ser substituídos por técnicas robustas e mais eficazes, tal como a prototipagem.

Dolci (2011. p, 84) reafirma a característica relacionada à superioridade da técnica de prototipação para o entendimento do escopo dos requisitos, nisto fala "*que a excelência técnica não se estabelece pela capacidade de lidar com a complexidade; pelo contrário, o aprimoramento técnico para manter a qualidade está baseada na simplicidade*". Esta simplicidade é fortemente utilizada na modelagem ágil (AM) que prega segundo o autor a construção de modelos simples com o que se sabe do momento e que estes modelos sejam aperfeiçoados mediante o surgimento de novos requisitos. Este fato elucidado pelo autor reforça a prototipagem como uma técnica superior na obtenção da qualidade do produto ágil.

Dolcio (2011) também afirma que a metodologia *Extreme Programming* (XP) faz uso da simplicidade através de prototipação evolutiva, quando avaliado que somente uma parte da aplicação é desenvolvida primeiramente evitando que se construa partes que poderiam ser desnecessárias no futuro da aplicação. Essa metodologia segundo o autor prima pela qualidade técnica de que os erros necessitam ser encontrados o mais rapidamente possível, fato que justifica o uso da prototipagem evolutiva como identificadora ágil de falhas.

Soares (2017) ressalta que a prototipagem pode ser seguida de malefícios, tais como mascarar a real complexidade do sistema ou criar falsas expectativas nos clientes que não condizem com a realidade. O autor denota diversos fatores que devem ser reforçados no processo para que a técnica obtenha sucesso como:

- A prototipação deve prover um mecanismo de comunicação;
- Um mecanismo de união entre o time;
- Manter a confiabilidade do cliente;
- Garantir de que os requisitos mais claros e definidos sejam usados na prototipagem;
- O processo de desenvolvimento deve prover as ferramentas para a prototipagem (hardware e software);
- O planejamento deve considerar a prototipagem;
- A prototipagem deve ser encarada como um processo de avaliação;
- A organização deve estabelecer a prototipagem como cultura;
- O processo da organização de auxiliar a escolha e utilização da prototipação;
- Os desempenhos dos protótipos devem ser considerados;
- As decisões de desenho devem ser definidas pelo uso conjunto da técnica;
- O processo deve prever a modularização dos protótipos;
- O processo deve prover a arquitetura dos protótipos;
- A revisão dos protótipos deve ser levada a sério;
- Os riscos devem ser considerados como forma de avaliação.

Conforme Soares (2017, p. 17) esses são fatores que devem ser comtemplados pelo processo para que a técnica de prototipação venha a ser empregada com sucesso no atingimento de seus objetivos. O mesmo corrobora que é muito importante que as empresas oficializem o uso da técnica para a aplicação e que “muitas empresas aplicam a técnica sem nem mesmo lhe dar o nome explicitamente”.

Chammas e Quaresma (2013) em seu trabalho sobre o enfoque ergonômico para a metodologia de design em interfaces defende também o uso da prototipação ao abordar a técnica de *Design thinking,* que segundo a autor é baseada na exploração de possibilidades priorizando mapas mentais ao invés do pensamento linear. Segundo os autores, o *Design Thinking* faz uso da observação comportamental das pessoas,

transformando as suas ações em produtos e serviços de melhoria, não desvinculando o usuário e processo.

Para os autores, a prototipagem é a peça mais fundamental do sucesso da técnica de *Design Thinking*, pois explica como o ato de transcender entre o físico e o abstrato sem compromisso com comportamento técnicos, apenas o suficiente para decidir o que fazer (Chammas e Quaresma, 2013). Sendo então muito mais fácil a visualização de todos os materiais envolvidos, demonstrando a capacidade de abstração da técnica. Os autores também destacam o uso da prototipagem junto a outras metodologias modernas, tais como o *Lean UX* (Chammas apud Gothelf, 2013), o *Rapid Contextual Design* e o *Wheel* (Chammas apud Hartson e Pyla, 2012). Também reiteram a importância do uso da técnica como artefato qualitativo para interfaces digitais, primando sempre pela experiência do usuário.

Outro trabalho interessante é o de Sano (2012), a mesma aponta em sua monografia a importância da prototipação como metodologia isolada sendo inserida junto os métodos prescritivos, bem como a sua natureza técnica, quando vista de forma ágil incluída, portanto nas diversas metodologias ágeis existentes.,

Todos os trabalhos anteriores citam a prototipação como técnica evidente para a elucidação dos requisitos e transformação da realidade abstrata destes em algo físico. Desta forma, a técnica apresenta-se ligada intimamente aos requisitos funcionais e não funcionais da aplicação, nas fases iniciais do processo, como pode ser visto nos trabalhos de Chammas e Quaresma (2013), que ressaltam a visão da prototipação para melhoria do *design* de interfaces, bem como a visão ágil da prototipação por Silva e Silveira (2016) e a visão processual da prototipação, por Soares (2017). Porém, seja a técnica aplicada a métodos prescritivos ou ágeis há um consenso entre os autores e na literatura pesquisada, que a técnica se adequa e é mais implementada nas fases iniciais do desenvolvimento de software e não dentro do processo e da etapa de construção, ou seja, já quando o cliente forneceu uma quantidade satisfatória de requisitos.

O diferencial desta pesquisa em relação aos estudos anteriores é o foco na análise dos benefícios, assim como os possíveis malefícios da técnica de prototipação aplicada no ambiente ágil *Scrum*, iniciando-se a partir de requisitos básicos já fornecidos pelo cliente.

No presente estudo pretende-se realçar a prototipação como uma ferramenta robusta para a garantia da qualidade do produto e como vigia de que a conformidade dos requisitos está sendo observada, no decorrer da construção ágil dos incrementos, podendo assim prever a mutabilidade e volatilidade dos requisitos, fato implícito nos métodos ágeis, citando-se o *Scrum*.

Nesse estudo não é de interesse argumentar uma conclusão definitiva de como deve se dar a aplicação da prototipação no ambiente ágil, mas argumentar através da observação, análise e coleta de dados como ocorre o processo, permitindo-se que futuros trabalhos possam explorar ou mesmo criar modelos padronizados, que permitam a replicação e implementação da técnica em outros processos de construção de softwares diferentes do *Scrum*.

3. Metodologia de Pesquisa

Esta seção tem por finalidade demonstrar os aspectos metodológicos desta pesquisa. Toda a pesquisa começa com a descrição de um método, argumenta Lakatos (2010) a respeito da definição de método:

> é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo – conhecimento válidos e verdadeiros – traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista.

Portanto é através de uma metodologia bem executada que o pesquisador pode apresentar um processo e/ou sistema adequadamente a seus interessados, de forma a dar subsídios verdadeiros e empíricos que validem a sua proposta.

