Notas de Aula

Informações da Aula

Nome do Curso	Teste de Software no ágil
Jornada	Testes Manuais
Professor	Priscila Caimi

Pré requisitos

- [ ] 7 princípios do teste de software
- [ ] Estratégia e Tipos de Teste de Software

Referências

Apostila Teste de Software para Equipes Ágeis - targettrust

Objetivos da Aprendizagem

Aprenda sobre o dia a dia de quem trabalha dentro do contexto ágil. Neste curso você irá aprender sobre o que é metodologia tradicional e ágil, etapas do ciclo ágil, como que funciona o teste nesta metodologia além de entender o que é esperado de você.

Você vai aprender

Aulas
Metodologia Tradicional
Manifesto Ágil
Diferenças do teste no ágil e tradicional
Testes ágeis, uma nova era
Estratégias de Testes para o Ágil
Construindo um processo de teste no ágil
O testador ágil
Automação no ágil
Integração continua

Aula: Metodologia Tradicional

Dentro das metodologias utilizadas dentro do ciclo de desenvolvimento de Software existe as tradicionais e as ágeis. Hoje irei te explicar sobre as tradicionais, quando que elas devem ser aplicadas, seus benefícios e como se é trabalhado dentro do time.

O que é

A abordagem tradicional faz parte do processo de Engenharia de Software que busca trabalhar de forma sequencial dentro do ciclo de desenvolvimento tanto para evolução de produto (desenvolvimento da aplicação) ou suporte ao legado (software pronto em produção).

Dentre as atividades realizadas durante este ciclo podemos citar em alto nível 3 etapas:

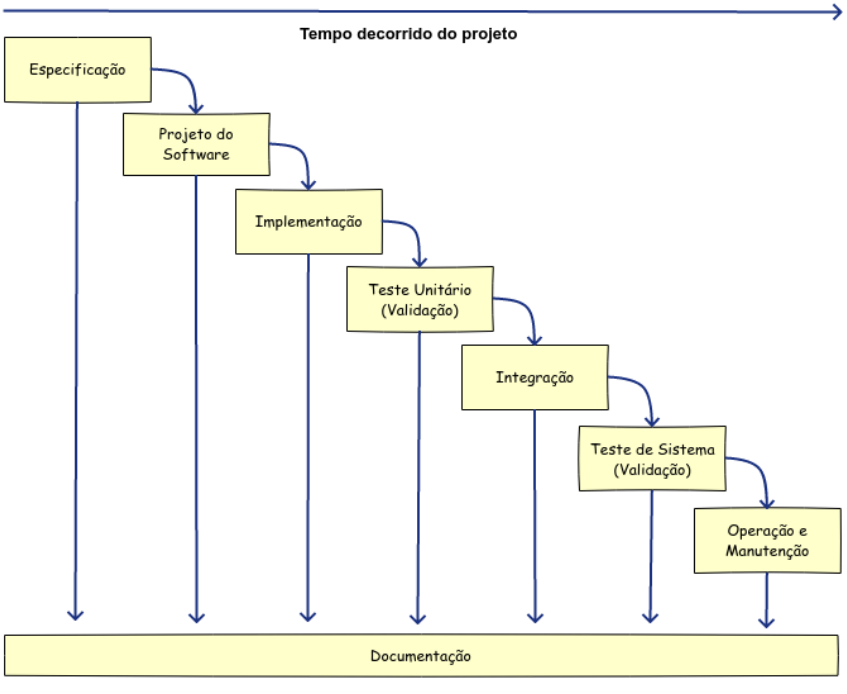
- especificação de projeto,
- implementação e

- teste

Abaixo vamos detalhar dois modelos tradicionais, um sendo o precursor (cascata) e o outro é a sua evolução (espiral).

Cascata

Vindo do inglês Waterfall, foi o primeiro modelo a ser reconhecido como uma metodologia para desenvolvimento de Software. Neste modelo, a sua característica principal dele é que tudo funciona como um esteira de produção, onde uma atividade somente poderá iniciar após a finalização da sua antecessora.



Referência da imagem: https://medium.com/contexto-delimitado/o-modelo-em-cascata-f2418addaf36

Na imagem acima é possível ver as etapas do ciclo de desenvolvimento, porém vale lembrar que cada caso é um caso, então não pode tomar como regra o modelo acima. Cada empresa possui uma necessidade e o modelo é adaptado.

