

APLICANDO MÉTODOS ÁGEIS EM PROJETOS DE TESTE AUTOMATIZADOS

APPLYING AGILE METHODS IN AUTOMATED TEST PROJECTS

César Vinícius de Paula Ferreira¹, Ana Carolina Cintra Faria²

¹ Aluno do Curso de Sistemas de Informação

² Professora Mestre do Curso de Sistemas de Informação

RESUMO

Os métodos ágeis são conhecidos por acelerar a construção de um software deixando de lado o que é “menos importante” e focando exatamente na produção e na entrega de um produto com qualidade. Por serem flexíveis, torna-se mais fácil uma empresa adaptar parte do seu processo de trabalho tendo como foco o resultado, gastando-se menos tempo e dinheiro. As atualizações e melhorias de um software vão surgindo com necessidades constantes e entregas frequentes dos incrementos de software que proveem mudanças mais seguras com um bom conjunto de testes elaborados para as entregas. Pretende-se neste artigo, analisar melhor maneira de utilizar as técnicas ágeis voltadas para os testes de software, a partir da aplicação de métodos ágeis. A atividade de teste de software é essencial na busca pela qualidade dos sistemas em desenvolvimento. Assumem importantes papéis dentro do processo que adota metodologias ágeis, pois apoiam a comunicação entre as partes envolvidas no processo. Com teste de software, orientados por algumas boas práticas pregadas pelos processos ágeis, conseguimos resolver em menor tempo uma maior quantidade de tarefas e consequentemente efetuar entregas mais seguras de maneira constante. Os responsáveis pelas especificações devem garantir um conjunto mínimo de informações que permitam desenvolver e testar os requisitos. Desta forma, testadores e desenvolvedores se interagem mais e se ajudam a aperfeiçoarem os requisitos antes mesmo de entrarem na interação. O foco no cliente também é uma boa prática pregada pelos métodos ágeis.

Palavras-Chave: Métodos ágeis; Teste de software para processos ágeis; XP – Extreme Programming; Scrum; TDD – Test Driven Development;

ABSTRACT

Introduction: Agile methods are known for accelerating the construction of software, leaving aside what is “less important” and focusing exactly on the production and delivery of a quality product. Because they are flexible, it becomes easier for a company to adapt part of its work process with a focus on results, spending less time and money. Software updates and improvements come with constant needs and frequent deliveries of software increments that provide safer changes with a good set of tests designed for deliveries. The aim of this article is to analyze the best way to use agile techniques for software testing, based on the application of agile methods. The software testing activity is essential in the search for the quality of systems under development. They assume important roles within the process that adopts agile methodologies, as they support communication between the parties involved in the process. With software testing, guided by some good practices preached by agile processes, we managed to solve a greater number of tasks in less time and consequently make safer deliveries on a constant basis. Those responsible for the specifications must guarantee a minimum set of information that allows developing and testing the requirements. In this way, testers and developers interact more and help each other to improve the requirements even before entering the interaction. Customer focus is also a good practice preached by agile methods.

Keywords: Agile methods; Software testing for agile processes; XP – Extreme Programming; Scrum; TDD – Test Driven Development;

Contato: cesar.ferreira@unidesc.edu.br; anacarolina.cintrafaria@gmail.com

INTRODUÇÃO

Muito se ouve falar de metodologias ágeis e em seus benefícios para o

desenvolvimento de sistemas. Na prática, quem trabalha com metodologias ágeis se diz bastante satisfeito por ele proporcionar resultados rápidos e constantes.

Os benefícios para as empresas que adotam os métodos ágeis vão desde melhorias na agilidade do desenvolvimento do software, como redução do custo de desenvolvimento dos projetos. Há também uma melhor colaboração entre as partes envolvidas, pois prega que o cliente seja um integrante da equipe que irá desenvolver o software. A ideia central é fazer com que toda a equipe se engaje no projeto e nas sistemáticas do processo ágil, existindo uma colaboração mútua entre os envolvidos.

Todos os principais processos ágeis como a Extremem Programming (XP), SCRUM, Crystal Clear, Adaptive Software Development, FDD (Feature Driven Development) e DSDM (Dynamic System Development Method), trabalham com a necessidade de testar o software sempre que possível. Uma vez que processos ágeis priorizam o código executável, não seria interessante entregar um produto que apresente falhas, por exemplo, em uma rotina que o cliente priorizou e que seja de extrema importância para o negócio dele.

Um ponto essencial para garantir que exista fidelidade das implementações em relação aos requisitos, está nos testes executados sobre o código produzido.

Testes manuais, comumente conhecido como testes unitários, ou testes automatizados consistem em encontrar anomalias no funcionamento do software baseado nos requisitos identificados e detalhados, veremos como esses testes podem garantir um melhor funcionamento dele.

Sendo assim, o problema de pesquisa deste artigo consiste em estudar quais as maneiras de utilizar técnicas ágeis voltadas para testes de software?

E como objetivo geral pretende-se analisar melhor maneira de utilizar as técnicas ágeis voltadas para os testes de software, a partir da aplicação de métodos ágeis. E mais especificamente: i) revisar a literatura sobre métodos ágeis; ii) identificar publicações que descrevem resultados de aplicação de métodos ágeis em testes de software iii) descrever os testes automatizados empregando metodologias ágeis.

METODOLOGIA

Esta pesquisa é destinada para o desenvolvimento do Trabalho de Conclusão de curso e a metodologia escolhida é a pesquisa bibliográfica, pois esse tipo de pesquisa baseia-se no estudo de referenciais teóricos publicados, a fim de solucionar um

problema(teoria), levantando através deste estudo uma discussão (PIZZANI et al., 2012).

A presente pesquisa procura discursar primeiramente pela parte conceitual dos temas que envolvem o tema de testes automatizados utilizando metodologias ágeis. Os itens conceituais apresentados a seguir servem de entendimento e estão organizados em uma linha de raciocínio lógico que irá proporcionar ao leitor o entendimento não só dos conceitos, mas como todo raciocínio foi se desenvolvendo ao longo da pesquisa até a conclusão do trabalho proposto.

FUNDAMENTAÇÃO TEÓRICA

Os processos de desenvolvimento de software, atualmente, estão muito bem definidos na maioria das equipes de desenvolvimento. Diferente dos anos 80, 90 e início dos anos 2000 as empresas de tecnologia atualmente estão alinhadas à necessidade de utilizar metodologias ágeis que priorizam o levantamento dos requisitos funcionais para que na etapa de priorização das tarefas a serem desenvolvidas, os desenvolvedores estejam cientes do que devem desenvolver e garantam o máximo de entregas funcionais no final da Sprint de desenvolvimento.

Porém são poucas as equipes que possuem, inseridas no processo de desenvolvimento, entregas incrementais dos testes automatizados. Em sua grande maioria são os próprios desenvolvedores que testam a aplicação no momento em que estão desenvolvendo e após a entrega publica-se a aplicação em um ambiente de QA (Quality Assurance) ou Homologação e uma pessoa responsável pelos testes na aplicação irá testa-la baseado nos requisitos descritos nas tarefas desenvolvidas. Porém com a metodologia ágil, fica mais complicado de testar todo o cenário sem uma documentação consistente, como acontecia com os Casos de Uso (User Cases). Algumas empresas optaram por elaborar um modelo de Estória de Usuário (User Stories) reduzindo o tamanho do caso de uso e trocando o fluxo básico e alternativos por Cenários, para não perder a rastreabilidade das regras de negócio que impactam tanto em regras já desenvolvidas na própria Estória de Usuário desenvolvida como possivelmente em outras que também já foram desenvolvidas. O fato é que quanto mais a aplicação cresce, fica mais difícil controlar o impacto das mudanças, evolutivas e corretivas, na aplicação. Desta maneira nossos aplicativos estão sujeitos a erros e podem falhar de formas inesperadas depois de sofrerem qualquer alteração, por isto, os testes automatizados são necessários após qualquer alteração de código.

Métodos Ágeis e o manifesto ágil

Em fevereiro de 2001, 17 grandes pensadores da área de desenvolvimento de softwares, que trabalhavam em grandes empresas de diversos segmentos, se reuniram numa estação de esqui nos Estados Unidos com o objetivo de conversar e trocar experiências sobre a maneira como cada um desenvolvia seus projetos de software. (MEDEIROS NETO, Camilo Lopes de, 2012, p15).

Preocupados com o fato de o desenvolvimento de software estar desordenado e que as metodologias de desenvolvimento de sistemas não estavam conseguindo acompanhar a velocidade com que as demandas eram apresentadas foi apresentado um conjunto de valores e práticas extremamente voltadas para o gerenciamento de projetos de software altamente focados em objetivos e na produção do software.

Pensadores como Martin Fowler, Kent Schwaber, Kent Beck, Ward Cunningham, Jon Kern, entre outros, mesmo sendo concorrentes entre si, faziam parte desta comitiva onde o objetivo maior era poder oferecer ao mundo uma alternativa a estas metodologias pesadas que são fortemente utilizadas até hoje.

Nesta reunião foram apresentadas várias metodologias voltadas para o objetivo e um conjunto de valores ágeis detectados resumiu todas as diferentes metodologias apresentadas pelos participantes, e esse consenso gerou os valores do Manifesto Ágil, que são eles:

1. Indivíduos e interações são mais importantes que processos e ferramentas;
2. Software em funcionamento é mais importante que documentação abrangente/ extensa;
3. Colaboração/Relacionamento com o cliente é mais importante que negociação de contratos;
4. Responder a mudanças é mais importante que seguir um plano/planejamento;

Esse conjunto de valores já gerou várias polêmicas devido à má interpretação deles. O primeiro item é um exemplo que denota que às vezes é melhor manter as pessoas interagindo e discutindo ideias e criando soluções do que sentar à frente de uma ferramenta e não conseguir gerar um produto de qualidade simplesmente porque não tem conhecimento suficiente do negócio ou faltou alguma percepção para que o profissional evoluísse da melhor forma a informação que ele recebeu.

Uma chuva de ideias mal elaboradas e mal interpretadas, refletem diretamente no produto que acabará em tempestade de reclamações e críticas, manchando a imagem do

profissional e da empresa, sem contar a perda de dinheiro com manutenção de código e retrabalho.

Estas ideias vão de encontro com a realidade, o que é mais importante para o cliente, o software funcionando de forma perfeita e que satisfaça as suas necessidades ou uma documentação extensa? Certamente o sistema funcionando, porém isto também não quer dizer que no mundo ágil não teremos que documentar, é necessário à documentação sim, porém a prioridade e o foco é manter o software funcionando.