Esta pesquisa se guiará pelo uso da abordagem quantitativa para avaliar as histórias de usuário trabalhadas durante 16 semanas (4 meses – 8 *Sprints*) na empresa "ABC". Marconi, 2011 apud Richardson (1999, p. 70) diz que o método quantitativo:

> caracteriza-se pelo emprego da quantificação tanto nas modalidades de coleta de informação quanto no tratamento delas por meio de técnicas estatísticas, desde as mais simples como percentual, média, desvio-padrão, às mais complexas como coeficiente de correlação, análise de regressão etc.

Diferentemente da abordagem quantitativa, a abordagem qualitativa tem por característica segundo Marconi (2011, p. 269):

> preocupar-se em analisar e interpretar aspectos mais profundos, descrevendo a complexidade do comportamento humano. Fornece análise detalhada sobre investigações, hábitos, atitudes, tendências de comportamento, etc.

A metodologia qualitativa será dada neste trabalho através do estudo de caso do desenvolvimento de software na empresa "ABC", podendo assim entender as nuances e agregar significado ao método quantitativo, seus porquês e correlações imediatas, que não poderiam ser entendidas isoladamente com o uso de somente um processo metodológico.

Neste trabalho decidiu-se pelo uso da pesquisa exploratória, visto a realidade inovadora do tema quando aplicado a metodologias ágeis. Segundo Saccol (2012, p. 37 apud Gil (2008, p. 27) as pesquisas exploratórias, tem por finalidade "desenvolver, esclarecer e modificar conceitos e ideias, tendo em vista a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores".

O estudo terá por objetivo avaliar o processo de prototipagem realizado na empresa "ABC", localizada em Porto Alegre, Rio Grande do Sul, uma multinacional do ramo de Recursos Humanos, dentro do time "Spartans", composto por 6 pessoas no Brasil e 6 pessoas no time indiano, de um total de 55 mil colaboradores no mundo. A empresa faz uso da metodologia *Scrum*, e o gerenciamento do trabalho ocorre em *sprints*, sendo uma *sprint* equivalente ao tempo de duas semanas.

Foi pesquisado estes dois times que formam um único time. Este estudo não remete a realidade da corporação como um todo e vale salientar que dado a característica volátil do processo *Scrum*, um mesmo método pode ser executado de formas diferentes, devido às realidades únicas aplicadas a cada time.

Primeiramente, através da técnica de estudo de caso se analisou o contexto da unidade “ABC” e como o processo de desenvolvimento é executado; mediante esta exposição teve-se oportunidade de entender como se deu o uso da prototipagem como processo de redução de defeitos.

Foram analisadas as histórias de usuário no repositório do projeto “Spartans”, responsável pelo controle de créditos fiscais americanos, durante 16 semanas (4 meses – 8 *Sprints*), bem como coletados os defeitos produzidos nos *sprints* posteriores. Estas histórias foram divididas em “Demonstração feita” e “Processo normal”, sendo que o valor correspondente a “Demonstração feita” representa que um protótipo executável foi apresentado ao cliente como forma de validação do mesmo, sem que o processo de desenvolvimento tenha sido terminado formalmente. Atividade “Processo normal” corresponde as fases naturais do processo de desenvolvimento de software (elicitação, projeto, modelagem, testes e implantação), não contemplando o uso da prototipagem. Para a demonstração destes dados foi utilizado, conforme Saccol (2012, p. 91) “*a técnica de análise de dados quantitativos baseada na frequência das respostas a uma determinada questão (com uso de tabelas, quadros e gráficos).* ”

Este trabalho foi realizado com vista na seguinte ordem de processo: fundamentação teórica, coleta de dados através da técnica de pesquisa documental e observação participante (Marconi, 2011) das histórias de usuário na empresa “ABC”, estudo de caso do uso da técnica de prototipagem na empresa “ABC”, tabulação dos dados, análise e apresentação destes.

4. A Abordagem Qualitativa

Esta seção destina-se a apresentação e análise dos dados coletados no contexto da empresa “ABC”, havendo-se de salientar-se que não houve interferência para com o processo estudado neste caso, não se motivando uma pesquisa-ação, conforme Marconi (2011).

Marconi (2011) ressalta que o valor do estudo de caso refere-se à observação de um grupo particular e que seus atributos não são generalizados a todos os processos semelhantes. Todavia, visa objetivar uma confirmação dos porquês de determinado fato estar ocorrendo. O autor também salienta que para a validação de um estudo de caso qualitativo, quanto mais dados forem coletados, visualizados e esboçados, mais rico fica o entendimento dos porquês de determinado acontecimento se dá dando credibilidade e significado ao sistema dos entrevistados.

4.1 O estudo de caso da empresa “ABC”

A empresa “ABC” produz softwares de gestão para sistemas de Recursos Humanos, atuando em vários países internacionalmente, configurando uma multinacional. Não somente a empresa trabalha com esses sistemas, mas contêm no portfólio de aplicações várias células/segmentos como o tratamento de créditos fiscais americanos, que nesta empresa é dada manutenção por times divididos globalmente entre Índia, EUA e Brasil, sendo o último o time foco deste trabalho.

No time do Brasil, denominado “*Spartans*” existem 6 integrantes (2 analistas de testes, 1 gerente de projetos, 2 desenvolvedores e 1 administrador de banco de dados) e

no time da Índia, mais 6 integrantes, divididos em igual segmento, ambos trabalhando com a metodologia ágil Scrum, como apresentada na seção 2.4 deste artigo.

Foi avaliado o período de 16 semanas nas interações do time Brasil-Índia-EUA, analisando-se o processo, assim como os artefatos produzidos no período, centrados basicamente nas histórias do usuário presentes, no *Sprint Backlog* da Figura 11. Também foram observadas as mudanças na organização (mencionada nos parágrafos abaixo), que levaram ao surgimento da técnica de prototipação, como aliada na entrega dos artefatos.

