# UMA ANÁLISE DA UTILIZAÇÃO DE METODOLOGIAS ÁGEIS ENVOLVENDO O DESENVOLVIMENTO DE *SAFETY-CRITICAL* SOFTWARE

**Luckeciano Carvalho Melo**
Instituto Tecnológico de Aeronáutica
H8A, apto. 113, CTA
122280-460 – São José dos Campos/SP
Bolsista PIBIC-CNPq
luckeciano@gmail.com

**Cassiano Monteiro**
Instituto Tecnológico de Aeronáutica
Rua João Teixeira Neto, 102 – apto 702 – Pq. Res. Aquarius
12246-160 – São José dos Campos/SP
cassianomonteiro@gmail.com

**Adilson Marques da Cunha**
Instituto Tecnológico de Aeronátuica
Divisão de Ciência da Computação
Praça Marechal Eduardo Gomes, 50
12229-900 – São José dos Campos/SP
cunha.adilsonmarques2@gmail.com

***Resumo.*** *O crescente uso de tecnologias digitais no mundo contemporâneo gerou uma larga demanda de softwares. Nos últimos anos, o desenvolvimento de softwares vem sofrendo mudanças no processo de transição de metodologias tradicionais para metodologias ágeis. Contudo, os chamados softwares safety-critical possuem o desenvolvimento mais delicado, com requisitos mais formalizados e rígidos. Objetiva-se, portanto, elaborar propostas ágeis para os processos de desenvolvimento de softwares desta natureza. Para tal, foi analisado o conjunto de recomendações de desenvolvimento de software aeronáutico contido na DO-178C, e criaram-se propostas de utilização de metodologias ágeis em seu atendimento. Com isto, conclui-se que é possível a utilização destas metodologias no desenvolvimento de software safety-critical de maneira satisfatória.*

*Palavras chave: Metodologias Ágeis, Software Safety-Critical, DO-178C, Engenharia de Software.*

## 1. Introdução

O crescente uso de tecnologias digitais no mundo contemporâneo gerou uma larga demanda de softwares. Estes permeiam pelas mais diversas áreas da produção humana e, portanto, tornam-se essenciais para o funcionamento da sociedade. Softwares são abstratos e intangíveis, uma vez que não dependem da propriedade dos materiais, não se governam por leis físicas e nem por processos de manufatura. Deste modo, exigem diferentes condutas e técnicas de design e produção (Sommerville, 2011).

A Engenharia de Software é uma área que lida com o desenvolvimento, a validação e a evolução do software. Seus principais desafios giram em torno de gerenciar a diversidade crescente, o atendimento de prazos de entrega e o desenvolvimento de software confiável.

Nos últimos anos, o desenvolvimento de software vem sofrendo mudanças drásticas no processo de transição de metodologias tradicionais para metodologias ágeis, evoluindo de um modelo baseado em escopo fechado (envolvendo desenvolvimentos em cascata) para um modelo baseado em escopo aberto (envolvendo desenvolvimentos iterativos, incrementais e ágeis) (Formulário PIBIC, 2014). Uma das principais características das metodologias ágeis tem sido a melhoria na aceitação e gestão de mudanças.

Contudo, há algumas áreas do desenvolvimento de software que veem limitações ao uso do desenvolvimento ágil. Nesta pesquisa serão tratados os softwares *safety-critical*, aqueles cujas falhas podem resultar em danos humanos ou econômicos (Turk, France, Rumpe, 2002). As recomendações para o desenvolvimento de softwares *safety-critical* contêm regras de como os seus requisitos devem ser representados (Habli, 2014). Os mecanismos de controle de qualidade apoiados por processos ágeis atuais, entretanto, não têm se mostrado suficientes para assegurar aos usuários que o produto é seguro.

Voltando-se para esse escopo definido de problemas, este artigo visa a elaboração de propostas de uso de metodologias ágeis em softwares *safety-critical*, com foco na indústria aeronáutica, de modo a respeitar as recomendações de engenharia e todo o formalismo de requisitos comuns às indústrias que operam com este tipo de software e que, por fim, assegurem um produto final seguro.

,

## 2. Materiais e Métodos

Na primeira etapa do trabalho, realizou-se uma pesquisa literária para obtenção de conhecimento sobre Engenharia de Software e as principais fases tradicionais de produção de um software. Posteriormente, realizou-se outra pesquisa para entendimento da cultura ágil e seus principais princípios. Após esta etapa do trabalho, foi realizado um levantamento bibliográfico sobre as duas principais metodologias abordadas: XP e Scrum. Por fim, realizou-se também uma análise de mercado para levantamento dos dados mais recentes sobre a adoção destas metodologias.

Na segunda etapa, realizou-se um estudo sobre uma recomendação específica da indústria aeronáutica: a DO-178C (DO-178C, 2011). Embora no planejamento inicial constasse o estudo de recomendações de outras indústrias de software *safety-critical*, optou-se por reduzir o escopo a fim de uma análise mais detalhada do processo de desenvolvimento de software para esta indústria. Posteriormente, elaborou-se propostas do uso das Metodologias Ágeis para os objetivos e atividades que compõem os principais processos descritos no documento.

Dessa forma, ao final deste trabalho, dispõe-se de conhecimentos e prática nos aspectos explorados durante a pesquisa. Dentre eles destacam-se: os princípios de Engenharia de Software e de softwares embarcados (Sommerville, 2011); os principais modelos de gestão de projetos de softwares (tradicionais e ágeis) (Beyer, 2010); os princípios da cultura ágil e suas técnicas; bem como as principais metodologias ágeis utilizadas no mercado (Scrum e XP) (Warden, 2008; Rubin, 2013); o processo de desenvolvimento de software para a indústria aeronáutica e a DO-178C. Além disso, dispõe-se da elaboração de propostas de uso das Metodologias Ágeis para softwares *safety-critical*, propósito final deste trabalho de Iniciação Científica.

## 3. Resultados e Discussão

### 3.1 A Cultura Ágil

A cultura ágil não diz respeito a desenvolver um software, o mais rápido possível. Na verdade, se trata de responder, da melhor maneira, aos imprevistos associados ao mercado de negócios, de modo que os custos de modificação do escopo original (e, portanto, de retrabalho) sejam os menores possíveis.