Estarei explicando um pouco das metodologias XP e Scrum, que são as mais utilizadas e difundidas no mundo, para uma melhor visão dos conceitos e benefícios dos métodos ágeis. Iremos perceber que ambas as metodologias incentivam testes a cada entrega, e que as entregas sejam curtas e funcionais.

Extreme Programming (XP)

Em 1990, Kent Beck começou a perceber em seu ambiente de trabalho, o que facilitava e o que dificultava a produção de um software. Porém sua percepção estava voltada para a satisfação do cliente e na forma como as equipes de desenvolvedores trabalhavam e não de como os programas eram escritos.

Kent então percebeu que o custo de desenvolvimento de um sistema estava muito mais ligado ao custo das pessoas do que ao hardware ou às licenças dos softwares utilizados, e que para um melhor andamento e crescimento de um projeto, era necessário envolver o usuário (cliente) no processo de produção e tornar os programas tão claros e simples, que possam ser lidos por pessoas não envolvidas no processo. Por isso Kent desenvolveu a metodologia XP.

A metodologia XP já foi utilizada por grandes empresas com a Chrysler no seu sistema de folha de pagamento global e na Ford Motors Company com o sistema VCAPS System.

O sistema da Chrysler que envolvia o controle de pagamento de seus 90mil empregados ao redor de todo o mundo. Esse projeto tem 2.000 classes e 30.000 métodos, e foi para produção sem nenhum atraso.

Já o sistema da Ford Motors Company, utilizada de metodologias tradicionais e enfrentava diversos problemas tradicionais como: Desenvolvedores ficar mais tempo consertando problemas encontrados do que desenvolvendo ou evoluindo o sistema. Após adotarem o XP dando foco em testes automatizados e integração contínua, em menos de um ano, algo que não conseguia evoluir a mais de quatro anos, foi desenvolvido, evoluído sem maiores problemas e alterações.

A Extreme Programming (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente [Beck (1999)]. As principais diferenças entre as outras metodologias e a XP são:

- Feedback constante;
- Abordagem incremental;
- A comunicação entre as pessoas é encorajada;

A maioria das regras da XP causam polêmicas no primeiro contato, pois podem não fazer muito sentido se aplicadas de maneira desordenada e de forma isolada.

Valores da XP

O conjunto das práticas adotadas pelo XP tem o foco no desenvolvimento rápido do projeto com o objetivo de garantir a satisfação do cliente, além de favorecer o cumprimento de prazos e estimativas fornecidas ao cliente. A XP possui quatro regras ou valores básicos que irão auxiliar no sucesso do projeto.

Comunicação - Permitir e manter o melhor relacionamento possível entre clientes e desenvolvedores, preferindo conversas, olho no olho, a outros meios de comunicação. A comunicação entre os desenvolvedores e o gerente do projeto deve ser constante. Procura-se o máximo possível comunicar-se pessoalmente, evitando-se o uso de telefone e o envio de mensagens por correio eletrônico. Conversas frente a frente tendem a ser mais proveitosas e as dúvidas podem ser esclarecidas naquele mesmo momento (2009, Don Wells).

Simplicidade - Permite a criação de código simples sem funções ou métodos desnecessários que possam poluir o código. Deve-se implementar o software com menor número possível de classes e métodos que atendam a demanda. E procurar implementar apenas os requisitos atuais e necessários postergando funcionalidades que podem ser importantes somente no futuro. A XP aposta que é melhor fazer algo simples hoje e pagar um pouco mais amanhã nas modificações necessárias do que implementar algo complexo e complicado hoje que talvez não venha a ser usado pelo cliente e foi solicitado por mero capricho. Estes requisitos que serão postergados devem ser anotados separadamente para não serem esquecidos, mesmo que tendem a ser abandonados ao longo do projeto, pois o próprio cliente irá enxergar que não serão mais necessários ao projeto (2009, Don Wells).

Feedback - O feedback deve ser constante o que implica em receber sempre informações do código e do cliente. A cada entrega pelo desenvolvedor, no caso de

rotinas, e pela equipe, no caso de módulos ou melhorias, todos os testes deverão ser repetidos e a tendência é que os erros diminuam e o cliente receba um produto eficiente e de qualidade. Em relação ao cliente, o feedback constante significa que ele irá receber constantemente uma parte do software totalmente funcional e deverá avaliar se a entrega foi bem-sucedida e reportar ao gerente da equipe os problemas encontrados e sugerir melhorias ao sistema. Eventuais erros e não conformidades encontradas por ele deverão ser rapidamente identificados e corrigidos nas próximas versões. A próxima versão deverá, se possível, ter corrigido todos os erros encontrados na versão anterior. Desta forma, a tendência é que o produto final esteja de acordo com as expectativas reais do cliente (2009, Don Wells).

Respeito – Todo membro da equipe contribui com valor mesmo que seja simplesmente entusiasmo. Desenvolvedores devem respeitar a experiência dos clientes e vice-versa. A Gestão deve respeitar o direito da equipe de aceitar a responsabilidade e receber autoridade sobre próprio trabalho (2009, Don Wells).

Coragem – A equipe deverá ter coragem para implantar os três valores anteriores. A coragem também dá suporte à simplicidade, pois assim que a oportunidade de simplificar o software é percebida, há um ganho na comunicação da equipe e uma maior interação entre os membros. Além disto, é preciso coragem para obter o feedback constante do cliente. Não são todos que possuem uma facilidade na comunicação e tem bom relacionamento, mas isto é estimulado e a intenção é que todos os membros, com o tempo, estejam nivelados nestas quatro regrinhas básicas (2009, Don Wells).

As doze práticas do XP

Kent Beck então começou a adotar métodos diferentes e percebeu melhorias de resultado através de experiências adquiridas nos projetos. Resolveu então mostrar na reunião para os outros representantes as doze práticas que ele já seguia e então sugeriu como base a ser seguida nas metodologias ágeis, como ele segue na Metodologia XP.

1. **Planejamento** – Consiste em decidir o necessário a ser feito e o que deve ser adiado no projeto. A XP baseia-se em requisitos atuais para desenvolvimento de software, não em requisitos futuros. A ideia é evitar os problemas de relacionamento entre a área de negócios (clientes) e a área de desenvolvimento e estimular a cooperação entre estas áreas de forma que cada uma deve focar em partes específicas do projeto e garantir o seu sucesso. A área de negócios deve decidir sobre o escopo, a composição das versões e as datas de entrega, já os desenvolvedores devem decidir sobre as estimativas de prazo, o processo de desenvolvimento e o crono-

grama detalhado para que o software seja entregue nas datas especificadas pela equipe.

2. Entregas frequentes – Tem como meta a construção de um software simples. De acordo com o surgimento dos requisitos, as atualizações serão feitas. Cada versão a ser entregue deverá ter o menor tamanho possível contendo os requisitos de maior importância para o negócio. O prazo das entregas poderá ser discutido entre o cliente e a equipe, porém cada versão entregue deverá ser feita em no máximo a cada dois meses, aumentando a possibilidade de feedback rápido do cliente. Isto irá diminuir os retrabalhos e irá evitar que apareçam surpresas quando o software for entregue e garantirá que a versão final estará de acordo com o que o cliente realmente quer.

3. Metáfora – É a descrição de um software sem a utilização de termos técnicos. A ideia é guiar o desenvolvedor de software a entender o negócio. Deverá ser descrito como se fosse descrever para um leigo no assunto.

4. Projeto simples – Os projetos devem ser desenvolvidos de forma simples e satisfazer os requisitos atuais sem se preocupar com requisitos futuros. Requisitos futuros devem ser levados em consideração assim que surgirem de fato. No desenvolvimento deve-se elaborar o projeto com o menor número de classes e métodos possíveis sem comprometer o funcionamento do software de forma a se ter um código simples e legível.

5. Testes – Os programadores escrevem continuamente os testes de unidade, que devem funcionar como guias para o desenvolvimento. O foco é validar as rotinas do projeto durante todo o processo de desenvolvimento. Os testes são criados antes mesmo do software programadores desenvolvem o software criando primeiramente os testes (TDD - Test Driven Development).

6. Programação em pares – A implementação do código é feita em dupla, ou seja, todo e qualquer desenvolvimento é escrito com dois programadores em um único computador. Um programador desenvolve, enquanto o outro observa continuamente o trabalho que está sendo feito. Estes papéis devem ser alterados continuamente o que favorece aos desenvolvedores estarem continuamente aprendendo um com o outro e automaticamente nivelando o conhecimento geral dentro da equipe.

7. Refatoração – Foca na reestruturação do sistema sem mudar seu comportamento. Tem ênfase na remoção de códigos duplicados, melhoria de rotinas, simplificação de código e deve ser feita apenas quando é necessário, ou seja, quando um desenvolvedor da dupla, ou os dois, perceber que é possível melhorar o código atual sem perder nenhuma funcionalidade.

8. Propriedade coletiva – O código do projeto pertence a todos os membros da equipe. Qualquer um pode mudar todo o código em qualquer lugar no sistema no momento que achar melhor. Todos são responsáveis pelo software inteiro e devem zelar pela sua organização e funcionamento. Caso um membro da equipe deixe o projeto antes de seu fim, a equipe deverá conseguir continuar o projeto com menos dificuldades, pois todos conhecem todas as partes do software.
9. Integração contínua – É a prática de interagir e construir o sistema de software várias vezes por dia, mantendo o sistema integrado continuamente evitando preocupações futuras. Os programadores deverão integrar um conjunto de modificações de cada vez e definir quem deverá fazer as correções quando os testes falharem. Deverá existir uma máquina de integração, que deve ter livre acesso a todos os membros da equipe.
10. Semana de quarenta horas de trabalho – As pessoas envolvidas no projeto deverão trabalhar em geral 40 horas e nunca fora do tempo estipulado. A XP assume que não se devem fazer horas extras constantemente. Caso seja necessário trabalhar mais de 40 horas pela segunda semana consecutiva, existe um problema sério no projeto que deve ser resolvido com um melhor planejamento e não com um aumento de horas trabalhadas. Esta prática procura melhorar os processos e planejamentos acabando com a teoria de que se o projeto está atrasado, vamos trabalhar mais e contratar mais pessoas. Os planos devem ser alterados e não sobrecarregar os envolvidos.
11. Cliente presente – É fundamental a participação do cliente durante todo o desenvolvimento do projeto. É aconselhável incluir ao menos um cliente na equipe de forma que este cliente fique em tempo integral disponível para responder perguntas e esclarecer dúvidas de requisitos, evitando atrasos e até mesmo construções erradas.
12. Código padrão – Padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores. Eles deverão escrever todo o código de acordo com as regras adotadas pelo grupo inteiro.