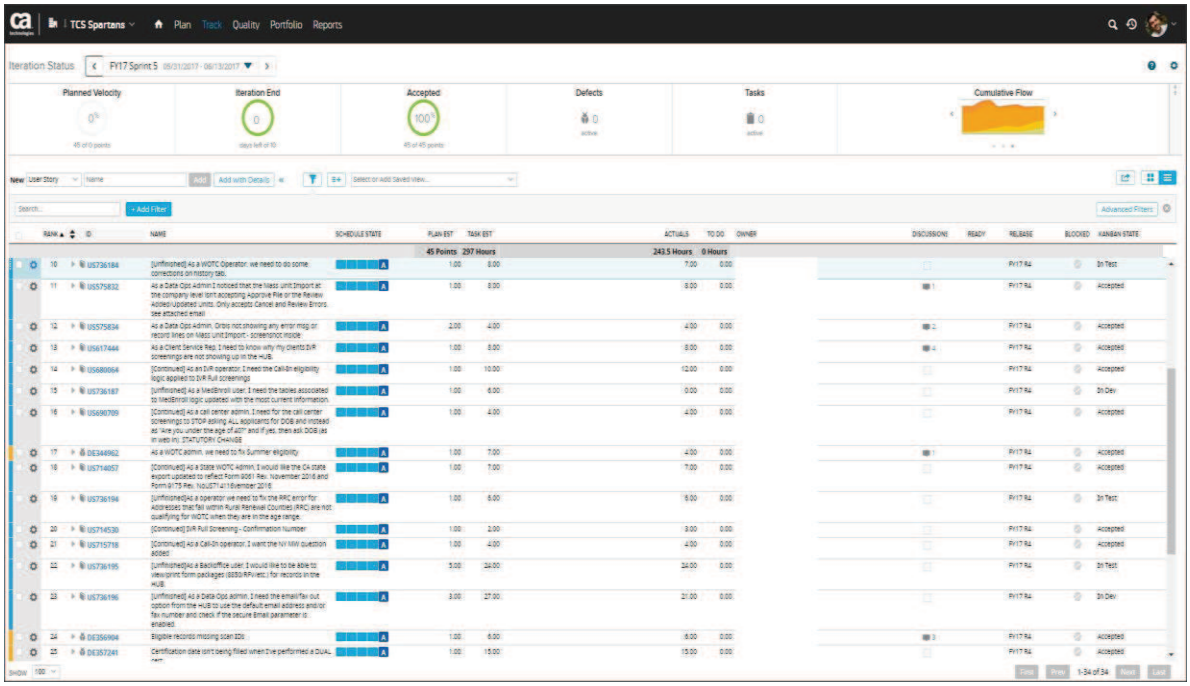


Figura 11 – *Sprint Backlog – Rally Tool*

O time Brasil, assim como todos os times globais fazem uso da ferramenta *Rally* para o gerenciamento das atividades referentes ao uso da metodologia ágil *Scrum*, conforme vista na Figura 11, que está representado o *Sprint* 5, do release FY17- R4, com todas as históricos de usuário já aceitas pelo cliente.

O processo na empresa "ABC" está em constante mudança organizacional, fato corroborado pela distribuição internacional dos times, ou seja, o conhecimento está em constante mudança, bem como as ferramentas da aplicação desenvolvidas, fatos implícitos como característica da metodologia Scrum, que leva a mudança como seu foco principal.

O time *Spartans* faz uso da metodologia *Scrum. E*ste time era composto principalmente por indivíduos, que estavam em outros times dissolvidos, devido à uma reorganização geral, ocasionada pelo fim do projeto de migração para uma plataforma mais inovadora, no qual o sistema antigo que era denominado "VISTA" construído sobre a linguagem Progress, foi reescrito para Java – Sistema ORBIS.

Isto culminou que grande parte dos integrantes anteriores fossem realocados em outros projetos e os remanescentes foram alocados neste novo time, que ficou

principalmente responsável por estabilizar a nova aplicação, assim como manter o processo *Scrum* em andamento.

Sendo assim, o time Spartans passou a contar com 6 membros brasileiros e 6 membros indianos, ressaltando-se que a observação e coleta dos dados somente iniciou-se após a conclusão desta reestruturação.

Segue abaixo uma breve descrição de como era realizado o processo *Scrum* do time *Spartans*, denota-se reuniões, pessoas e artefatos:

- *Product Backlog*: Mantido pelo time junto com o *Product Owner*, apresentando-se desorganizado em muitas das vezes, não tendo as histórias priorizadas como deveriam ser, conforme vistas na seção 4.2 sobre a metodologias *Scrum*. Muitas das histórias eram mantidas atualizadas pelo time, com recorrentes reuniões e conversas paralelas com o *Product owner*;
- *Sprint Backlog*: Concentra, como já mencionado as histórias que priorizadas pelo *product owner* serão trabalhadas no período determinado, neste caso a priorização era definida por email recorrentes advindos da comunicação intensa com product owner e *Scrum master*, sendo que como salientado muitas histórias prioritárias estavam indo para o *Sprint* corrente sem o mínimo de informações necessárias, o que acarretava que muitas histórias ficavam paradas no *Sprint* sem poder ser trabalhadas, enquanto não houverem uma real certeza do que deveria ser feito;
- *Sprint Planning*: Esta reunião, no período analisado, não era realizada, sendo que os pontos eram postos nas histórias levando-se em conta o nível de entendimento pessoal e não coletivo. Também grande parte das histórias eram analisadas após entrarem no *Sprint Backlog* (quando muito urgentes), isto acarretando que muitas histórias estavam no *Sprint Backlog*, sem um real escopo definido e final;
- *Daily meeting*: é realizada todos os dias pelo time, nela como dita o processo Scrum levanta-se os impedimentos, o que foi feito e o que se fará;
- *Sprint review*: Este processo era realizado em uma mesma reunião, mas diferentemente das regras *Scrum* visava mostrar não somente histórias completas, mas também incrementos inacabados:
    - Apresentação de histórias em progresso (ocorrida no meio do *Sprint*): este é mais um momento da reunião de *review* do que uma nova reunião propriamente dita, era o momento de o time apresentar as suas histórias em processo que ainda não estão terminadas, mas para se obter uma aprovação dos usuários quanto ao estado do processo, nesta etapa do processo de *review* que entra a ideia da prototipagem evolutiva, pois os artefatos eram refinados ou mesmo aprovados pelos clientes finais;
    - Apresentação de histórias acabadas: este momento ocorria junto ao primeiro momento no qual o cliente dava seu aceite nas histórias acabadas e promovia as mesmas para a release, ressalta-se que é nunca foi -se observado a negativa total de um incremento, mas somente uma negativa parcial, nestes casos era aberta uma nova história de usuário

para o processo *Sprint* como forma de manutenção e evolução do artefato anterior.