As metodologias ágeis visam, portanto, se adaptarem ao escopo variável. Para isto, estas abordagens modificam a maneira como o processo de desenvolvimento ocorre. Ao invés de um grande processo em cascata, como visto na Fig. 1, o desenvolvimento acontece em pequenas iterações, ilustrado na Fig. 2. Ao final, há sempre um produto "entregável", no sentido de que por mais simples, todas as suas funcionalidades encontram-se completamente implementadas. A ideia destas pequenas iterações encontra-se em obter um *feedback* rápido e constante, além da possibilidade de modificação do projeto original, adicionando ou retirando outras funcionalidades. Deste modo, o cliente pode acompanhar e direcionar todo o processo, a fim de que o produto fique mais parecido com sua proposta inicial.

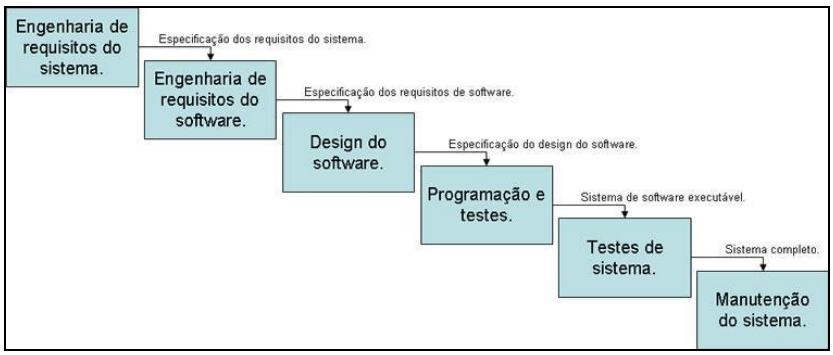


Figura 1: Processo de desenvolvimento em cascata (Sommerville, 2011)

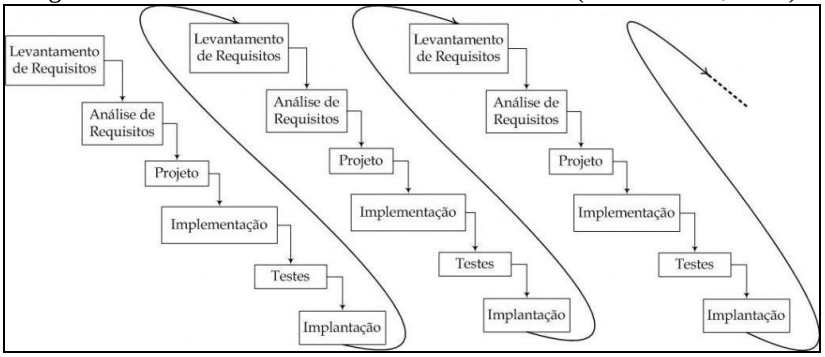

Figura 2: Processo iterativo (ágil) de desenvolvimento (Sommerville, 2011)

Para definir os princípios da cultura ágil, houve o Manifesto Ágil, uma declaração de princípios que fundamentam esse tipo de desenvolvimento, contendo quatro valores fundamentais (Agile Manifesto, 2001) e propõe que (Beyer, 2010; Warden, 2008):

- Há apenas um único time e o cliente encontra-se nele;
- Não vale a pena gastar muito tempo em design e este deve ser simples;
- Comunicação rápida vem antes de documentação extensa;
- Pequenas iterações são ideais para o processo de desenvolvimento;
- O *feedback* tenha rapidez e continuidade; e
- O trabalho deve ser energizado e o local de trabalho informativo.

### 3.2 A Constituição do Time de Scrum e XP

Inicialmente, uma das principais características do projeto ágil se trata do engajamento do cliente no processo de desenvolvimento do software. Logo, o **cliente** faz parte como integrante do time Scrum e do time XP. Este deve fornecer o *feedback* do produto ao final das iterações, de modo que o produto evolua da maneira que ele queira, levando em consideração as condições do mercado que atua.

O segundo componente, tanto do time Scrum como do time XP, chama-se **product owner,** responsável por fazer a ligação entre o mundo de desenvolvimento e zela pela imagem do produto. Na prática, isto significa documentar esta imagem, entrar em contato com os **stakeholders** (investidores, clientes externos), incorporar o *feedback*, gerar as funcionalidades, fazer a lista de prioridades nas reuniões de planejamento, direcionar o time de desenvolvimento, revisar o progresso, lidar com as demos de iterações e envolver os clientes.

Um elemento "opcional" (no sentido de que o próprio cliente ou product owner pode se encarregar de fazer seu trabalho) identifica-se como **domain expert.** Este guia o conteúdo técnico do software, de modo que suas principais regras sejam respeitadas. Outro elemento opcional chama-se **analista de negócios**, responsável por esclarecer as necessidades do cliente para o product owner, refinando-as. Além disto, ajudam programadores a expressar detalhes técnicos, em termos de negócios (Warden, 2008).

O componente responsável pela implementação técnica das funcionalidades é o **time de desenvolvimento**. Este time compõe-se por:

- Programadores - responsáveis pela implementação do código e do comportamento;
- Designers de Interação - responsáveis pela interface gráfica e experiência do usuário com o software;
- Designer e arquitetos de software - encarregados da criação e estrutura do projeto;
- Especialistas técnicos - encarregados por alguma implementação mais complexa; e
- *Testers* - encarregados de ajudar a produzir um produto de qualidade, por meio da criação de testes.

Por último, há o papel de **ScrumMaster** (no Scrum) ou **Coach** (termo mais utilizado em XP). Sua função gira em torno de fazer com que o time abrace os princípios, valores e práticas da metodologia ágil utilizada (Rubin, 2013), de modo a manter o uso dela durante todo o processo.

Note-se que as equipes das duas metodologias têm bastante similaridade. A quantidade de membros da equipe depende do projeto. As metodologias ágeis, em geral, pregam também muito a versatilidade dos membros do time, justamente para que se adequem a pequenos projetos. Logo, um membro pode desempenhar várias dessas funções anteriormente descritas.

### 3.3 O Processo de Desenvolvimento em Scrum e XP

No processo de desenvolvimento em Scrum ou XP, uma série de etapas formam iterações repetidas diversas vezes até o produto final. No Scrum, estas iterações chamam-se **sprints**.