O XP adota uma abordagem orientada a objetos que inclui um conjunto de regras e práticas que ocorrem de forma contínua e repetitiva.

As quatro fases do XP

Em resumo Kent desenvolveu a metodologia XP com base em fases que juntas contemplam o conjunto de doze regras descritas acima e formam um ciclo contínuo e repetitivo para cada requisito do projeto que corresponde às quatro fases do XP.

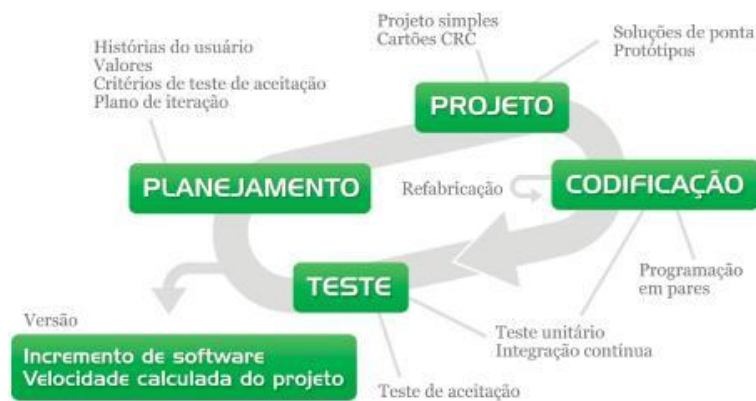


FIGURA 1: Fases do Processo XP Fonte: (PRESSMAN, 2006, p.64)

- **Planejamento** - As atividades de planejamento começam com a criação de um conjunto de histórias. Os membros da equipe avaliam cada história e lhe atribuem um custo. Se a história precisar mais do que três semanas de desenvolvimento, pede-se ao cliente para dividir a história em histórias menores e novamente é atribuído um novo custo às histórias menores. À medida que o trabalho de desenvolvimento for evoluindo, o cliente pode adicionar novas histórias ou até mesmo eliminar histórias que ele julgue desnecessário.
- **Projeto** – Mantenha a simplicidade. Aqui entra a regrinha do “menos é mais”. O projeto simples tende a ser preferível em relação a uma apresentação tumultuada e complexa. O projeto deverá fornecer diretrizes de implementação para uma história. As histórias podem ser divididas e separadas em cartões. Escrever nestes cartões leva os envolvidos a raciocinar e apresentar o seu entendimento da situação, discutir ideias e escolher o melhor caminho a ser seguido. Nestes cartões podem ser identificadas as classes, orientadas a objeto, que são relevantes para o desenvolvimento do sistema. Se um problema é complexo, deve ser criado de forma imediata um protótipo operacional, chamado de “solução de ponta”. Neste caso a intenção é diminuir o risco antes mesmo de começar a codificar a história. Assim conseguimos validar melhor as estimativas e corresponder às expectativas do cliente.
- **Codificação** – Depois que as histórias foram desenvolvidas e todo trabalho preliminar do projeto estar feito a equipe deverá desenvolver uma série de testes unitários. Os testes devem ser definidos e codificados antes mesmo que o módulo, que irá ser testado esteja escrito. Os testes unitários deverão representar cada história descrita. A atitude de pensar nos testes que serão feitos já refina a codificação e contribui para a eliminação prematura de possíveis erros.
- **Teste** – Os testes unitários devem ser implementados de forma que possam ser automatizados e possam ser executados com facilidade de forma repetida. Conforme os testes vão sendo organizados conseguimos

ter uma ideia de como o sistema deverá ser implementado. Com o crescimento dos testes unitários e a sequência em que estes testes são organizados temos uma indicação contínua de como o sistema está organizado e conseguimos visualizar a evolução do sistema e também conseguimos perceber com antecedência quando as coisas estiverem indo para um caminho não satisfatório. Estes testes de aceitação são chamados de “testes de cliente”, são especificados pelo cliente e focalizam as características funcionais do sistema que são passíveis de revisão e modificação por ele.

Scrum

O nome Scrum foi escolhido pela similaridade entre o jogo de Rugby, é a forma de reiniciar o jogo após uma falta accidental ou outro incidente em que não é possível determinar quem cometeu a falta. A proposta é possuir um método de desenvolvimento de sistemas de forma adaptável, rápida e promover uma auto-organização na equipe e que ela seja capaz de se recuperar e criar soluções com as dificuldades e problemas enfrentados.

Scrum é um framework estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990. Scrum não é um processo ou uma técnica para construir produtos; em vez disso, é um framework dentro do qual você pode empregar vários processos ou técnicas.

A Scrum deixa clara a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las. No entanto que ela complementa as práticas do Extreme Programming (XP) o que acaba servindo como uma evolução para aqueles que já utilizam desta prática, que pode ser considerada com o ponto de partida.

O que mais chama atenção no Scrum é que ele é simples de ser entendido e se bem aplicado, em sua prática pode levar ao controle do processo de desenvolvimento de software.

O Scrum visa ser um conjunto simples e eficaz de regras e ferramentas para maximizar resultados, gerando o mínimo possível de sobrecarga administrativa. Abaixo apresentamos de forma resumida uma visão geral do Scrum.

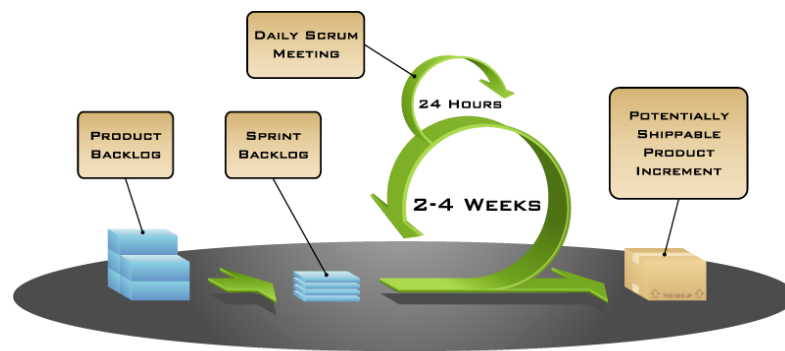


FIGURA 2: Visão geral do Scrum. Fonte: (Wikimedia – adaptada pelo autor).

Tudo começa com um “Product Backlog” que nada mais é que uma lista com os desejos do seu cliente para o sistema que será desenvolvido. Esta lista deverá ser priorizada e dividida em “Sprint Backlog” onde irá contemplar as tarefas que serão realizadas dentro de um período de uma a quatro semanas. Ao final desse período é feita uma entrega funcional que poderá ser testada e aprovada. Iremos detalhar mais estes processos nos passos que seguem abaixo.

Os papéis existentes no Scrum

No Scrum, existem três papéis.

- **Product Owner** – É o dono do produto, é o papel de maior responsabilidade e visibilidade no método Scrum, representa o cliente e é responsável pelas decisões de qual trabalho da equipe maximizará o valor do produto e agregará mais aos envolvidos, define metas, prioriza e aprova a qualidade, sempre buscando garantir o maior retorno ao investimento.
- **Scrummaster** – É o membro da equipe que garante o bom andamento do projeto, assegurando que seus ritos sejam cumpridos, que seus artefatos sejam usados de maneira correta e qualquer obstáculo ao trabalho da equipe seja resolvido. É o cara que remove todos os obstáculos, custe o que custar.
- **Equipe** – É quem realiza as ações de construção do projeto. Não há hierarquia entre os membros da equipe, e ela se auto gerencia na distribuição e no agendamento das tarefas constantes no Sprint backlog.

Artefatos e ritos do Scrum

Os ritos do Scrum são, basicamente, três reuniões.

1. A reunião de planejamento do Sprint, onde são priorizadas e divididas as tarefas.

2. A reunião de revisão do Sprint, em que se faz um apanhando do que deu certo, do que deu errado e do que é possível melhorar.
3. E as reuniões diárias, nas quais se faz um relatório do que foi feito no dia anterior, de quais as tarefas do dia de hoje e quais os obstáculos que podem impedir o bom andamento das tarefas.

Os artefatos ou objetos sociais são o “Product backlog”, “Sprint backlog” e o “Burndown Chart”, que é uma planilha e um gráfico que mostram o consumo dos recursos do projeto (tempo, pontos de função ou qualquer outra medida adotada) conforme ele avança, servindo como uma rápida ferramenta visual para a correção imediata de rumos.

Na prática os artefatos e os ritos podem sofrer algumas variações, mas os papéis no Scrum são praticamente imutáveis.

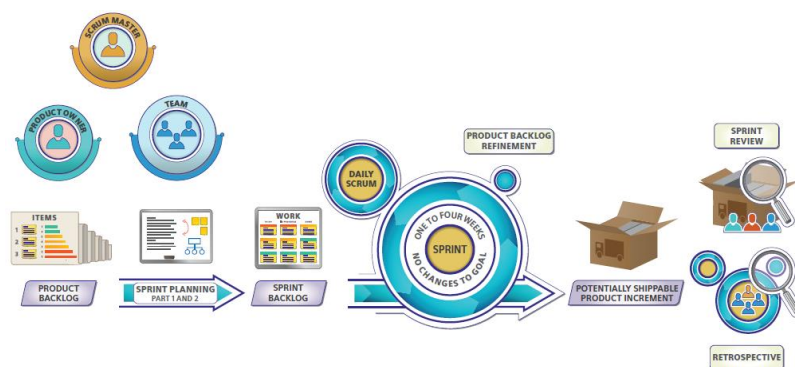


FIGURA 3: Visão do Scrum com papéis e artefatos. Fonte: (Wikipédia – adaptada pelo autor).

- **Product Backlog** – É a lista de tarefas, onde é listado tudo o que é necessário ao projeto, independente do número de “sprints” que compõe. Ele deve contemplar o que no XP chamamos de “User Stories”. Nesta lista deverá contemplar o que o cliente deseja junto às tarefas necessárias para atender a esses desejos de forma bastante sintética. Estas tarefas são priorizadas de acordo com o que o Product Owner (quem está pagando) deseja. Isto deve ser feito da forma mais simples possível, permitindo a busca textual por tarefas em um documento que possa ser acessado e alterado por todos os membros do projeto. Com o “Product backlog” em mãos, agora ele pode ser subdividido em “Sprints backlogs”.
- **Sprints Backlogs** – É a lista de tarefas que serão executadas dentro de cada Sprint por uma determinada equipe. Nada impede que você subdivida as tarefas entre equipes distintas e tenha sprints diversas ocorrendo simultaneamente.
- **Burndown Chart** – Nada mais é que um gráfico que informa uma unidade de medida estipulada nas “sprints backlogs”, de forma que permita que o time avalie se o projeto está caminhando de maneira adequada. Atua como um elemento de realimentação, pois a cada Sprint finalizada deverá

ser alimentado com a unidade de medida estipulada para a execução e o controle de qualidade.