- *Sprint Retrospective*: Era realizada após o final do *Sprint review*, levantando-se os pontos bons e ruins da *Sprint* que passou, porém não exercia uma atuação como processo melhorador para os próximos *sprints*, mas muito mais como uma reunião que o rito necessitava;
- *Scrum Master*, *time* e *Product Owner*: A relação entre estes ocorria conforme o rito *Scrum*, o *Scrum master* era o responsável pela organização do *Sprint* backlog, marcação das reuniões e destravamento de empecilhos com o *product owner*. O *product owner* era apresentava uma característica participativa, porém reclusa em não levantar todos os requisitos das histórias de usuário fornecidas no product *Backlog*. O time era participativo e ajudava o *Scrum master* em suas funções.
- Histórias de usuários: eram delimitadas pelo *product owner*, não comumente priorizadas pelo mesmo formalmente na ferramenta *Rally*, mas por formas informais como e-mail. Apresentavam muita pobreza quanto a detalhes carecendo de uma intervenção do time e do *Scrum* master para levantamento de requisitos, este por sua vez não realizado na reunião de Sprint Planning devido a sua não existência;
- *Burndown* da *Sprint*: Enfraquecido pela irreal exposição dos dados da equipe, devido ao pensamento individualista presenciado no período, bem como a não atualização correta dos dados de horas e estimativa de horas das histórias do usuário;
- Requisitos: eram definidos através de perspectivas informais (e-mails, conversar pelo bate papo, afins) pelo time e *Scrum master* mediante intervenções com o *product owner*;

Como verificado acima, este era o processo realizado na empresa “ABC”.

Na perspectiva da realidade do ciclo de desenvolvimento dos incrementos, o processo ocorria conforme visto na seção 4.2, na seguinte ordem: Primeiramente Requisitos>Projeto> Codificação e Teste unitário> Testes de sistema/Integração> e por fim os Testes de aceitação do usuário.

Ressalta-se aqui que devido à natureza atemporal do processo *Scrum*, nota-se que os incrementos eram apresentados ao cliente na etapa de codificação, se fazendo uso da técnica de prototipagem evolutiva. Com isto os testes unitários eram antecipados, configurando o uso de um processo ágil para identificação e mitigação de erros, que poderiam ser promovidos para a etapa de testes de sistema, ou omitidos, sendo assim levados aos sistemas de produção.

Após os testes de sistema e os testes de aceitação, a história de usuário passa a ter a idéia de finalizada e o “pronto” é finalmente atribuído ao incremento, sendo a história movida como aceita no *Sprint backlog* (como visto na Figura 11) e sendo possível sua promoção ao sistema de pré-produção (IAT).

Cada time da empresa “ABC” faz uso de um ambiente diferente, que se adequa as necessidades de cada fase do ciclo de desenvolvimento da aplicação. Sendo o

ambiente dos desenvolvedores denominado de DIT, o de testadores conhecido como FIT, o de pré-produção denominado de IAT e, por fim, outro ambiente chamado PROD, que é o sistema de produção.

Quanto à realização dos testes, salienta-se que os testes unitários são realizados no ambiente de DIT e os testes funcionais e testes de aceitação no ambiente de FIT. O ambiente de IAT é onde os *releases* ocorrem após a certificação em FIT. Este ambiente é utilizado pelos clientes e também testadores como uma última oportunidade de validação, sendo caracterizado por testes de integração e regressão, mas não fazendo parte do processo *Scrum*, que tem sua última etapa concluída dentro do sistema de FIT.

4.2 Análise da Prática de Prototipação na Empresa "ABC"

É premissa deste tópico apresentar uma análise pessoal e individualista para com o processo da empresa "ABC", tendo em vista os conceitos já apresentados sobre as metodologias tradicionais e ágeis na totalidade da seção 2.

Conforme verificado no tópico 4.1, a empresa "ABC" sofria com uma reestruturação, motivada principalmente por mudanças profundas e dramáticas no sistema principal, necessárias à sua sobrevivência. Portanto, todos os processos e sistemas subjacentes se viram afetados pelo fato, todavia com o auxílio e uso da metodologia ágil *Scrum,* o processo ainda que não maduro conseguiu atingir os propósitos da sustentabilidade, mesmo que não totalmente eficiente.

Como forma de melhor adequar o processo a esta fase de adaptação e transformação intensa, o time decidiu efetivar as cerimônias *Scrum*, não rigorosamente, mas substituindo e as adequando a sua realidade processual e localizada. Este fato é confirmado por Schwaber (2009), quando fala da alta versatilidade e adaptabilidade da metodologia *Scrum*.

A prototipagem evolutiva foi utilizada como verificado em um único momento do ciclo de vida, divergentemente do comumente usado na literatura, porém em diferentes pontos do processo:

1. Na fase de codificação e testes unitários no ambiente de DIT;
2. Junto a reunião de *Sprint Review* ao final do Sprint para os interessados de forma a validar os incrementos levando-se em conta atributos de ordem visual como design, usabilidade, eficiência e confiabilidade, dando oportunidade para que o cliente refine melhor os requisitos.

Não se afasta a possibilidade de que a prototipagem descartável de baixa fidelidade tenha sido usada, mas não comumente pelo time como parte da elicitação de requisitos. Foi, no entanto, verificado o uso de *mockups* e *wireframes,* durante o período analisado pelos clientes como forma de exemplificar o que desejavam ao time.

Quando se fala do uso da prática de prototipação evolutiva neste estudo de caso, também se retorna ideia de que muitas das histórias de usuários não tiveram o uso da técnica devido à realidade conclusiva dos requisitos apresentados, não se fazendo necessária a técnica, portanto foi presenciada a sua exclusão.

Também histórias que eram baseadas em incrementos complexos demais, ou referentes a protótipos muito grandes para uma demonstração foram excluídos pelo

time, o que nos remete ao pensamento que somente histórias simplificadas e com um intervalo de requisitos definidos e básicos, fáceis de se visualizar e demonstrar aos clientes obtinham a presença da técnica. Salienta-se que não foi interesse desta análise julgar o certo e o errado em como os processos de desenvolvimento devem ou deveriam ocorrer mediante a extensa literatura envolvida.

Conforme Molinari (2012), ainda salienta-se que os estudos de casos têm a natureza de demonstrar conhecimentos aplicáveis a variáveis isoladas e restritas a um único sistema, todavia a replicação e generalização se tornam difíceis de estender a outros ambientes, mas não se descarta a possibilidade de ocorrerem de formas semelhantes ou divergentes, quando analisadas as mesmas variáveis de ação.

5. A Abordagem Quantitativa

Esta seção tem por objetivo apresentar os dados quantitativos coletados através da observação do *Sprint Backlog* do time "*Spartans*", no período de 16 semanas (4 meses – 8 Sprints), além de uma discussão sobre os principais pontos encontrados.

5.1 Apresentação e Análise dos Dados Quantitativos

Verificou-se o *sprint backlog* destas 8 semanas e as histórias de usuários aceitas durante o período foram divididas em "Não realizado", histórias onde o ciclo de vida não teve a técnica de prototipação usada e "Realizado", onde a técnica foi aplicada na etapa de codificação e de testes unitários aos clientes. Também se criou uma categoria denominada "Outros", onde estão contidas histórias de investigação, que são itens feitos para elucidar dúvidas pertinentes a defeitos e/ou histórias de usuário pré-existentes.