Inicialmente, o product owner se reúne com o cliente e outros stakeholders para conhecer os requisitos iniciais do software desejado. Estes apresentam suas necessidades, de acordo com seus objetivos de mercado.

Em uma primeira reunião com toda a equipe, o product owner descreve em detalhes os requisitos dos stakeholders e, juntamente com o time, escrevem as funcionalidades em **Product Backlog Items - PBIs** (ou, no XP, **User Stories**). Então, o time monta seu Product Backlog inicial (ou User Scenario). Esta atividade comumente chama-se de ***grooming*** (vale ressaltar que, em algumas equipes, o *grooming* realiza-se somente pelo product owner).

Posteriormente, em uma outra reunião, a equipe começa a discutir sobre as implementações referentes às próximas iterações, na chamada **Release Planning** (no Scrum, ainda existe o **Sprint Planning**, que trata especificamente do sprint em questão). Nela, ocorre o **Planning Game** (ou **Planning Poker**): o product owner descreve os PBIs para os desenvolvedores. Tais elementos dividem-se em **tarefas** e cada membro da sua equipe avalia o esforço

aparentemente despendido para a implementação, por meio de algum sistema de valores, denominados **Scrum Points** (ou, no XP, **Story Points**). Então, direciona-se as tarefas aos desenvolvedores, de modo que encaixem no período de tempo do sprint, por meio de estimativas e da velocidade da equipe (Rubin, 2013). Além disto, priorizam-se estes PBIs de acordo com seu *bussiness value* (identificado pelo product owner) e de seu esforço de implementação (identificado pelos desenvolvedores nos Scrum/Story points).

O sprint, então, entra em execução. Os desenvolvedores começam a implementar as tarefas ao produto, refinando-o. Todos os dias há uma reunião de quinze minutos, chamada **Standup Meeting** (conhecida no Scrum como **Daily Scrum**). Seu objetivo é integrar os trabalhos de todos na equipe, fazer algumas modificações na execução da iteração, e atualizar o *dashboard* do time.

Ao final do sprint, a equipe revisa todo o trabalho realizado e aponta o que realmente terminou (a definição de terminado nessas abordagens - ou "*Done Done*" - tem contornos rígidos, para assegurar qualidade no produto e organização em todo processo de desenvolvimento). Por fim, fazem uma retrospectiva da iteração, para analisar todo processo de produção, propiciando melhorias em sua execução.

Esse processo (desde a Release Planning) repete várias vezes, até o fim da Product Backlog. Ela, por sua vez, pode constantemente sofrer alterações, de acordo com os requisitos do cliente, desde que não interfira no sprint em execução.

Observe que os processos de desenvolvimento em Scrum e em XP têm bastante semelhanças. A diferença encontra-se, na sua maior parte, nos termos utilizados para cada etapa do processo.

### 3.4 Scrum *versus* XP

Até este momento, todo o conteúdo apresentado mostra-se bastante semelhante entre as metodologias. Porém, há alguns quesitos que as diferenciam.

O Scrum se trata de uma metodologia com técnicas mais simples de implementação. Além disto, conta com integrantes interfuncionais (o nome "Scrum", original do Rugby, vem disto), ou seja, que podem resolver quaisquer tarefas. Por sua maior "versatilidade", o Scrum não tem utilização apenas em desenvolvimento ágil de software, mas em gerenciamento de projetos em geral, pois independe da área do conhecimento deste. Além disto, por ser "simples", se adapta desde pequenos times até aqueles de grande tamanho (neste caso, torna-se comum a formação de múltiplos times e o desenvolvimento do "*Scrum de Scrums*").

Por outro lado, o XP foca no desenvolvimento. Esta metodologia possui várias práticas e princípios que auxiliam e melhoram a produtividade, no que se refere a produção de softwares. Dentre elas, têm-se:

- **Programação em pares -** *U*ma técnica de XP na qual o desenvolvimento de uma tarefa fica a cargo de um par de desenvolvedores do time. Enquanto um tem responsabilidade de escrever o código, o outro faz uma revisão e pensando em propostas de como melhorar o código. A ideia trata de aumentar a produtividade evitando retrabalho e o nivelamento técnico da equipe;
- ***Test-Driven Development* -** Também conhecido como TDD, trata-se de um rápido ciclo de testes, criação de código e refatoração (Warden, 2008). Esta técnica funciona da seguinte maneira: um par de programadores, quando estiver adicionando uma nova funcionalidade, criará pequenos testes para saber se o código posteriormente escrito atende aos requisitos da tarefa em questão. Estudos mostram que TDD reduz substancialmente a incidência de defeitos (Pressman, 2006) e trata-se de uma das principais técnicas do XP;
- **Propriedade coletiva de código -** Em XP, um princípio bastante importante trata de considerar o código desenvolvido uma propriedade coletiva, ou seja, todos os desenvolvedores são responsáveis por mantê-lo sempre o melhor possível; e
- **Refatoração -** Prática simples. Um processo de mudança de design do código sem modificar seu funcionamento - *o que* ele faz continua o mesmo, mas *como* isto é feito, muda (Warden, 2008).

Além disso, também se têm outros princípios como manter padrões de código e uma linguagem acessível a todos da equipe, *build* de dez minutos e integração contínua. Todos visam o melhoramento da produtividade da equipe e da qualidade do software produzido.

Por fim, tendo em vista toda a constituição de um time Scrum/XP, o processo de desenvolvimento das atividades e as principais diferenças entre as duas metodologias, chega-se a conclusão de que Scrum e XP se complementam, uma vez que, enquanto o Scrum é mais versátil e prático para gerenciamento de projetos, o XP possui técnicas que melhoram, substancialmente, o nível da equipe e do produto.

### 3.5 Análise de Mercado

Para avaliar a possibilidade de adoção de práticas ágeis em processos de desenvolvimento de software safety-critical, e seu impacto na indústria, é interessante realizar uma análise de mercado através de pesquisas de alguns fatores, relacionados à adoção das metodologias ágeis. Com isso, esta análise teve por objetivo coletar e interpretar

dados de maneira a realizar uma avaliação da adoção destas abordagens em diferentes indústrias (levando em conta aspectos de produtividade e qualidade do produto desenvolvido).

A primeira pesquisa trata da adoção das metodologias ágeis por parte de empresas de TI e as dificuldades encontradas para tal. Estes dados foram retirados de questionários elaborados em fevereiro e março de 2014 (Ambysoft, 2014).