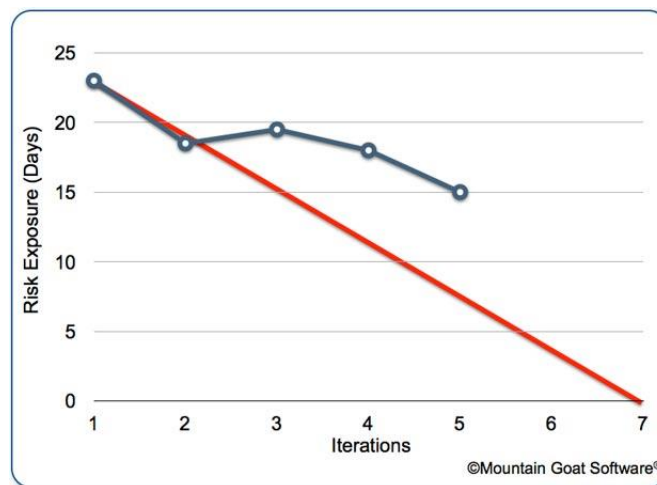


FIGURA 4: Exemplo do gráfico “Burndown Chart”. Fonte: (mountain Goat Software – Mike Cohn – www.mountaingoatsoftware.com).

A linha vermelha é o caminho ideal para que você cumpra seu objetivo de acordo com o que foi mensurado nas “Sprint backlogs”. Em azul está a realidade do projeto. Histórias concluídas fazem com que a linha desça e informe como vai o projeto. Histórias que levam mais tempo do que esperado esticam a linha para a direita de forma que você consiga visualizar o tempo em que ela foi executada.

Quando a linha de risco não está descendo a uma taxa adequada, a equipe pode querer modificar e adequar o “orçamento” ajustando o cronograma no próximo Sprint e ajudar a mitigar os riscos. Um detalhe significativo é: aumentar ou diminuir o tempo de uma Sprint não quer dizer alterar o tempo final do projeto. A não ser que o “Product Owner” tenha consciência desta modificação e aceite o que foi proposto.

Construindo o Product Backlog

Já conseguimos perceber que o ponto de partida no Scrum é “Product Backlog”, que é a lista de tudo que o produto ou projeto deve contemplar. Porém ainda não definimos como iremos fazer para adquirir esta lista com as informações que realmente interessam. Para isso deveremos entender melhor o que são as “User Stories”.

“User Stories” permitem àqueles que conhecem a técnica da construção de uma solução guiar quem necessita dessa solução no exercício de descrevê-la de forma simples e concisa. Dever ter o foco na história e não mencionar termos técnicos nela.

Uma boa história deve ser explicada em média de 30 segundos e cada história deve “caber” no trabalho a ser executado em uma semana pela equipe de

desenvolvimento e deve ser facilmente testável.

O cliente deve poder acompanhar o desenvolvimento do sistema lendo a “User Story” que ajudou a escrever.

Quando auxiliamos o cliente na construção de “User Stories”, devemos ter em mente que elas serão atendidas, na construção da solução, com a realização de uma série de tarefas. Cada tarefa deve ser específica, contida em si, bem definida e também independente. A tarefa pode ser medida para que seu custo possa ser efetivamente avaliado. Estas tarefas deverão ser definidas de forma que possam ser realizadas e devem ser relevantes para a história, caso uma tarefa pareça não estar relacionada com a história, então a tarefa ou a história deverá ser revisada. No final um período de tempo deve ser definido para a tarefa.

A escrita das “User Stories” ajuda na descoberta das principais dores a serem atendidas, facilitando a priorização posterior das tarefas.

Para priorização de “User Stories”, ou simplesmente tarefas geralmente utilizam-se o Poker do Planejamento ou “Poker Planning”. Consiste em uma prática de estimativa de tarefas simples e eficiente.

Ao invés de estimar horas exatas, estima-se em pontos. Os pontos utilizados no “jogo” são parecidos com a sequência de “Fibonacci”, ou seja, o próximo número é a soma dos dois primeiros números anteriores: 0, 1, 2, 3, 5, 8, 13, e assim por diante. A sequência mais utilizada é: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 e 100, porém vai de cada empresa adotar a sua melhor sequência. Os números representam o tamanho das tarefas e são apresentados pelos desenvolvedores no momento em que vão mensurar as tarefas a serem desenvolvidas.

A partir do “Product backlog”, já se determina as primeiras tarefas a serem executadas e então se estima o esforço de desenvolvimento em casa uma delas. Todos, “com exceção do “Scrummaster” recebem as cartas do Poker do Planejamento e mais os ícones de “?” e “infinito”.

Todos os desenvolvedores devem apresentar suas cartas, ao mesmo tempo, informando quantos pontos ele acha que irá utilizar para desenvolver aquela tarefa. O ícone “?”, significa que o desenvolvedor não tem ideia de quanto tempo ele irá levar para desenvolver a atividade, e o ícone “infinito”, significa que a tarefa irá tender a se alastrar e não finalizar rapidamente.

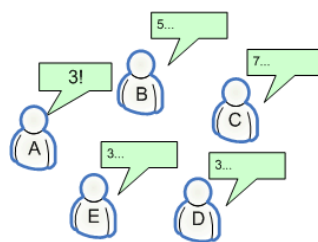


FIGURA 5: Exemplo de discussão de estimativa. Fonte: (<http://luizricardo.org/2013/09/entendendo-o-triangulo-de-ferro-porque-nao-podemos-ter-tudo/>)

E nessa dinâmica de cada um expor sua opinião e justificar os pontos estimados. Haverá uma troca de conhecimento e um amadurecimento entre os integrantes da equipe, provavelmente surgirão ideias diferentes sobre a mesma tarefa o que irá influenciar pessoas a opinarem sobre a melhor solução das tarefas.

Tarefas muito grande, tendem a ser divididas como foi falado anteriormente, se a equipe definir que tarefas maiores que 8 ou 13 pontos são tarefas que tendem ao infinito, ela obrigatoriamente deverá ser quebrada em outras tarefas até que consigam mensurar de forma correta.

A ideia é discutir a variação de estimativa entre os desenvolvedores e no final a equipe deve chegar a um consenso do peso da tarefa, partindo para a estimativa das demais tarefas.

Entendendo as Sprints

É durante os sprints que a maior movimentação do Scrum acontece. As “Sprint Backlogs” são definidas na primeira reunião, com a presença do Product Owner, e detalhadas na segunda reunião, com o Scrum master e a equipe do projeto. Os itens agrupados por assuntos e definidos com prioridade naquele instante podem ser inseridos em uma “Sprint”. O prazo da Sprint deve ser definido pela equipe e não deve ser maior que duas semanas. Dentro de cada Sprint estarão às histórias já estimadas com as horas de trabalho, discutidas e definidas pela equipe através do Poker do Planejamento.

Estas histórias devem estar literalmente em um quadro de avisos ou tarefas que chamamos de Kanban. Na prática você divide um quadro em três colunas básicas: “Fazer”, “Em Execução” e “Feito”. Nestas colunas você deverá colar suas etiquetas adesivas com as tarefas, e nas próprias etiquetas adesivas, as anotações relativas às tarefas. Essas etiquetas vão avançando no quadro à medida que as tarefas progredirem.

Durante o Sprint, a equipe deve estar totalmente protegida de qualquer interferência externa e o Scrum master deve garantir que todos os seus componentes tenham recursos necessários ao desenvolvimento de seus trabalhos, mas jamais permitir

o surgimento de tarefas adicionais. A equipe, caso queira, pode fazer, ela mesma o refinamento das tarefas com as quais está trabalhando, mas com a atenção do Scrum master, que deve evitar que “lampejos de criatividade” tragam para o Sprint coisas que estão fora de seu escopo, comprometendo entregas prometidas ou seu tempo de execução.

Na reunião diária são feitas três perguntas aos participantes das sprints:

1 – O que você fez ontem?

2 – O que você fará hoje?

3 – Quais obstáculos que impedem (ou podem impedir) seu trabalho?

E durante a reunião, à medida que os participantes forem informando e atualizando o scrummaster, as etiquetas adesivas, no quadro KanBan, são atualizadas e movidas apropriadamente entre suas colunas. Como todos acompanham diariamente o trabalho em conjunto, podem surgir questões que os próprios membros da equipe possam resolver entre si.

O Burndown Chart, que vimos anteriormente serve também de alerta constante do consumo do tempo disponível até o final do Sprint.

Testes no Scrum e no XP

No Scrum, como o XP, prezam pelas “Entregas Frequentes” e os testes de software a cada Entrega. No final de cada Sprint é feita uma reunião para apresentar ao Product Owner o que ele solicitou na reunião de início do Sprint.

Nesta reunião além da apresentação deverá ser entregue um produto funcional para que o Product Owner possa testar e qualificar o que lhe foi apresentado.

Antes da apresentação, ou entrega de cada Sprint, o produto deverá ser testado de forma que possa evoluir até a sua entrega final, tendo todas as rotinas e etapas testadas.

Novas solicitações ou modificações poderão acontecer e caso impacte em outras histórias, os sprints deverão ser ajustados de acordo com a nova demanda.

O Scrum prega por duas boas práticas para teste de software. Uma é o “Test-First”, que foi herdada do XP, onde a cada nova história gerada o desenvolvedor deverá escrever um código de testes, cujo escopo atenda unicamente a uma funcionalidade específica.

E a outra regra é que o Scrum prega que o “Time” deve ser multidisciplinar. O grande desafio proposto, na verdade é formar um Time multidisciplinar, com vários profissionais especialistas em cada necessidade do projeto, e que todos se ajudem mutuamente para completar os trabalhos. Aqui é que está o grande segredo, e também o

que normalmente gera confusão. O Scrum provoca o Time a desenvolver esta multidisciplinaridade, expandindo e espalhando os conhecimentos a respeito das disciplinas envolvidas e disseminando entre o Time as capacitações adquiridas.

No entanto, não se deve confundir com “Todos devem saber tudo e fazer tudo”. O que a multidisciplinaridade tenta provocar no Scrum é que os indivíduos aprendam um com os outros e com isso sejam capazes de ajudar o Time e realizarem tarefas que antes não conseguiam realizar, mas não como especialistas, apenas como executores que contribuem para completar as tarefas. A multidisciplinaridade é no Time e não nos indivíduos.