Utilizou-se o gráfico de linhas, por sua capacidade de demonstrar a evolução dos artefatos na decorrência do tempo. Foram criados 3 gráficos no total, referentes aos três momentos desta análise: (*i*) o gráfico "Evolução de adoção da técnica x Quantidade total de histórias por Sprint"; (*ii*) o gráfico "Evolução dos defeitos por categoria"; (*iii*) gráfico "Evolução do total de defeitos x Total de histórias".

A seguir demonstra-se através da abordagem quantitativa, os benefícios e malefícios envolvidos com o uso da técnica de prototipação, juntamente a metodologia ágil Scrum, no time de estudo "*Spartans*", verificando-se as situações envolvidas no decorrer do tempo, que levaram a evolução dos valores apresentados.

Ressalta Marconi (2011) que a abordagem quantitativa tem por principal caraterísticas propor, modificar, esclarecer e fundamentar respostas e conceitos através da percepção numérica e exata dos fatos, não deixando margem para interpretações vagas e interferências pessoas do pesquisador ao sistema. Portanto é importante mensurar dados que permitam validar questões do sistema sem enfoque a dúvidas, margeando as incompreensões com dados empíricos.

O Gráfico 1 demonstra a evolução da adoção em 8 *Sprints* da técnica de prototipagem para demonstração de histórias de usuários, que ainda estão sendo trabalhadas pelo time de desenvolvimento contra a quantidade de histórias em que a técnica não foi utilizada, bem como o total de histórias por *Sprint*.

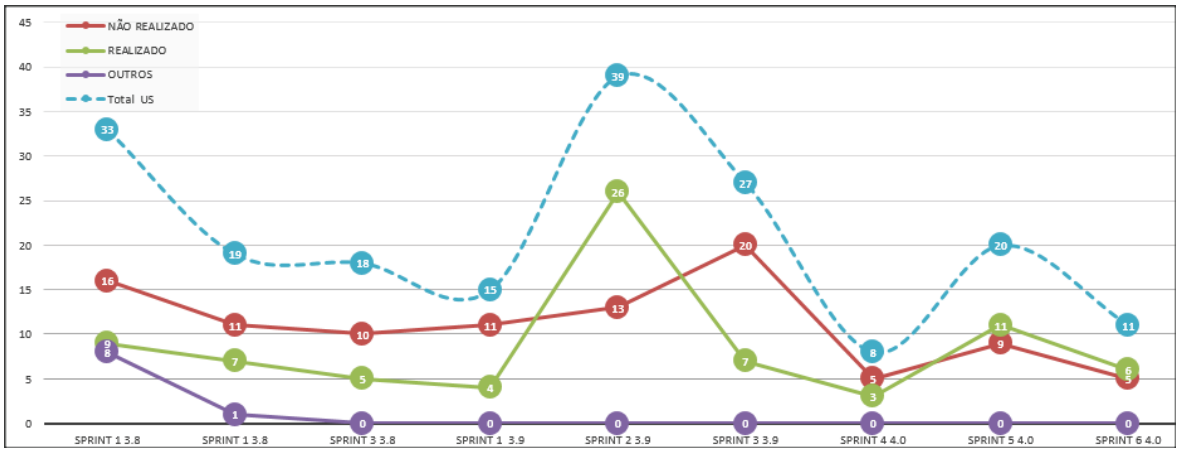


Gráfico 1 – Evolução de adoção da técnica de prototipação

Nota-se no Gráfico 1 que houve uma consequente diminuição do número de histórias com uso da técnica de prototipação até o final do *Sprint 1* do release 3.9. Todavia, o número é expressivo quando comparadas as grandezas "Total de histórias x Realizado x Não realizado", demonstrando que a equipe estava empenhada em demonstrar os incrementos, antes do conceito "pronto" estar efetivamente concluído.

Outra nota importante deste gráfico se deve aos acontecimentos da *Sprint 2 R3.9,* na qual houve um consequente aumento no número de histórias trabalhadas, bem como o aumento expressivo de histórias que tiveram o uso da prototipação, fato aqui motivado pela incerteza crescente quanto aos requisitos do momento.

A *Sprint 3 R3.9* definiu-se pela consequente baixa adoção da prototipagem aqui podendo ser motivada pelo período de finalização dos testes da *release de IAT* e consequente diminuição de histórias definidas pelo *Product Owner*. Caracterizando um período conhecido como "*code freezing*" ou congelamento de codificação e criação de novas histórias em FIT e DIT. Este período conforme verificado na empresa "ABC" é caracterizado pelo momento de corrigir os defeitos, que por ventura foram encontrados dos testes feitos em IAT, para posterior *deploy* no ambiente de PROD.

Verificando-se as histórias de usuário do *Sprint 2 3.9* pode-se salientar a existência de uma divisão de histórias de usuários complexas em incrementos menores, para melhor elucidação de requisitos e facilidade de entrega ao cliente, fato que contribuiu satisfatoriamente com o aumento significativo da quantidade de histórias, que tiveram a técnica realizada.

Analisando-se a relação de "total de histórias por *sprint*" e "realizado" também no Gráfico 1, nota-se que a técnica de prototipação foi adotada com sucesso em média 37% do total de histórias por *Sprint* durante o período de análise.

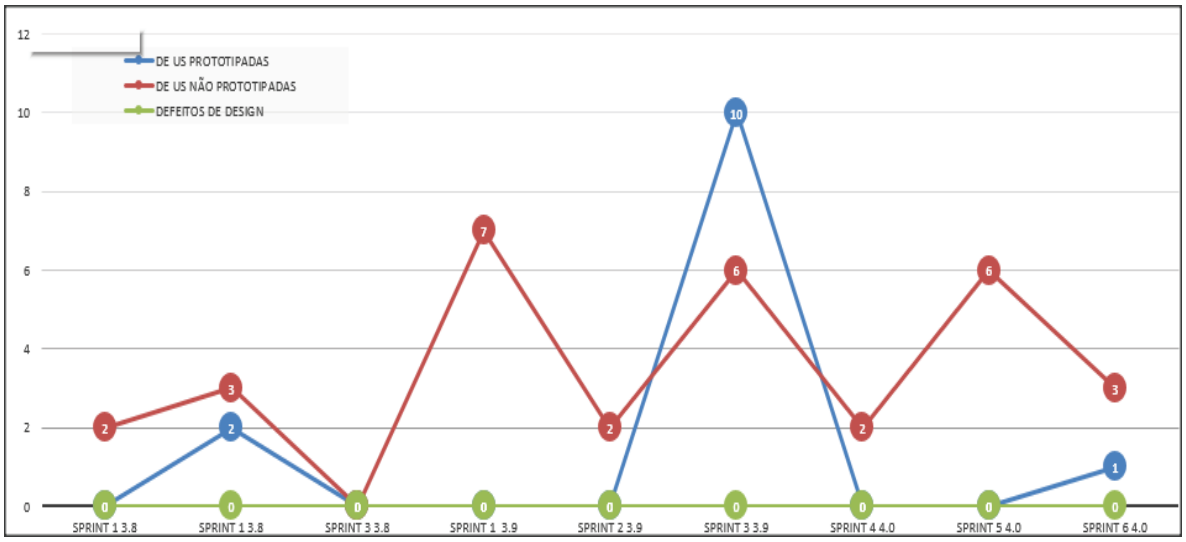


Gráfico 2 – Evolução dos defeitos por categoria

O Gráfico 2 tem por característica demonstrar a evolução dos defeitos em decorrência dos *Sprints,* agrupando-os em "Defeitos relacionados a histórias (US) prototipadas" e "Defeitos relacionados em histórias (US) não prototipadas", bem como a presença de defeitos relacionados a *design*.