Note-se que, quando perguntadas em que condição a empresa se encontrava quanto à agilidade:

- 14,91% encontravam-se começando a utilizar as técnicas ágeis;
- 25,44% aplicavam metodologias ágeis, em menos da metade de seus times;
- 49,12% aplicavam metodologias ágeis, em mais da metade de seus times; e
- 10,53% não se utilizavam de metodologias ágeis.

Além disso, cerca de 76% das empresas entrevistadas consideram difícil ou muito difícil mudar a cultura de negócios para amparar o uso de metodologias ágeis e cerca de 60% consideram também a cultura de TI um problema difícil ou muito difícil de mudar.

Por outro lado, essas mesmas empresas afirmam (com índices acima de 70%) que tem muita importância não só mudar essa cultura de negócios e de TI, como também adotar práticas como integração contínua e TDD.

Outra pesquisa diz respeito a comportamento de adoção dos principais processos de desenvolvimento de software, entre os anos de 2012 a 2014:

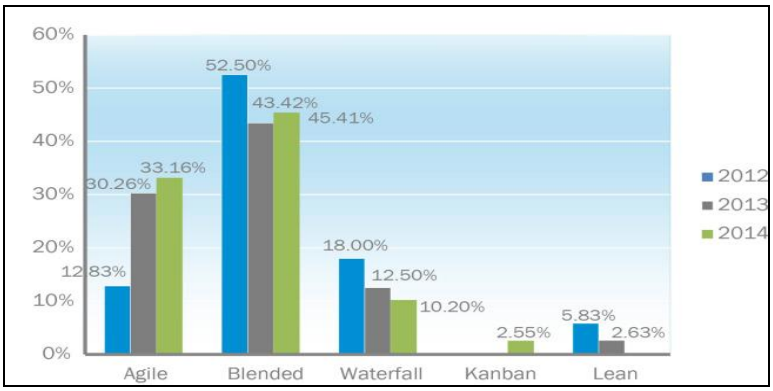


Figura 3: Adoção das metodologias de produção de software entre 2012 e 2014 (Actuation Consulting, 2015).

Note-se que a categoria "*Blended*" domina. Trata-se de um mesclado entre técnicas de produção cascata (*Waterfall*) e práticas ágeis.

A Fig. 3 mostra que, durante esses anos, há um grande crescimento no uso de metodologias ágeis, em contraste à queda do tradicional. Logo, nos últimos tempos se observa uma constante e gradual transição entre o mundo tradicional e o mundo ágil.

Por fim, a principal pesquisa sobre metodologias ágeis refere-se a 8$^{th}$ *State of Agile*. Seus dados mais atuais datam de 2013.

A pesquisa mostra que, de todos as empresas questionadas, cerca de 88% praticam, de algum modo, o desenvolvimento ágil, em 2012, 84% e, em 2011, 80%. Além disto, mais da metade delas já se utiliza de tal cultura de 2 a 5 anos.

De acordo com essa pesquisa, 73% das empresas considera que, após iniciadas as atividades com abordagem ágil, seus produtos são entregues com sucesso aos clientes mais rapidamente.

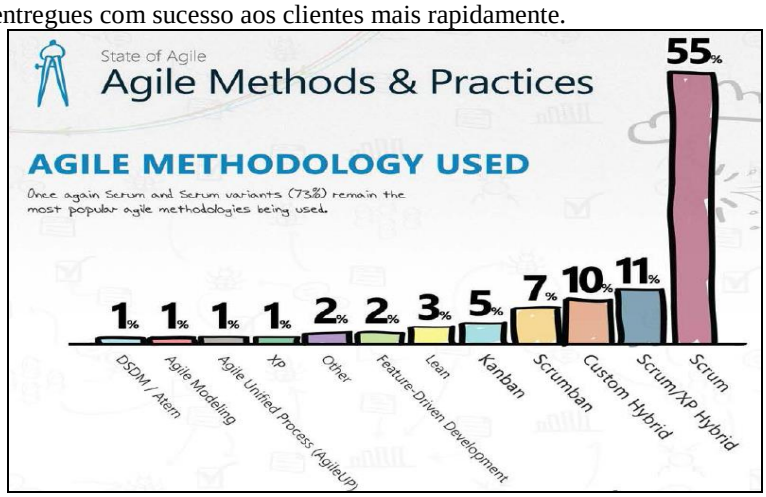

Figura 4: As principais metodologias ágeis do mercado atual (8th State of Agile, 2014).

Da Fig. 4, pode-se tirar uma das maiores conclusões deste trabalho. O Scrum "puro" enquadra-se como a abordagem mais utilizada em nível mundial, com 55% dos entrevistados. Isto se deve a sua versatilidade quanto ao tamanho da equipe e a área de uso, e também a sua simplicidade de implementação. Em segundo lugar, vem um híbrido de XP e Scrum. O XP "puro" detém apenas 1% das empresas.

Uma nova análise de 2014 tem como base a *9th State of Agile Survey*. Esta nova pesquisa propicia uma base comparativa entre os dois últimos anos em empresas que utilizam as Metodologias Ágeis.

De acordo com essa pesquisa, 94% das empresas que responderam utilizam práticas ágeis no seu trabalho. No ano anterior, este índice era de 88%, o que mostra que a aderência aos métodos ainda cresce. Quanto ao tempo de uso, essa pesquisa revelou que, em 2014:

- 24% das empresas possuem mais de cinco anos de práticas ágeis, contra 19% do ano anterior;
- 32% possuem de 3 a 5 anos, contra 33% de 2013;
- 29% possuem de 1 a 2 anos, contra 40% de 2013; e
- 15% possuem menos de um ano, contra 8% do ano anterior.

Por outro lado, o número organizações que possuem a maioria de seus projetos ágeis diminuiu. Em 2013, eram 52% das entrevistadas, contra apenas 45% em 2014. O números mais elevado de empresas adeptas de métodos ágeis, em conjunto com os números também maiores de empresas pouco experientes em práticas ágeis demonstram a entrada de novas empresas no mercado ágil. Isto pode ser verificado pelo fato de que a porcentagem de projetos ágeis por empresa diminuiu, uma vez que as empresas menos experientes ainda encontram-se em processo de transição dos métodos tradicionais para práticas ágeis.