Esta multidisciplinaridade afeta diretamente nos testes do projeto, pois com o tempo os integrantes do grupo irão adquirindo novos conhecimentos, em outras áreas do projeto, e acabam por corrigir eventuais erros ou problemas que surjam de forma espontânea, e a equipe ganha em conhecimento, agilidade e testes dentro do projeto.

RESULTADOS

Testes de software para processos ágeis

Processos ágeis priorizam o código executável ao invés de uma extensa documentação escrita. O que garante a fidelidade das implementações em relação aos requisitos propostos está justamente nos testes executados sobre o código produzido.

No Scrum podemos perceber uma forte presença de liderança no projeto e no gerenciamento dos requisitos. Os requisitos são fortemente testados após cada Sprint e garante que todo o software seja testado à medida que vai evoluindo.

No XP vimos que existe um conjunto de práticas para desenvolvimento de software onde se prega também a refatoração do código, programação em pares para que um desenvolvedor possa auxiliar o outro e diminua incidências no código e equilibre o conhecimento da equipe e também estimula o desenvolvimento dirigido por testes.

Requisitos para metodologias de Teste

A diferença entre as metodologias de testes guiados por planos e as metodologias ágeis é basicamente o formato e o momento em que os testes são executados e ainda os documentos que são gerados em cada etapa de testes.

Nos processos guiados por planos, existe toda uma quantidade extensa de artefatos o que resulta em uma análise mais criteriosa sobre o sistema a ser desenvolvido e os testes nesse caso geram apenas mais um artefato produzido pelo processo antes da liberação da versão final.

Como nos processos ágeis, não se importa muito com a documentação e o foco é no objetivo e no resultado, e não em gerar artefatos, os testes são de suma importância. Como as entregas são mais curtas, a cada entrega é gerada uma bateria de testes o que garante o sucesso da metodologia. As correções são feitas assim que o problema é detectado, ou seja, de forma imediata, e não no momento da entrega do produto final.

Testes Manuais e Testes Automatizados

O XP adota a técnica do “Test-First”, onde a cada nova história gerada o desenvolvedor deverá escrever um código de testes, cujo escopo atenda unicamente a uma funcionalidade específica.

Um novo teste deve ser adicionado a uma bateria de testes já existente e todo o processo de teste deverá ser executado novamente de forma que não apresente erros. Os processos são adicionados de forma acumulativa até que chegue ao final do projeto, onde teremos um teste para todo o software, que deverá ser executado por completo sem erros.

Outra possibilidade do processo seria escrever a rotina de teste depois que ela já tenha sido implementada pelo desenvolvedor, esta técnica recebe o nome de “Test-Last”, o que não altera o restante do processo, pois o teste também deverá ser adicionado à bateria de testes e não poderão ocorrer erros em sua execução.

Não existem regras que obrigam a utilizar um tipo específico de teste em um software. A empresa pode utilizar de técnicas que julgam melhor e que acreditam que se chegue a um resultado satisfatório para ela e para o seu cliente. Os testes mais comuns são os testes manuais e os testes automatizados, que podem ser executados em todas estas baterias de testes.

Testes manuais são os testes em que o desenvolvedor ou até mesmo o próprio usuário, utiliza o sistema em busca de erros tanto lógicos, ou seja, de desenvolvimento e programação quanto erros em que rotinas foram mal elaboradas. É o tipo de teste menos eficiente, porém o mais executado. Dificilmente uma pessoa conseguiria testar um sistema de forma que não existissem mais erros. O usuário enquanto opera o sistema também está testando, de forma indireta, mesmo que não esteja à procura de algum erro. Erros encontrados são repassados para a equipe responsável para devidas correções.

Nem todas as funções de um sistema podem ter seus testes automatizados devido a diversos fatores como o custo excessivo de automação ou até mesmo devido à incompatibilidade das ferramentas de automação utilizadas com determinados componentes do sistema.

Os testes automatizados consistem na utilização de programas específicos e estáveis capazes de testar exaustivamente um software através de scripts de testes pré-configurados. Será necessário um profissional com conhecimentos na ferramenta escolhida que saiba manuseá-la e configurá-la para que obtenha a análise e o resultado esperado.

Como nos processos ágeis cada iteração é entregue uma versão funcional do software, as constantes modificações no sistema podem gerar efeitos colaterais em uma interface já testada. Se essa interface teve seus testes automatizados, os efeitos colaterais tendem a ser descobertos com mais facilidade. Assim, toda vez que um teste falha, o testador pode verificar o motivo da falha e reportar o defeito ou, caso seja uma alteração no requisito, corrigir o teste.

É aconselhável automatizar pelo menos os testes das interfaces mais estáveis, pois automações realizadas em interfaces estáveis não sofrem muitos problemas de manutenção.

Testes de Unidade

Muitos dos praticantes envolvidos com metodologias ágeis pregam a utilização dos testes unitários para que os softwares sejam desenvolvidos com mais segurança e qualidade.

“Unit Test”, comumente chamados é um teste único, sobre uma rotina, ou um requisito específico, que visa manter a legibilidade, a qualidade e minimizar problemas no processo de desenvolvimento.

Um teste unitário examina o comportamento de uma unidade distinta de trabalho, deve ser capaz de examinar o comportamento do código sob as mais variadas condições, devem testar unidades de trabalho de forma isolada, pois ele testa aquele, e somente aquele método.

Os “Unit Tests” agregam valores para a aplicação e naquilo que se pretende testar, devem ser específicos, objetivos e focados. Precisam ser legíveis, para que qualquer outro desenvolvedor possa olhar para sua classe e ir lendo cada um deles, sem dificuldades em saber o que eles estão testando. O código deve ser tão legível quanto seu código funcional. Um “Unit Teste” não depende de outro, são únicos. Precisam rodar de forma rápida, pois lentos impactam na quantidade de vezes que você irá executá-lo. Seus nomes devem ser claros e específicos, evitando descrever toda descrição no Unit Test no nome do método. Não há problemas em ter vários Unit Test testando resultados diferentes para o mesmo método.

TDD (Test Driven Development)

Acreditando que esta técnica que chamamos anteriormente de “Test-First”, onde escrevemos os testes primeiro do que o desenvolvimento e que nos auxiliam a resolver alguns problemas de forma antecipada, seja a melhor forma de testar um software e ao mesmo tempo elaborar um plano de teste de forma incremental e que acompanhe o desenvolvimento do software, foi criada a técnica chamada TDD, o Test Driven Development – Desenvolvimento dirigido por testes.

O TDD é a junção do TFD (Test First Development – Confiança no funcionamento) com a atividade de Refactoring (Limpeza – Alta qualidade do código desenvolvido).

Escrever os testes primeiro é a melhor forma de atingir alguns objetivos como: Validar o problema, verificar cenários positivos e negativos e o Design também pode ser guiado pelos testes.

Ou seja, você consegue testar seu software antes de tê-lo pronto e não apenas criar os testes depois de todo o software já pronto. Com o TDD não validamos somente se há um erro de lógica no código escrito, como também se os requisitos estão bem definidos, pois os testes são criados em cima dos requisitos especificados, consequentemente não haverá surpresas na entrega do produto.

Testar software, convenhamos que para a maioria dos desenvolvedores não seja uma atividade “divertida”, mas todo mundo sabe que é necessário e de extrema importância.

O TDD prega pela técnica que você deve primeiro escrever testes e depois escrever código. Os testes devem guiar o desenvolvimento do código. A intenção é que os testes proveem uma especificação de “o que” um pedaço de código faz e este pedaço de código deverá servir também como uma documentação do sistema.

Em suma uma breve rotina para o ciclo deve ser, escrever poucos casos de testes, implementar o código, escrever outros poucos casos de testes e implementar o código novamente e assim por diante, de forma que de pouco em pouco você terá todo o ambiente de teste codificado e todo o código implementado sem erros, pelo menos é o que se espera.

O ciclo do TDD é pequeno e simples:

- 1 – Escreva um teste que falha. (Red)
- 2 – Faça o teste passar rapidamente. (Green)
- 3 – Refatore.

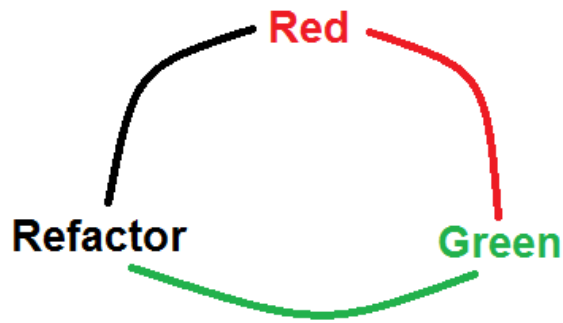


FIGURA 6: Ciclo básico do TDD.

A filosofia do TDD se alinha com os métodos ágeis, principalmente com a XP e também ao Scrum e possui o mesmo pai que a XP, Kent Beck.

Também é possível utilizar o TDD de forma independente e acrescentá-lo aos processos que não utilizam de metodologias ágeis. Basta adotá-lo de forma sistemática no processo de desenvolvimento de software. É muito mais uma questão de disciplina que de mudança filosófica. Algumas ferramentas como xUnit, jUnit e o próprio Visual Studio da Microsoft já possuem suporte a esta metodologia.

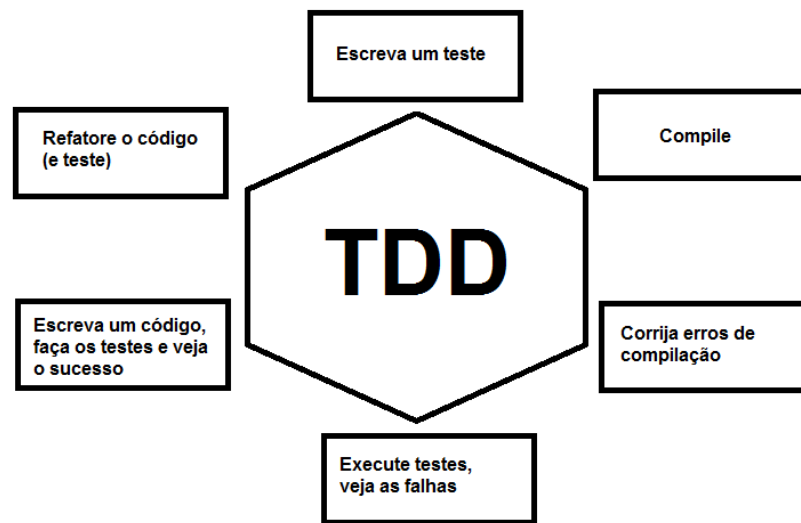


FIGURA 7: Ciclo detalhado do TDD.