O primeiro fato marcante se deve a inexistência de defeitos referentes aos requisitos de *design*. Este por conseguinte pode ter suas causas atribuídas ao uso da técnica de prototipagem para validação das interfaces em seu estado de progresso. Todavia, como notado no Gráfico 1, muitas histórias de usuários não faziam uso da técnica de prototipação. Este impasse pode ser resolvido, quando analisada a amostragem das histórias, que não fizeram uso da técnica. Nestas notou-se que os requisitos não eram tidos como voláteis e que as modificações não eram delimitadas por alterações visuais imediatistas, como em outras histórias de usuário na qual a técnica foi empregada.

Novamente se faz aqui importante diferenciar histórias que não poderiam ter a técnica executada. Neste contexto é possível após a análise das mesmas levantar as seguintes ocasiões em que a técnica não deve ser empregada:

- Histórias em que as mudanças eram tidas integralmente em requisitos funcionais complexos, chamadas a subsistemas;
- Histórias em que as mudanças eram complexas demais para que o processo fosse integralmente demonstrado de forma satisfatória;
- Histórias em que os requisitos estão completos e não existe uma presença de mudança nos mesmos;
- Histórias que tiveram a presença de alteração somente de requisitos não funcionais e que não poderiam ter seus incrementos demonstrados pelo uso da técnica de prototipação, cita-se requisitos de desempenho, recuperação, chamadas a subsistemas externos, segurança e confiabilidade;
- Histórias que detinham um contexto voltado para execução de *scripts* de bancos de dados.

Retornando a análise do Gráfico 2 pode-se conjecturar um sucesso do uso da técnica de prototipagem, no que condiz a diminuição do número de defeitos encontrados. Todavia, houve um aumento expressivo na presença de defeitos no *Sprint 3.3.9*, motivado pelos testes de IAT que compreendem os testes de todas as histórias de usuário dos *Sprints* da *release* 3.9.

Analisando-se as histórias de defeito detalhadamente pode-se levantar as seguintes hipóteses para esse aumento do número de defeitos, mesmo em histórias em que a técnica foi empregada:

- Histórias de usuário complexas em que muitos requisitos funcionais e não funcionais estão ligados entre si e com outros sistemas existindo uma omissão dos mesmos;
- Super estimativa do uso da técnica como mitigadora de defeitos e clarificadora de requisitos omissos;
- Volatilidade dos requisitos de usuário;
- Novos incrementos que não são totalmente compreendidos pelo cliente e pelo time de desenvolvimento;
- Existência de requisitos que dependem de chamadas a outros sistemas;
- Cenários de usuários omissos devido a não previsão dos mesmos;
- Complexidade exagerada e interdependência de histórias de usuário;
- Integração dos incrementos apresentou-se falhas omitidas pela ação do cenário;

Mesmo que levantadas estas ponderações que restringem o uso da técnica de prototipação como fator de melhoria na qualidade, os dados quantitativos demonstram que as histórias que não tiveram o uso da técnica tiveram a presença do maior número de defeitos. Isso leva a crer que a prototipação mesmo que superestimada, pode mitigar a produção de defeitos melhorando a qualidade dos artefatos entregues, valendo-se da premissa da qualidade de software que salienta a inexistência de sistemas que não tenham falhas.

O Gráfico 3 apresenta uma evolução do número total de defeitos em comparação ao total de histórias trabalhadas por *Sprint* e ao número de histórias que tiveram prototipação. Estes defeitos aqui mensurados são a soma dos defeitos com e sem uso da técnica de prototipação, não categorizados quanto ao seu período no ciclo de desenvolvimento do incremento. Esta avaliação confirma o quão importante é levantar os defeitos da aplicação o mais cedo possível, evitando que se alastrem para o ambiente de produção e possam afetar os usuários finais.

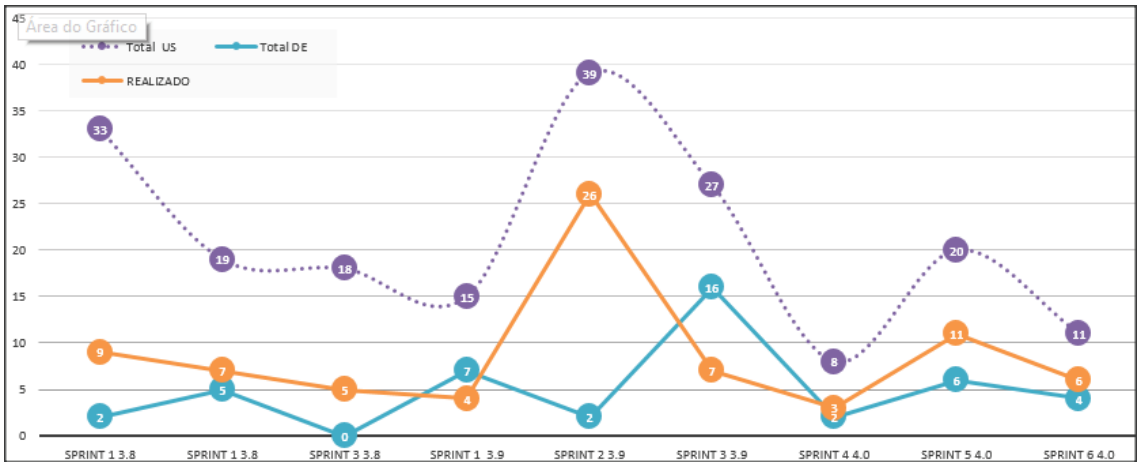


Gráfico 3 – Total defeitos x Prototipação x Total histórias

Outra característica notada é a relação de diminuição de defeitos, quando realizada a comparação de histórias que tiveram o uso da técnica contra o total de defeitos, validando o uso benéfico da prototipação como indicadora de melhoria na entrega dos incrementos. Nota-se que o *Sprint 3* e *1* da *release 3.9* tiveram menos histórias com o uso da técnica, o que pode ter motivado o surgimento de mais defeitos na totalidade.