A Fig. 5 mostra a distribuição das empresas nas Metodologias Ágeis utilizadas em 2014, analogamente à Fig. 6:

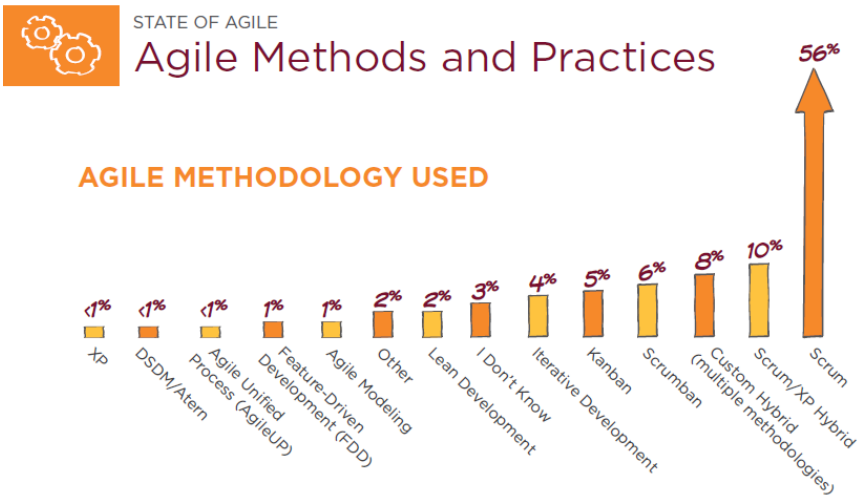


Figura 5: As principais Metodologias do mercado de 2014 (9th State of Agile, 2015).

Percebe-se, da Fig. 8, que o comportamento de crescimento sofreu poucas mudanças. O Scrum continua sendo a principal técnica ágil, seguida do híbrido entre Scrum e XP.

### 3.6 DO-178C

A DO-178C é um documento de recomendações conhecido da indústria aeronáutica, e possui como objetivo fornecer orientações à produção de software para sistemas e equipamentos de bordo, os quais pretendam exercer sua função com um certo nível de confiabilidade de segurança em conformidade com determinados requisitos de aeronavegabilidade (DO-178C, 2011).

Seu escopo compõe-se de uma breve introdução de aspectos relacionados ao desenvolvimento de software e o ciclo de vida deste; dos processos do ciclo de vida do software com respectivos objetivos e atividades; uma visão geral do processo de certificação; dados do ciclo de vida; e considerações adicionais.

De acordo com a DO-178C, o processo de desenvolvimento de um software dessa natureza inicia-se após a fase de planejamento, no **Processo de Requisitos** do software. Este trabalha no desenvolvimento dos requisitos de alto-nível. Isto inclui requisitos funcionais, de desempenho e interface. Além destes, incluem-se, também, requisitos não-funcionais, particularmente importantes para sistemas críticos, tais como segurança, disponibilidade, robustez, entre outros. Os objetivos deste processo são o desenvolvimento dos requisitos de alto-nível e a definição e fornecimento dos requisitos de alto-nível derivados para os processos do sistema, incluindo o processo de avaliação de segurança do

sistema. O processo de requisitos conta como entrada os requisitos do sistema, a interface do hardware, o plano de desenvolvimento do software e as recomendações de requisitos.

Posteriormente, ocorre o **Processo de *Design*** do software. Este refina os requisitos de alto-nível com o objetivo de desenvolver uma arquitetura de software e requisitos de baixo-nível que possam ser utilizados na implementação do código. Estes, por sua vez, são fornecidos para os processos do sistema, incluindo o de avaliação de segurança. O processo tem como entrada os requisitos do software, o plano de desenvolvimento e os padrões de design do software. Seus resultados primários incluem a arquitetura do software e os requisitos de baixo-nível.

Após isso, acontece o **Processo de Desenvolvimento do Código** do software. Este objetiva a implementação dos requisitos de baixo nível, de acordo com a arquitetura previamente elaborada e, adicionalmente, com nível qualidade e tratamento de defeitos estabelecidos. O processo tem como resultado o código fonte.

Então, há o **Processo de Integração**. Encara-se em dois níveis: a integração de subsistemas de um software e a integração do software final com o hardware. A integração do software acontece quando várias equipes desenvolvem partes independentes do software simultaneamente e, ao fim, todas as partes se integram. Pelo fato de todas as partes não serem desenvolvidas em série, tomam-se algumas precauções O outro ponto de integração consiste em embarcar o software no hardware que o utilizará, isto é, integração hardware-software. Neste caso, o ponto principal continua sendo a fase de testes, que deve cobrir não simplesmente a prospecção de bugs do sistema ou testes de robustez, como também teste de dispositivos físicos, como o uso de baterias.

Em seguida, ocorre o **Processo de Verificação** do software. Este processo tem como objetivo e atividade a verificação do software. A verificação se define como uma avaliação técnica dos *outputs* dos processos anteriores (planejamento, desenvolvimento e até do mesmo). O processo em si não se baseia apenas em testes, pois estes, em geral, não expõem a ausência de erros. Portanto, o processo de verificação combina revisões, análises e testes.

Esse processo tem como propósito verificar:

- O desenvolvimento dos requisitos de alto e baixo nível de acordo com a arquitetura do software;
- Se o código fonte foi desenvolvido satisfazendo os requisitos e arquitetura; e
- Se o executável satisfaz os requisitos e é robusto para responder corretamente a condições anormais;

Finalmente, ocorre o **Processo de Gerenciamento de Configuração** do software, cujo objetivo é gerenciar e controlar as versões do software, bem como linhas de base (referência) e capacidade de reprodução do processo. Há também o **Processo de Garantia de Qualidade (*Quality Assurance - QA),*** que avalia o ciclo de vida do software e seus outputs, de forma a obter garantias de que os objetivos são satisfeitos, deficiências detectadas, mapeadas e resolvidas, e o produto e seu ciclo de vida estão em conformidade com os requisitos de certificação. Possui como objetivo fornecer convicção de que os processos do ciclo de desenvolvimento do software resultam em um produto em conformidade com os requisitos, através da garantia de que estes processos são desempenhados de acordo com os planos e padrões aprovados.