Para construir um projeto a fim de praticarmos TDD devemos escolher um problema para solucionar, no caso um requisito levantado. Definida as “User Stories” e os critérios de aceitação, fica claro que se a implementação do critério de aceitação estiver concluída na forma de testes automatizados, e esta estiver com a indicação de erro, então a user story não está pronta.

Geralmente teremos, ao invés de termos um projeto com uma documentação baseada em casos de uso, teremos um projeto com uma documentação baseada e organizada em user stories, que é nada mais que os métodos que mais acompanha

equipes que trabalham com métodos ágeis.

Isto favorece na ideia central e principal dos métodos ágeis em que o TDD é aplicado. A intenção é que tenhamos, código sem duplicação, evitar código inútil com classes muito grandes e sub-classes muito semelhantes, ou seja um código de alta qualidade e sem falhas.

Testabilidade

Testabilidade é uma forma de medir quão bem e quão eficiente uma aplicação pode ser testada. Aplicações desenvolvidas pensando em testabilidade economizam tempo, dinheiro e diminuem o risco. Como nas metodologias ágeis, as iterações são curtas e o tempo para testar o software também, desenvolver pensando em testabilidade passou a ser crucial. Normalmente aplicações que possuem rotinas e métodos mais fáceis de testar serão em um mesmo intervalo de tempo, mais testadas do que aplicações que são difíceis de testar. Neste caso é importante se preocupar com as entregas sempre funcionais. Se o código não está rodando em sua plenitude, ele necessita de testes e correções, independentemente do tempo que se leve para a próxima iteração. Geralmente áreas que são mais difíceis de testar são as que apresentam maior risco ao negócio, então não se pode começar uma nova iteração sem que a anterior esteja funcionando em sua plenitude.

Uma organização madura em processos ágeis deve possuir equipes misturadas de testadores e desenvolvedores. Essa interação mais próxima entre as equipes ajudará na divisão de responsabilidades para a automação dos testes, na adoção do desenvolvimento dirigido a testes e na execução regular de grandes suítes de teste. Nos processos ágeis isto é uma questão de necessidade evidente onde, cada vez mais se torna vital que o software seja desenvolvido pensando-se na facilidade de testá-lo.

Testes Automatizados e Frameworks

Para garantirmos a Testabilidade da aplicação e acelerarmos a automatização dos testes de softwares, podemos utilizar de programas e frameworks para auxiliar e acelerar a bateria de testes. A principal vantagem é que além de conseguirmos executar os testes de maneira rápida conseguimos também executar testes em partes específicas do software.

Como ferramenta que auxilia na automatização dos testes podemos utilizar o Selenium e para aferir a qualidade do código, podemos utilizar o xUnit (Microsoft .Net) ou JUnit(Java) para inserir e manipular valores nos elementos identificados pelo Selenium e

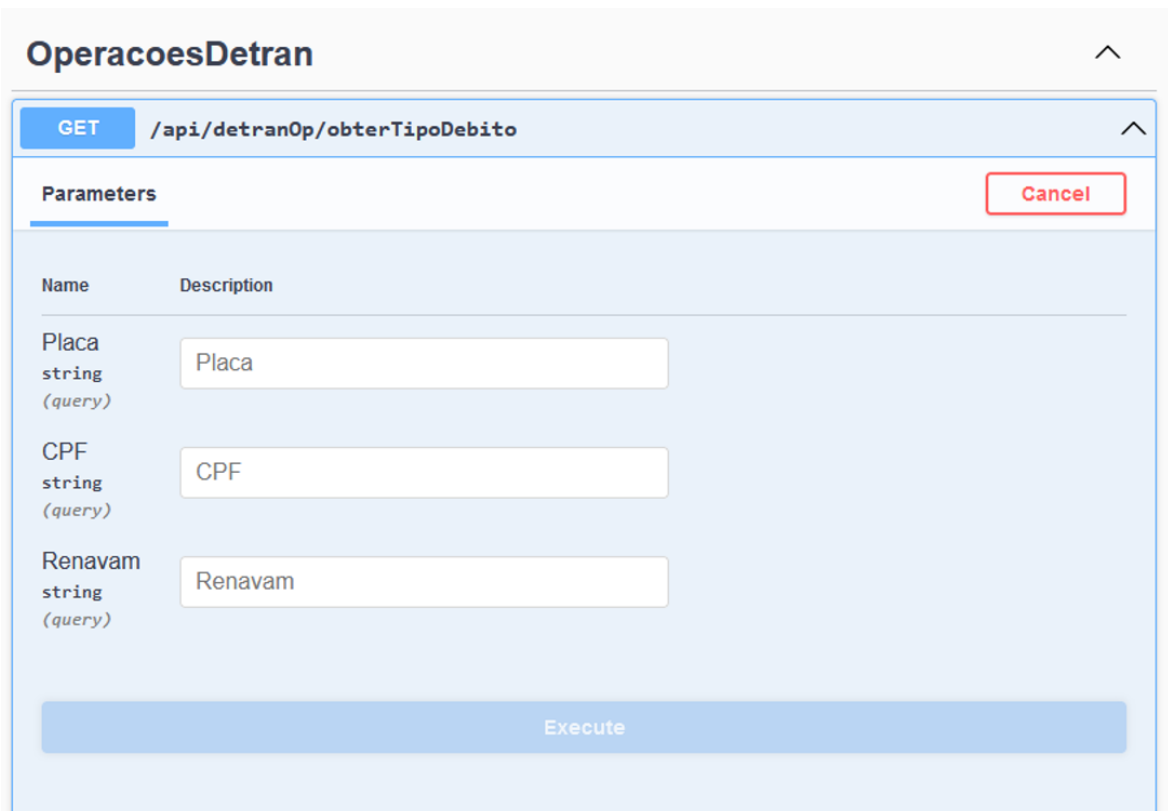
garantir maior confiabilidade durante os testes.

Selenium

O Selenium é um software que auxilia na automatização de testes de aplicações desktop, sites e aplicações web. Utiliza de APIs de automação de navegador fornecidas pelos próprios fornecedores de navegadores, ou seja, Google, Microsoft, Mozilla e outros.

O Selenium utiliza de uma IDE (Integrated Development Environment) para controlar o navegador e executar testes, exemplo: Ele utiliza das APIs fornecidas pela Google, no caso do Chrome, pela Microsoft, no caso do Internet Explorer e Edge, e da Mozilla no caso do Firefox para conseguir identificar os elementos da página e atribuir a eles valores de maneira automática.

A IDE nada mais é do que a ferramenta que os desenvolvedores devem utilizar para desenvolver os casos de teste. Esta IDE funciona como uma extensão dos navegadores de forma que ela consegue mapear todos os componentes da página e fornecer todo o contexto dos elementos identificados, como mostrado nas imagens a seguir.



The screenshot displays the Selenium IDE interface for a GET request to the endpoint `/api/detranOp/obterTipoDebito`. The interface includes a 'Parameters' section with a table of query parameters and an 'Execute' button.

Name	Description
Placa string (query)	Placa
CPF string (query)	CPF
Renavam string (query)	Renavam

Buttons: GET, Cancel, Execute

Figura 8: Teste automatizado utilizando o Selenium, simulando movimentos do usuário.

Na imagem acima podemos identificar uma API(Application Programming Interface) desenvolvida que nos pede três parâmetros. Para testá-la devemos preencher os três

campos e clicar no botão executar de maneira que a API dispare uma requisição para um endereço (url) desejado. Utilizando e configurando o Selenium conseguimos programar os testes para serem executados, com os valores que desejarmos, e assim controlar só não o caminho satisfatório como também prever os possíveis erros e testar o comportamento da aplicação.

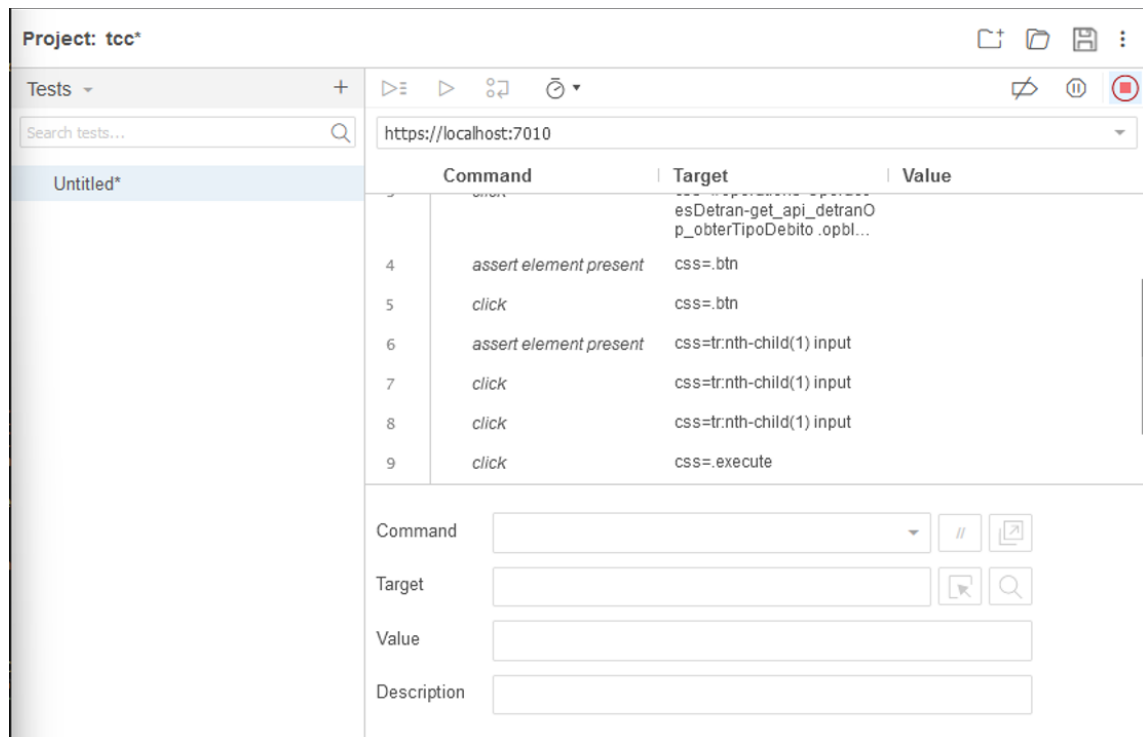


Figura 9: Tela do Selenium mapeando os eventos e elementos da página que está sendo testada.