5.2 Discussão

Este tópico tem por finalidade apresentar de forma geral os resultados obtidos mediantes ao uso da técnica de prototipação na empresa "ABC", agrupando os benefícios e malefícios, que podem ser vistos no Quadro 3, encontrados durante a análise qualitativa e quantitativa dessa pesquisa.

Realça-se que o método Scrum como visto na seção 4.2 deste trabalho não faz menção da técnica de prototipagem como parte da literatura, porém argumenta que novos processos devem ser inseridos no método de forma a dar qualidade e andamento aos demais processos. Nesse sentido, esse estudo propõe uma abordagem de uso de prototipação, num ambiente ágil de desenvolvimento baseado em Scrum.

A pesquisa demonstrou o uso dessa técnica como fortalecimento da ausência da reunião de *Sprint Planning* prevista nas cerimônias. Entretanto, mesmo que a reunião por ventura estivesse sendo executada, a técnica não deixa de ser avaliada como uma técnica de melhoria na qualidade dos incrementos, bem como o seu uso fora das etapas iniciais do processo não deve ser encarada de forma discriminatória. Conforme visto exemplificado nos tópicos 5.1 e 5.2 podem haver fortes indícios, que cooperam para que a técnica seja encorajada também, quando na presença de um artefato semiconcluído, de forma a pré validar os requisitos com os clientes.

Esta abordagem, no entanto, exagera na participação do cliente para com o projeto, fato somente possível quando se utilizadas métodos como o *Scrum*, onde o *Product Owner* personifica o cliente dentro do time, promovendo o *feedback* imediato quanto aos incrementos finalizados.

Benefícios
Agilidade quanto a mudanças pontuais
Antecipação na descoberta de defeitos.
Aprimoramento de funções da aplicação
Aumento da confiança do *Product Owner*
Aumento da experiência dos envolvidos quanto a parte do sistema demonstrada.
Aumento na qualidade dos incrementos entregues
Diminuição do retrabalho e economia de tempo
Diminuição dos defeitos referentes a requisito de *design*, estética, usabilidade, *interface*.
Economia de horas de trabalho e retrabalho.
Economia de tempo e aproveitamento da equipe de QA.
Feedback rápido para mudanças assim como implementação ágil
Fortalecimento dos requisitos funcionais não complexos e de requisitos não funcionais ligados principalmente ao design
Melhoria na estimativa das tarefas da história do usuário
Melhoria no entendimento dos requisitos existentes e possível descoberta de requisitos omissos.
Melhoria no gerenciamento de histórias de usuário complexas, que podem ser quebradas em partes menores
Menos histórias de usuários desnecessárias
Mitigação de defeitos graves que estão ligados a funções básicas do incremento
Possível antecipação de fases do processo, erros podem ser descobertos ou novas funcionalidades inclusas
Redução de riscos
São ótimas ferramentas para sistemas simplificados e que tem relações não complexas
Malefícios
A interdependência dos requisitos não pode ser avaliada corretamente.
A técnica não é usual para todos os incrementos.
Comportamentos complexos e incrementos novos são difíceis de representar.
Defeitos de integração não podem ser previstos
Demora na resposta das dúvidas pode acarretar estagnação da continuidade do trabalho nos incrementos pós apresentação.
Desejo da agilidade na entrega do incremento pelo *Product Owner* pode fazer com que o teste de software seja menosprezado
Erros podem ser inseridos devido à falta de conhecimento e estudo por parte do *Product Owner* quanto a aplicação e sua integração
Falsa sensação de pronto, *Product Owner* pode achar que a tela apresentada será a final.
Gasto de tempo excessivo para ganhos não expressivos
Incrementos complexos não são possíveis de demonstrar.
Obscurecimento dos requisitos não funcionais e funcionais mais internos (usabilidade, desempenho, suportabilidade), podem levar o *Product Owner* a achar que este incremento está livre de falhas.
Product Owner pode não levar em consideração que o incremento está em desenvolvimento, não representando a realidade final do mesmo.

Quadro 3 – Benefícios e malefícios do uso da técnica de prototipação

No Quadro 3 são listados benefícios e malefícios, ligados ao uso da técnica de prototipação no processo de desenvolvimento Scrum da empresa "ABC", que foram primeiramente compilados dos fatos observados e dos documentos analisados pelo investigador, durante o período de 16 semanas (4 meses - 8 Sprints) no time Spartans e, posteriormente, validados na organização através de levantamento da opinião de integrantes desse projeto.

Quanto à participação na etapa de validação dessa pesquisa, os integrantes da equipe (Gerente de projeto, Scrum Master, analistas de testes e desenvolvedores) foram convidados através de e-mail a participar de uma reunião de análise e feedback das conclusões iniciais dos benefícios e malefícios do uso da técnica, fornecidos pelo investigador, no período de 2 horas.

Os mesmos resumiram os dados como apresentados acima, no quadro 3, elencando os tópicos mais relevantes ao projeto, excluindo e reformulando as duplicidades encontradas, assim como pontos incoerentes. Foi questionado ao grupo se haveriam adendos as idéias iniciais propostas pelo investigador e foi acordado entre o time que não haveriam pontos discordantes, pois durante o processo houve participação do time.

Pode-se destacar ainda, os seguintes pontos positivos dessa abordagem proposta, levantados durante essa etapa de validação da pesquisa:

- entrega de funcionalidades com menos defeitos;
- uma melhoria significativa na obtenção e elucidação dos requisitos;
- uma economia considerável do time de qualidade, que pode então investir esse tempo sobressalente em partes como a automatização de testes com foco a regressão do sistema;

Quanto aos pontos negativos identificados na abordagem proposta destacam-se:

- uma alta dependência da disponibilidade de tempo do cliente para a validação dos requisitos;
- uma superestimativa do uso da técnica para artefatos complexos, causando confusão para o cliente e aumentando do tempo do projeto.

Embora possam haver malefícios, que estão intimamente ligados a questões de bom senso no uso da técnica, a prototipação se mostra como forte aliada na melhoria da garantia da qualidade dos incrementos, que estão em processo de construção, bem como não pode ser descartada a sua natureza elucidadora de requisitos para a engenharia de software, na fase inicial relativa aos requisitos.

6. Considerações Finais

Este trabalho apresentou o estudo de caso da empresa “ABC”, com o uso da técnica da prototipação, baseando-se na análise dos processos tradicionais e nas metodologias ágeis bem como nos trabalhos preexistentes sobre o tema. Ressaltou-se, que essa técnica pode ser adotada como uma aliada incontestável da melhoria da obtenção e da validação dos requisitos, bem como da qualidade dos incrementos.