### 3.7 Proposta Ágil

Uma vez analisados os processos de desenvolvimento de software *safety-critical* de acordo com as recomendações e padrões que constam na DO-178C, analisou-se a integração dos princípios das Metodologias Ágeis nestes processos. Utilizam-se várias das técnicas ágeis em distintos processos, de modo que se possa inferir a capacidade de implementação destes métodos garantindo o nível de confiança na segurança conforme requisitos de aeronavegabilidade estabelecidos.

A princípio, o processo de desenvolvimento do software safety-critical seguirá o framework das Metodologias Ágeis: o desenvolvimento em sprints de curta duração definida previamente, a partir de User Stories planejadas pelo Product Owner e detalhadas pelo time de desenvolvimento, com reuniões de Sprint Review/Retrospective (para revisão do "entregável" e do processo) e Standup Meetings, para micro replanejamento. Neste cenário, oss processos da DO-178C não ocorrem em cascata, mas sim paralelamente durante o decorrer dos sprints.

As propostas elaboradas, por outro lado, partem dos princípios ágeis e como estes se integram – ou não - nos processos analisados da DO-178C. Os resultados, por fim, encontram-se consolidados e tabelados no Anexo 1.

Todo desenvolvimento começa no *Sprint Zero*. Trata-se de um planejamento independente e fase de preparação que se usa para acelerar o ritmo para o desenvolvimento de seu aplicativo de software e estabelecer a viabilidade do projeto (Static Architect, 2010). As principais características do Sprint Zero são (Scrum Alliance, 2014): a elicitação dos requisitos iniciais (o **Processo de Requisitos** começa aqui), bem como estabelecimento do *Scrum Master* e da "*definition of done"* para cada requisito. Esta última deve ser bastante detalhada, de maneira a seguir todas as recomendações e padrões mencionados na DO-178C. Ocorre também o estabelecimento dos primeiros objetivos e restrições de *Stakeholders*, e do plano de entrega de *releases* e/ou protótipos (início do **Processo de Planejamento** do software).

Posteriormente a esta etapa, começam os sprints de desenvolvimento. Nestes, todos os processos ocorrem paralelamente: o **Processo de Requisitos** gerará novas *User Stories* (para requisitos funcionais) a partir dos requisitos de alto-nível, as quais serão detalhadas pela equipe de desenvolvimento, destacando-se a *definition of done* para cada *User Story*. Para requisitos não-funcionais, podem ser criadas "*System Stories*", a fim de detalhar os requisitos deste tipo. Uma *System Story* é um tipo análogo à *User* Story, porém não tratando especificamente de requisitos relacionados a valor de negócio, mas a algo ligado à arquitetura e a questões de qualidade do sistema (Stack Overflow, 2010).

Enquanto isso, o **Processo de Design** analisará as *User Stories* já definidas, a fim de desenvolver uma arquitetura de software simples, robusta, flexível e incremental, bem como gerar os requisitos de baixo-nível que serão representados por tarefas de implementação que compões uma *User Story*. O **Processo de Desenvolvimento do Código**, por sua vez, implementará os requisitos de baixo nível representados pelas tarefas, resultando no código fonte de acordo com a arquitetura previamente estabelecida, e utilizando técnicas ágeis para desenvolvimento de código com qualidade e velocidade, tais como *pair programming*, integração contínua, *build* automatizada, *test-driven development*, controle de versão e outros. Já o **Processo de Integração** conectará os subsistemas do software, para sua interação seja corretamente verificada, e também integrará o software final ao hardware desejado. Para isso, utilizam-se, dos Métodos Ágeis, ferramentas automatizadas de integração contínua.

Paralelamente, o **Processo de Verificação** ocorre analisando a conformidade entre requisitos de alto nível e *User Stories*, e entre requisitos de baixo nível e tarefas de implementação. Além disso, detalha e revisa *User Stories* com métricas de verificabilidade (que podem ser implementadas com build automatizada). O código pode sofrer refatoração incremental neste processo durante o sprint. Verifica-se o produto por meio de testes realizados durante o Sprint, sendo revisado no *Sprint Review*. Por outro lado, o processo sofre uma análise e revisão (para garantir conformidade com recomendações e padrões) no *Sprint Retrospective.* Por fim, discutem-se sobre mudanças urgentes durante o *Standup Meeting* e encara-se retrabalho dos processos de desenvolvimento do software como novas *User Stories* para implementação nos futuros sprints.

No caso do **Processo de Gerenciamento de Configuração**, a ideia básica gira em torno do uso de ferramentas para controle das versões e rastreabilidade de *User Stories*, para mapeamento e correção de problemas, bem como controle de mudanças. Para o **Processo de Garantia de Qualidade**, a garantia de que o processo de desenvolvimento tem conformidade com as recomendações e padrões da DO-178C se dá por meio de criação de *User Stories* com critérios de aceitação e do cumprimento da *definition of done* pré-estabelecidos. Além disso, este processo de Garantia de Qualidade opera no cumprimento do processo ágil, por meio do papel do *Scrum Master* e do *Standup Meeting* (*Daily Scrum*). Fases como *Sprint Review* e *Sprint Retrospective* integram este processo garantindo tanto a qualidade do produto quanto do processo, respectivamente. Por fim, este também coordena o ambiente de desenvolvimento, desde o *Sprint Zero* e durante todos os sprints.

## 4. Conclusões

As metodologias ágeis têm possibilitado avanços na produtividade das empresas e na qualidade do produto final.

Foca-se, neste projeto, na elaboração de propostas de metodologias ágeis ao desenvolvimento de softwares *safety-critical*, que ainda enfrentam certos problemas para adequarem o formalismo de seus requisitos ao mundo ágil.

Este trabalho de Iniciação Científica propiciou uma pesquisa literária sobre as principais abordagens ágeis do mercado, com uma posterior comparação e análise de mercado, bem como a análise do documento DO-178C como recomendações para desenvolvimento de software *safety-critical* para a indústria aeronáutica, expondo padrões do processo de desenvolvimento. Além disso, propiciou a integração entre as práticas ágeis e a indústria *safety-critical*, por meio de elaboração de propostas de adoção de práticas ágeis neste tipo de desenvolvimento.

Conclui-se, portanto, que é possível utilizar-se dos mais variados princípios das Metodologias Ágeis no desenvolvimento de software *safety-critical*, como um conjunto de ferramentas auxiliares e potencialmente produtivas, garantindo não somente a conformidade com os padrões de aeronavegabilidade, como também fomentando uma maior qualidade do software produzido.