Fonte: Imagem fornecida pelo autor.

Desta maneira conseguimos não só efetuar testes automatizados, mas também incrementar novos testes de acordo com a evolução do software baseado em cada sprint entregue. Garantimos que ao final de cada sprint todo o teste, incluindo as funcionalidades já desenvolvidas, sejam executadas novamente e que minimize os riscos e erros na aplicação em situações onde uma funcionalidade está diretamente ligada a outra.

Permitimos com o processo incremental incluir nas entregas constantes, previstas no Scrum e no XP, testadas juntamente com o código incremental entregue no final da sprint.

xUnit

O xUnit.net é uma ferramenta de teste de unidade gratuita, de código aberto focado na comunidade .Net Framework (Microsoft). Ela é uma evolução do NUnit v2 que foi uma

adaptação feita do junit, exclusivo para o JAVA, para rodar em projetos do .Net Framework.

O xUnit é a tecnologia mais recente para testes de unidade e suporta as linguagens C#, F#, VB.NET e outras linguagens suportadas pelo .Net Framework. Como o xUnit é compatível também com o .Net Core , ele roda tanto em MacOS, Windows e Linux OS. O xUnit utiliza de classes de teste geradas como modelo de projeto que devem conter os seguintes métodos:

```
[Fact]
public void Test1()
{

}
```

Figura 10: Tela do Selenium mapeando os eventos e elementos da página que está sendo testada.

Fonte: https://miro.medium.com/max/640/1*Gxd63lh-qFKM2evEPba_gg.webp.

Para testarmos a aplicação, precisamos criar um projeto exclusivo para teste e em seguida precisamos adicionar a referência do Selenium no projeto de teste. Desta maneira os pacotes que trabalham com o navegador serão incorporados à aplicação e poderemos escrever códigos que utilizam dos elementos identificados pelo Selenium e juntamente com o xUnit conseguimos manipular estes elementos e inserir valores distintos para os testes.

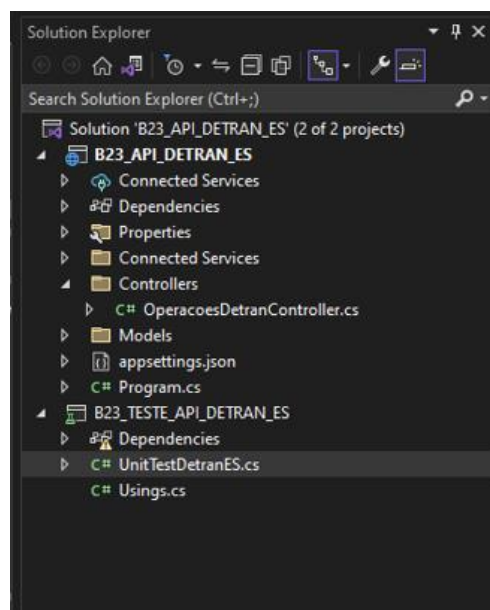


Figura 11: Projeto principal da API juntamente com o projeto de teste da API.

Fonte: Imagem fornecida pelo autor.

Ao executarmos os testes conseguimos verificar quais testes obtiveram sucesso e quais falharam. Como primeiro passo do TDD necessitamos então de codificar o teste baseado em uma funcionalidade, afim de fazer o teste passar.

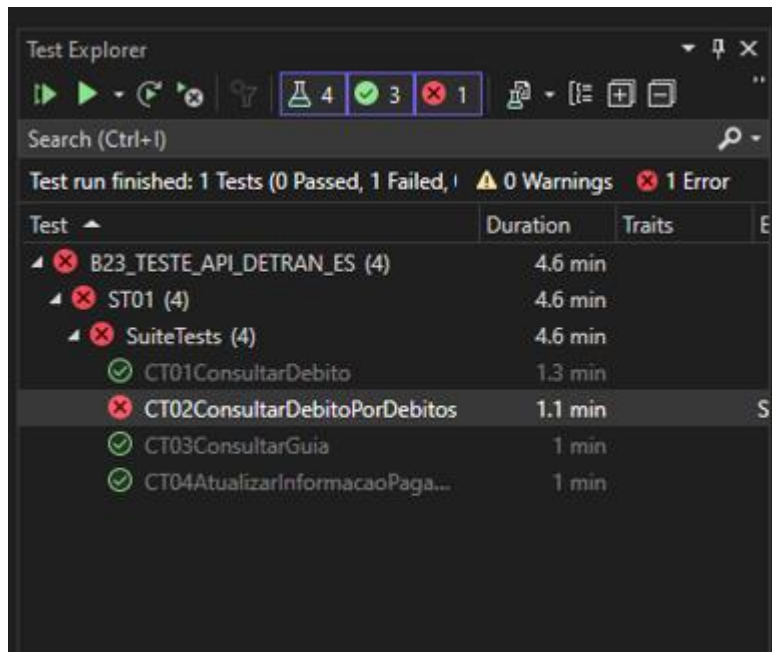


Figura 12: Execução dos testes automatizados mostrando as falhas encontradas.

Fonte: Imagem fornecida pelo autor.

No momento em que o teste falha, por não termos uma resposta satisfatória para os métodos criados, necessitamos então fazê-lo passar codificando de acordo com a funcionalidade presente na tarefa que foi priorizada na sprint do projeto. Uma vez que o desenvolvedor conseguiu de fato executar todos os passos com sucesso, entramos na fase de refatoração, assim conseguimos garantir que além de automatizar todos os testes todas as funcionalidades estão desenvolvidas conforme solicitado.

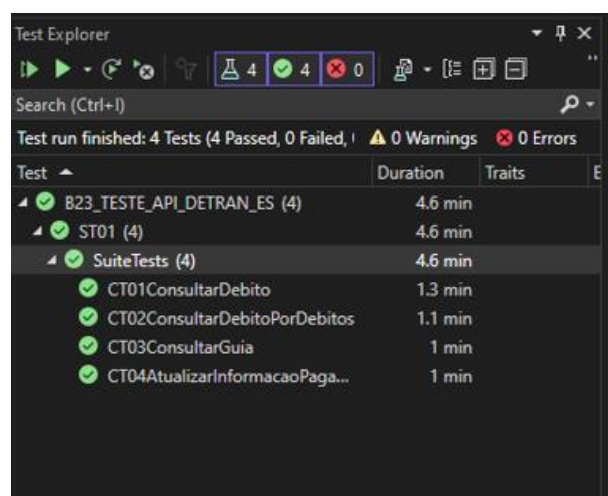


Figura 13: Execução dos testes automatizados mostrando sucesso nos métodos testados.

Fonte: Imagem fornecida pelo autor.

O TDD é uma ótima opção para pequenos ciclos de repetições e se encaixa muito bem na metodologia XP e Scrum. Contudo, pelo fato de ser uma etapa a mais no processo de desenvolvimento existe uma certa resistência mas com o tempo conseguimos perceber o aumento da produtividade da equipe.

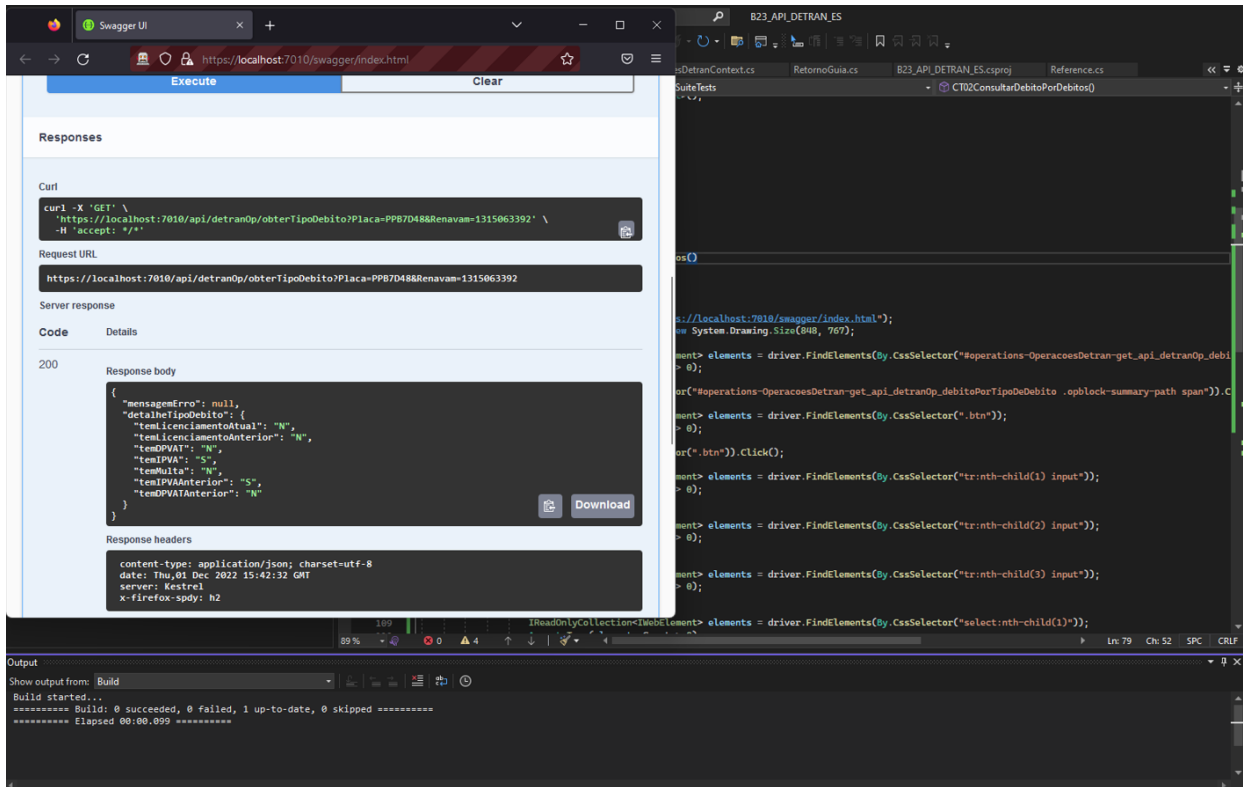


Figura 14: Execução dos testes automatizados mostrando a execução com sucesso da API.

Fonte: Imagem fornecida pelo autor.

Sua implantação existe um novo esforço da equipe que está acostumada com uma outra abordagem. Para diminuir o estresse da equipe com esta mudança, é aconselhável separar uma pessoa específica para codificar e criar os testes de modo que esta pessoa seja responsável por fazer toda a gestão das tarefas que foram aprovadas e quais não foram além de controlar as funcionalidades que irão compor o código funcional que será publicado em ambiente produtivo.