Quanto à técnica não é interesse desta pesquisa concluir que o seu uso fará com que projetos tenham a idéia de “zero defeito” (Pressman, 2011), sentimento que não deve ser encorajado. Entretanto, deve ser almejado, seja em cenários de desenvolvimento de software onde há uso do Scrum, como também em métodos dito tradicionais, como no caso do ciclo de vida em Cascata.

Confirma Schwaber (2009) e Pressman (2011) que os processos e metodologias devem buscar agregar o melhor de suas abordagens de forma a construir produtos que agreguem valor final ao usuário. Para tanto, nesse estudo foi apresentada a técnica de prototipagem como uma aliada nestes intentos, podendo ser usada nos momentos mais inicias da construção de software como forma de mitigar o aparecimento de defeitos nas fases finais do processo de construção.

Fora firmemente ressaltado, quando revistos os processos tradicionais de construção de software na seção 2, que a prototipação é comumente conhecida por auxiliar os *stakeholders* para a elicitação de requisitos em fases inicias. Bem como, analisada a sua realidade ágil em modelos ágeis, bem evidentes na subseção 2.6 *Trabalhos relacionados*. Mesmo que a abordagem tenha evoluído para uma perspectiva suscetível a mudanças, a prototipação é, no entanto, vista como integrante da fase inicial de construção e não levada a cabo quando os incrementos estão sendo construídos.

Também na seção 2 desse estudo fora observado os processos do método Scrum e sua orientação à mudança, dados que são novamente revisitados na seção 4, que apresentou o caso de sucesso da empresa “ABC”, assim como uma rápida análise dos fatores de sucesso, bem como a demonstração quantitativa dos benefícios obtidos na seção 5.

Ficou-se possível então assumir, após todas as ponderações desta pesquisa, que Scrum é um ambiente ágil, focalizado na mudança e na prototipagem, no contexto da organização objeto desse estudo. Como salientada, é uma técnica madura e robusta, que pode ser usada juntamente a todas as cerimônias do Scrum, sem haver efeitos danosos a qualidade dos artefatos finais. Ao contrário, a técnica mostrou possibilitar a entrega de produtos mais confiáveis e com maior qualidade ao cliente, nesse contexto.

Pretendeu-se com esta pesquisa evidenciar uma técnica que pode ser aplicada e replicada em qualquer organização, que queira aumentar a qualidade dos artefatos entregues a seus clientes, criando-se mais uma barreira de prevenção, para que mais erros não se perpetuem pelo sistema.

É também interesse dessa pesquisa desmistificar a obrigatoriedade da prototipação, como somente uma técnica para uso nas fases inicias do processo de construção de software, fato que é comumente divulgado pela comunidade acadêmica (Pressman,2011; Sommerville, 2011; Soares, 2017; Zemel, 2011).

Demonstra-se que a mesma pode ser empregada quando na existência de um produto de software, como forma de melhorá-lo, e ser mais um fator de consequente aumento da qualidade do produto.

Como pesquisas futuras, a partir dos resultados desse trabalho, sugere-se propor uma padronização para o processo de prototipação, que possa ser seguido em outras empresas como um modelo sendo necessárias, no entanto, novas observações levando-se em conta outras realidades organizacionais. Todavia, foi visualizado através da

observação do uso da técnica de prototipação na empresa “ABC” nos tópicos da seção 5, formas possíveis de implantação, porém não definitivas.

Destaca-se, por fim, que o resultado deste trabalho não pode ser encarrado como conclusivo ou mesmo seus valores apresentados como definitivos. Todavia é de interesse que trabalhos e estudos futuros sejam acrescidos e empregados quanto à realidade benéfica da prototipagem em uso em outras metodologias ágeis, bem como no Scrum de forma mais aprofundada, possibilitando a apresentação de um modelo padronizado que possa ser moldado para qualquer abordagem de desenvolvimento, de forma a ser uma ferramenta fixa e eficaz na melhoria da qualidade de software.

Referências

Bartié, A. (2002) “Garantia da qualidade de software - As melhores práticas de engenharia de software aplicadas à sua empresa”, Editora ELSEVIER, 13ª edição.

Borghi, R. A. (2015) “Elicitação de requisitos com Scrum Aplicada no início do projeto e na definição da solução do sistema”. Scrum Alliance.

Chammas, A., Quaresma, M. (2013) “Um enfoque ergonômico sobre a metodologia de design de interfaces digitais para dispositivos móveis”, Arcos Design.

Cockburn, A. (2002) “Agile software development. Boston”, Addison-Wesley.

Dolci, D. B. (2011) “Teorizando as Práticas dos Métodos Ágeis no Desenvolvimento de Software visando ao processo de Inovação em empresas”, FURG. Instituto de ciências econômicas, administrativas e Contábeis.

Lakatos, E. M. (2010) “Fundamentos de metodologia científica”, 7 ª edição, Atlas.

Marconi, M. A. (2011) “Metodologia científica”, Atlas, 6 ª edição.

Molinari, L. (2012) “Testes de Software: Produzindo sistemas melhores e mais confiáveis”, Érica, 4ª edição.

Pham, A.; Pham, P. (2012) “Scrum em Ação: gerenciamento e desenvolvimento ágil de projetos de software”, Novatec.

Pressman, R. S. (2011) “Engenharia de Software - uma abordagem profissional, AMGH Editora, 7ª edição.

Rios, E. (2012) “Base de conhecimento em teste de software”, Martins Editora, 3ª edição.

Saccol, A. (2012) “Metodologia de pesquisa em administração: Uma abordagem prática”, Editora Unisinos

Sano, S. (2012) “Metodologia de desenvolvimento de sistemas prescritivo x ágil”, Universidade presbiteriana Mackenzie.

Sbrocco, J. H. T. C.; Macedo, P. C. (2012) “Metodologias ágeis: engenharia de software sob medida”, Érica, 1. ed..

Schwaber, K. (2009) “Guia do SCRUM”.

Silva, A. F., Silveira, F. R. V. (2016) “Uma abordagem sobre engenharia de requisitos para metodologia ágil SCRUM”, IX Encontro de Computação. ENUMCOMP

Soares, B. C. (2017) “Requisitos para utilização de prototipagem evolutiva nos processos de desenvolvimento de software para web”, Universidade Federal de Minas Gerais (UFMG)."

Sommerville, I. (2011) “Engenharia de Software”, Person Editora, 9ª edição.

Zemel T. (2011) “Wireframes para web: guia completo de desenvolvimento”. DPW. Disponível em: http://desenvolvimentoparaweb.com/ux/wireframe-web-guia-completo/