## 5. Agradecimentos

Agradeço às pessoas ligadas diretamente a mim, pelo apoio durante esse período inicial do projeto.

Ao Prof. Dr. Adilson Marques da Cunha e ao Aluno de Pós-Graduação do ITA Cassiano Monteiro, pela orientação durante essa parte inicial do trabalho. Foram essas pessoas que me deram a oportunidade e o direcionamento necessários para que eu pudesse aprender sobre Metodologias Ágeis, sua cultura e principais práticas.

Agradeço também ao CNPq que, vinculado ao Ministério da Ciência e Tecnologia (MCT), apoia a pesquisa brasileira e contribui diretamente para a formação de jovens

Pesquisadores, investindo e promovendo o aumento da produção de conhecimento e gerando novas oportunidades para universitários desejosos em iniciar uma vida de pesquisa e desenvolvimento nas diversas áreas do conhecimento.

Agradeço, por fim, àquelas pessoas que fazem o mundo se superar a cada dia.

## 6. Referências


8th Anual State of Agile Survey. 2014, VersionOne, Inc. All Rights Reserved.

9th Anual State of Agile Survey. 2015, VersionOne, Inc. All Rights Reserved.

Agile Manifesto. Disponível em < www.agilemanifesto.org > Acesso em 26 de Agosto de 2015.

Agile Survey: Results of January 2014. Disponível em <www.ambysoft.com/surveys/agileJanuary2014.html#Results > Acesso em 28 de fevereiro de 2015.

Beyer, Hugh, 2010. User-Centered Agile Methods, 1st ed., Morgan & Claypool.

DO-178C: Software Considerations in Airborne Systems and Equipament Certification. RTCA Inc., 2011.

Formulário de Inscrição do bolsista PIBIC 2014/2015.

Ibrahim Habli, Andrew Era. Formalism of Requirements for Safety-Critical Software: Where Does the Benefit Come From? Department of Computer Science, University of York, 2014.

Pressman, Roger S. Engenharia de Software. 6ª ed. Rio de Janeiro. McGraw-Hill, 2006.

Rubin, Kenneth S., 2013. Essential Scrum: a practical guide to the most popular agile process, 1st ed., Peason Education.

Saiedian, Janzen, David, and Hossein. 2005. "Test-Driven Development: Concepts, Taxonomy, and Future Direction." IEEE Computer Society. 38(9): 43-50.

Product Develipment Process Adoption Can Past Predict Future. Disponível em < http://www.actuationconsulting.com/product-development-process-adoption-can-past-predict-future> Acesso em 28 de fevereiro de 2015.

Sommerville, Ian, 2011. Software Engineering, 9th ed., Pearson Education.

Sprint Zero: Where Your Software Project Begins. Disponível em <http://static.architech.ca/wp-content/uploads/2010/09/Sprint-Zero-Where-Your-Software-Project-Begins.pdf> Acesso em 4 de agosto de 2015.

State of IT Union 2014: Q2. Disponível em <http://www.ambysoft.com/surveys/stateOfITUnion2014Q2.html > Acesso em 28 de fevereiro de 2015.

System Stories for Agile Architeture. Disponível em: < http://stackoverflow.com/questions/1828057/system-stories-for-agile-architecture> Acesso em 5 de agosto de 2015.

The Idea and Rationale of Sprint Zero. Disponível em < https://www.scrumalliance.org/community/articles/2014/november/the-idea-and-rationale-of-sprint-zero> Acesso em 5 de agosto de 2015.

Turk, Dan. France, Robert. Rumpe, Bernhard. Limitations of Agile Software Processes. In: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, May 26-30, Alghero, Italy, pg. 43-46, 2002.

Warden, Shane, Shore, James, 2008. The Art of Agile Development, 1st ed., O'Reilly Media.


## 7. Anexo: Tabela de Propostas Ágeis

Tabela 1: Propostas de utilização de princípios ágeis no Processo de Planejamento da DO-178C

| ***Processo de Planejamento*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Planejamento do processo de desenvolvimento e elicitação de requisitos iniciais | *Sprint Zero* | Elicitação dos requisitos iniciais por *User Stories*, estabelecimento da *definition of done*, designação do *ScrumMaster*, planejamento de releases |

Tabela 2: Propostas de utilização de princípios ágeis no Processo de Requisitos da DO-178C

| ***Processo de Requisitos*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Análise dos requisitos de interface e sistema funcional | *Sprint Zero* | Criação de protótipos não-funcionais e exposição para o cliente |
| Especificação dos requisitos funcionais de alto-nível | *Product Backlog e User Stories* | Cada requisito funcional de alto-nível deve ser traduzido em uma *User Story* consistentes com a DO-178C |
| Especificação dos requisitos não funcionais de alto nível | *Product Backlog e System Stories* | Criação de *System Stories*, que tratam de questões ligadas ao comportamento do sistema |

| | | |
|---|---|---|
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Product Backlog e User Stories* | Erros e problemas do software são tratados por meio da criação de novas *User Stories* |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Sprint Review* | A incorporação do feedback para esclarecimento ou correção ocorre no *Sprint Review* |
| Requisitos de alto nível devem ser consistentes e verificáveis | *Definition of Done* | O estabelecimento da *definition of done* de cada *User Story* deve atender a DO-178C |

Tabela 3: Propostas de utilização de princípios ágeis no Processo de Design da DO-178C

| ***Processo de Design*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Especificação dos requisitos de baixo nível | *Product Backlog e User Stories* | Criação de tarefas de implementação a partir das *User Stories* |
| Requisitos de baixo nível devem ser consistentes e verificáveis | *Definition of Done* | O estabelecimento da *definition of done* deve atender a DO-178C e atender os padrões de design |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Product Backlog e User Stories* | Erros e problemas do software são tratados por meio da criação de novas *User Stories* |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Sprint Review* | A incorporação do feedback para esclarecimento ou correção ocorre no *Sprint Review* |
| Especificação da arquitetura de software | *Design Flexível e uso de Design Patterns* | Construção de uma arquitetura robusta por meio de um design flexível |
| Consistência entre subsistemas e interfaces | *Design Flexível através de Design Patterns e Test Driven Development* | *Test Driven Development* e *Design Patterns* podem ser utilizados como técnicas de design evolutivo através de refatoração, trazendo consistência entre interfaces do sistema. |