A vantagem é que esta pessoa consegue não só obedecer a ordem do TDD onde deve-se fazer o teste falhar em um primeiro instante mas como ela também terá o controle após o desenvolvimento sendo responsável por toda execução de testes funcionais e automatizados além de devolver ao desenvolvedor responsável o teste falho como também será responsável por incrementar os códigos que passaram na branch principal do projeto.

CONSIDERAÇÕES FINAIS

Em um ciclo de desenvolvimento tradicional, aonde os testes vêm depois da implementação temos como indício vários problemas encontrados, como uma maior quantidade de erros, pois o software somente é testado após o produto ter sido finalizado e estar prestes a entregar ao cliente.

O que acontece em geral é que o tempo com análise e desenvolvimento do software é muito superior à quantidade de testes efetuados e como sempre os prazos estão curtos sobram-se pouco tempo para testar o software, assim o produto é entregue muitas vezes com má qualidade e o custo com a manutenção do software se torna cada vez mais cara.

Os testes escritos antes do código são uma forma de explorar o problema antes de tentar resolvê-los sem total entendimento. Faz com que os desenvolvedores envolvidos com o problema adquiram conhecimentos em cada teste executado, o que fornece segurança para o próximo passo e favorece a simplicidade do código a ser desenvolvido.

Com a programação voltada para o objetivo e no sucesso dos softwares, passando pela maior integração com o cliente e com testes constantes, as melhorias na agilidade e diminuição de custo com o desenvolvimento e manutenção são grandes atrativos para os clientes e para as empresas desenvolvedoras de software.

O manifesto ágil pretende demonstrar que é necessário encontrar um ponto de equilíbrio entre o que escrever na documentação e o que não escrever a fim de evitar situações em que as documentações são armazenadas e nunca mais consultadas.

Os responsáveis pelas especificações devem garantir um conjunto mínimo de informações que permitam desenvolver e testar os requisitos. Desta forma, testadores e desenvolvedores se interagem mais e se ajudam a aperfeiçoarem os requisitos antes mesmo de entrarem na interação.

A interação mais próxima entre a equipe ajuda na divisão de responsabilidades para a automação dos testes, na adoção do desenvolvimento dirigido a testes e na execução regular de grandes suítes de testes.

Para essas equipes a testabilidade não é uma questão de escolha e sim de necessidade evidente. Portanto, cada vez mais se torna vital que o software seja desenvolvido pensando-se na facilidade de testá-lo.

REFERÊNCIAS

BROD, Cesar; **SCRUM – GUIA PRÁTICO PARA PROJETOS ÁGEIS**. 1. ed. São Paulo: Novatec, Agosto/2013.

LOPES, Camilo; **TDD – NA PRÁTICA**. 1. Ed. Rio de Janeiro: Ciência Moderna, 2012.

LEAL, Igor Gonçalves; **REQUISITOS DE METODOLOGIAS DE TESTE DE SOFTWARE PARA PROCESSOS ÁGEIS**. Artigo Científico – Depto Ciência da computação – UFMG. Disponível em: <http://homepages.dcc.ufmg.br/~rodolfo/dcc823-1-09/Entrega2Pos/igor2.pdf>. Acesso em: 14 jan. 2014.

SOARES, Michel dos Santos; **COMPARAÇÃO ENTRE METODOLOGIAS ÁGEIS TRADICIONAIS PARA O DESENVOLVIMENTO DE SOFTWARE**, Artigo Científico – UNIPAC. Disponível em: <www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>. Acesso em: 14 jan. 2014.

SCHUWABER, Ken e SUTHERLAND, Jeff; **GUIA DO SCRUM – UM GUIA DEFINITIVO PARA O SCRUM – AS REGAS DO JOGO**, Disponível em: <https://www.scrum.org/Scrum-Guide>. Acesso em: 20 mar. 2014

WELLS, Don; **THE VALUES OF EXTREME PROGRAMMING**, Disponível em: <http://www.extremeprogramming.org/> Acesso em: 20 mar. 2014

HABIB, Eduardo; **TESTES ÁGEIS**. Revista “A engenharia de software Magazine”, Edição 27. Disponível em: <www.devmedia.com.br> Acesso em: 15 jul. 2014

Alexandre; “**Test Driven Development: TDD Simples e Prático**” Disponível em:< <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>> Acesso em: 05/12/2022

Site: <<https://www.selenium.dev/documentation/overview/>> Acesso em: 05/12/2022.

BIERMAN, Gavin. ABADI, Martin. TORGERSEN Mads., **Understanding TypeScript**. Springer-Verlag Berlin Heidelberg 2014. P. 258. Disponível em: http://link.springer.com/chapter/10.1007/978-3-662-44202-9_11. Acesso em: 05 abr. 2021

CABREIRA, LISSANDRO. “**ESTUDO DE CASO: UTILIZAÇÃO DO MICROSOFT SHAREPOINT COMO FERRAMENTA COLABORATIVA PARA GERENCIAMENTO DE PROJETOS DE INFRAESTRUTURA DE TI.**” UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS MBA - ADMINISTRAÇÃO DA TECNOLOGIA DA INFORMAÇÃO, vol. 1, no. 1, 2010, p. 63. REPOSITÓRIO JESUITA, <http://www.repositorio.jesuita.org.br/bitstream/handle/UNISINOS/5916/Lissandro+Zanchet+Cabreira.pdf?sequence=1>. Acesso em: 20 out 2020.

CARVALHO, ROMÃO e FAROLEIRO. **Governança e Gestão de Projetos de TI: integração COBIT 5 e PMBOK**. Disponível em: <http://capsi.apsi.pt/index.php/capsi/article/download/478/433>. Acesso em: 17 abr. 2021.

DÁVALOS, R. V. **O Uso de Recursos Computacionais para dar Suporte ao Ensino de Pesquisa Operacional**. XXIV Encontro Nac. de Eng. de Produção - Florianópolis, SC, Brasil, 03 a 05 de nov de 2004, p. 8.

GALLO, JÉSSICA. “**COMPARATIVO ENTRE AS VERSÕES 1.2 E 2.0 DA NOTAÇÃO BPMN E SUA APLICAÇÃO EM DIAGRAMAS DE PROCESSOS DE NEGÓCIOS.**” 2012, p. 26, http://repositorio.utfpr.edu.br/jspui/bitstream/1/23520/3/MD_ENGESS_I_2012_11.pdf. Acesso em 25 abr. 2021.

GRILLO, FILIPE, e FORTES. **“Aprendendo JavaScript.”** JavaScript, vol. 1, no. 1, 2008, p. 43. Acesso em: 22 abr. 2021.

KRASNOKUTSKA, INESSA e RIDUSH. **Usage of Self Created Mind Map Environment in Education and Software Industry.** Volume I: Main Conference. ed., vol. 1, Zaporizhzhia Ukraine, Chernivtsi National University, 2018.

LIQUITO, SÓNIA. **“Ferramenta de Gestão de Projetos: Integração de CMMI, TSP e Scrum.”** FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO, vol. 1, no. 1, 2012, p. 92. Repositório aberto, <https://repositorio-aberto.up.pt/bitstream/10216/65530/1/000154229.pdf>. Acesso em: 22 out. 2020.

MEDEIROS, Higor – **“Comparativo entre os bancos de dados MYSQL e MONGODB”**, 2014, Disponível em: <https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/664>. Acesso em: 03 de abril de 2021

PERMANA, et al. **“Scrum Method Implementation in a Software Development Project Management”**. STMIK STIKOM BALI - 2015 REPOSITÓRIO, https://www.researchgate.net/publication/283435871_Scrum_Method_Implementation_in_a_Software_Development_Project_Management

PIRES, CLÁUDIO, et al. **“Software para Gestão de Relacionamento Entre Funcionários e Empresa.”** CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS – CAMPUS V, vol. 1, no. 1, 2015, p.46. REPOSITÓRIO, <http://www.digddv.cefetmg.br/wp-content/uploads/sites/52/2017/11/Produtiva.pdf>. Acessado em: 22 mar. 2021.

PMBOK 6ª, et al. **“Guia do CONHECIMENTO EM GERENCIAMENTO DE PROJETOS.”** PROJECT MANAGEMENT INSTITUTE - PMI, 2017. Acessado em: 22 mar. 2021

SOARES, et al. **“Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software.”** UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS - MINAS GERAIS, 2020. REPOSITÓRIO, <http://www.periodicosibepes.org.br/index.php/reinfo/article/view/146#:~:text=Em%20particular%20s%C3%A3o%20apresentadas%20as,n%C3%A3o%20em%20processos%20e%20planejamentos>. Acessado em: 22 mar. 2021

SILVA, E. P. A. da; SOTTO, E. C. S. **A UTILIZAÇÃO DO IONIC FRAMEWORK NO DESENVOLVIMENTO DE APLICAÇÕES HÍBRIDAS EM ARQUITETURA ORIENTADA A SERVIÇO.** Revista Interface Tecnológica, [S. l.], v. 15, n. 1, p. 97-108, 2018. DOI: 10.31510/inf.v15i1.333. Disponível em: <https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/333>. Acesso em: 27 abr. 2021.

SOLÓRZANO, Ana Luísa V.; CHARÃO, Andrea S.. **Explorando a Plataforma de Computação em Nuvem Heroku para Execução de Programas Paralelos com OpenMP.** In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS) , 2017, Ijuí. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2017 . ISSN 2595-4164.

SOMMERVILLE, et al. **“Engenharia de software, Ian Sommerville.”** PERSON EDUCATION BRASIL - 9ª edição, 2011, REPOSITÓRIO: <http://www.facom.ufu.br/~william/Disciplinas%202018->

2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf. Acessado em: 22 mar 2021

SOUSA, Flávio R. C., et al. “**Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios.**” vol. 1, no. 1, 2011, p. 3.

VALLE, CIERCO, SOARES e JUNIOR, et al. **Fundamentos do gerenciamento de projetos.** 3 ed., Rio de Janeiro, FGV, 2014. GOOGLE LIVROS, https://books.google.com.br/books?hl=pt-BR&lr=&id=_CmHCgAAQBAJ&oi=fnd&pg=PA2&dq=kerzner+gerenciamento+de+projetos&ots=zwJjFQUdvQ&sig=4EjwZrHQZ7wnS8fvkYuiXG0iN6w#v=onepage&q=kerzner%20gerenciamento%20de%20projetos&f=false. Accessed 16 11 2020.

VERNADAT, F. B. **Enterprise Modeling and Integration: principles and applications.** 1. ed. London: Chapman & Hall, 1996.