Tabela 4: Propostas de utilização de princípios ágeis no Processo de Desenvolvimento do Código da DO-178C

| ***Processo de Desenvolvimento do Código*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Desenvolvimento do código fonte pela implementação dos requisitos de baixo nível | *Product Backlog e User Stories* | As tarefas de implementação, que simbolizam os requisitos de baixo nível, são desenvolvidas |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Product Backlog e User Stories* | Erros e problemas do software são tratados por meio da criação de novas *User Stories* |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Sprint Review* | A incorporação do feedback para esclarecimento ou correção ocorre no *Sprint Review* |
| Desenvolvimento do código fonte | *Pair Programming* | O *pair programming* auxilia o desenvolvimento, aumentando a produtividade e qualidade do software, bem como nivelando tecnicamente o time |
| O código fonte deve estar de acordo com os padrões de código do software | *Ferramentas automatizadas (controle de versão, build automatizada, integração contínua)* | Tais ferramentas automatizam a verificação dos padrões de código |
| Fornecimento de configurações definidas e | *Ferramentas automatizadas* | O uso de ferramentas de controle de versão auxilia o gerenciamento de configurações |

| | | |
|---|---|---|
| controladas | *(controle de versão, build automatizada, integração contínua)* | |
| Desenvolvimento do código fonte pela implementação dos requisitos de baixo nível de acordo com a arquitetura do software | *Test-driven development* | Desenvolvimento do código por meio de testes previamente programados que verificam a implementação de cada requisito de baixo nível |

Tabela 5: Propostas de utilização de princípios ágeis no Processo de Integração da DO-178C

| ***Processo de Integração*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Inputs inadequados ou incorretos devem ser fornecidos para os processos vitais | *Ferramentas automatizadas (controle de versão, build automatizada, integração contínua) Sprint Review* | A verificação do processo de integração pode ocorrer por meio de ferramentas automatizadas a cada modificação de subsistema, informando quaisquer problemas aos responsáveis. A incorporação do feedback para esclarecimento ou correção ocorre no *Sprint Review* |

Tabela 6: Propostas de utilização de princípios ágeis no Processo de Verificação da DO-178C

| ***Processo de Verificação*** | | |
|---|---|---|
| **Objetivo/Descrição** | **Proposta Ágil** | **Descrição** |
| Garantia de que o ambiente de desenvolvimento do software foi fornecido como especificado | *Sprint Zero* | O *Sprint Zero* serve também para preparação do ambiente de desenvolvimento, o qual continua evoluindo durante todo o ciclo de vida do software |
| Satisfação dos critérios de transição nos processos de desenvolvimento | *Definition of Done* | A *definition of done* pode constar métricas relacionadas aos critérios de transição |
| Garantia do seguimento de planos e padrões | *Sprint Retrospective* | A cada Sprint, o *Sprint Retrospective* verifica a conformidade do processo |
| Satisfação dos critérios de transição nos processos de desenvolvimento | *Definition of Done e Sprint Review* | Aceitação do produto durante esta etapa pelo *Product Owner* através de critérios claros pré-definidos |
| Revisão de conformidade do software e reprodução do processo | *Rastreabilidade de Product Backlog e User Stories* | O registro de *User Stories* serve como uma "linha do tempo" do processo de desenvolvimento, auxiliando sua reprodução |
| Garantia do seguimento do processo de desenvolvimento previamente estabelecido | *Scrum Master e Sprint Restrospective* | O *ScrumMaster* tem o papel de manter a conformidade do processo de desenvolvimento estabelecido durante todo o processo. A conformidade do processo de desenvolvimento estabelecido é verificada durante o *Sprint Retrospective* |
| Garantia de que desvios de planos são detectados, analisados, mapeados e resolvidos | *Standup Meetings (Daily Scrum)* | Estas reuniões diárias servem para pequenos replanejamentos, gerenciando o surgimento de quaisquer desvios |

Tabela 7: Propostas de utilização de princípios ágeis no Processo de Gerenciamento de Configurações da DO-178C

| ***Processo de Gerenciamento de Configurações*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Revisão de mudanças | *Gestão e rastreabilidade de Product Backlog no Sprint Review* | *User Stories* servem para mapeamento e correção de problemas, e controle de revisão de mudanças |
| Estabelecimento e controle de *baselines* | *Releases* | Cada *Release* pode ser tratada como referência para modificações na próxima. |

,

Tabela 8: Propostas de utilização de princípios ágeis no Processo de Garantia de Qualidade (QA) da DO-178C

| ***Processo de Garantia de Qualidade (QA)*** | | |
|---|---|---|
| **Objetivo/Atividade** | **Proposta Ágil** | **Descrição** |
| Garantia de que o ambiente de desenvolvimento do software foi fornecido como especificado | *Sprint Zero* | O *Sprint Zero* serve também para preparação do ambiente de desenvolvimento, o qual continua evoluindo durante todo o ciclo de vida do software |
| Satisfação dos critérios de transição nos processos de desenvolvimento | *Definition of Done* | A *definition of done* pode constar métricas relacionadas aos critérios de transição |
| Garantia do seguimento de planos e padrões | *Sprint Retrospective* | A cada Sprint, o *Sprint Retrospective* verifica a conformidade do processo. |
| Satisfação dos critérios de transição nos processos de desenvolvimento | *Sprint Review* | Aceitação do produto durante esta etapa pelo *Product Owner* na transição entre *Sprints* |
| Revisão de conformidade do software e reprodução do processo | *Rastreabilidade do Product Backlog e User Stories* | O registro de *User Stories* deve acompanhar sua documentação de especificação definida no processo |
| Garantia do seguimento do processo de desenvolvimento previamente estabelecido | *Scrum Master e Sprint Restrospective* | O *ScrumMaster* tem o papel de manter a conformidade do processo de desenvolvimento estabelecido durante todo o processo. A conformidade do processo de desenvolvimento estabelecido é verificada durante o *Sprint Retrospective.* |
| Garantia de que desvios de planos são detectados, analisados, mapeados e resolvidos | *Standup Meetings (Daily Scrum)* | Estas reuniões diárias servem para pequenos replanejamentos, gerenciando o surgimento de quaisquer desvios |