Para algumas empresa este modelo funciona muito bem, porém vale ressaltar uns pontos negativos que sempre será um ponto de alerta para times que trabalham com este modelo.

Retrabalho: Sempre que for encontrado uma problema em qualquer fase deste ciclo, será necessário parar, avaliar onde está o problema, e assim retornar para esta fase ajustar o que for necessário e depois ir etapa a etapa novamente.

Exemplo: ao chegar na fase de validação, foi observado um defeito que foi gerado pela ambiguidade da documentação, neste caso será necessário:

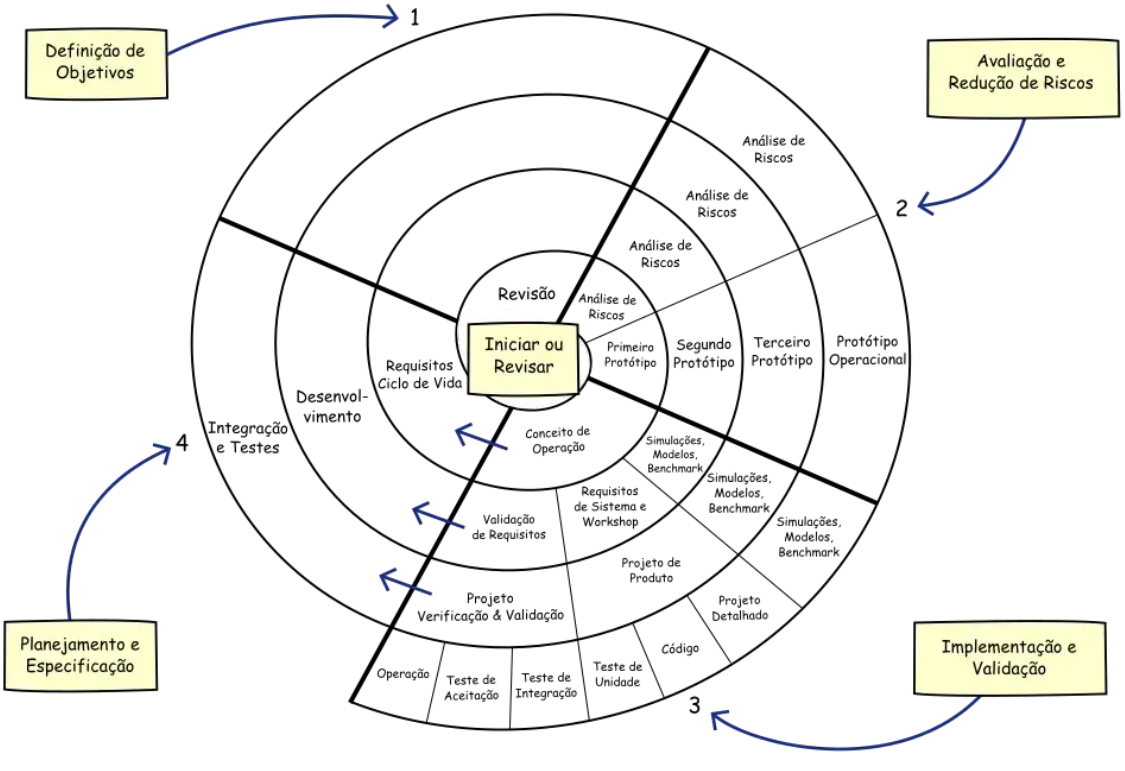
- voltar para a fase de especificação
- evoluir para projeto de software
- reimplementar
- refazer os testes unitários
- refazer a integração
- para somente depois voltar para a fase de validação.

Projetos incrementais: para projetos que sofrem com constante modificação, este modelo não é a melhor escolha visto que a cada nova solicitação deve retornar para o inicio do fluxo.

Liberação de versão: para clientes apressadinhos este modelo irá gerar uma versão para teste e/ou validação de cliente, somente ao final de cada etapa, e dependendo do tempo que dura cada fase pode demorar de 1 a mais de 3 meses entre cada versão.

Espiral

Este modelo foi criado por Barry Boehm em 1988 em busca de evoluir o modelo incremental. Aqui iremos trabalhar de forma incremental e a cada interação, uma nova fase é aborda.



Referência da imagem: https://medium.com/contexto-delimitado/o-modelo-em-espiral-de-boehm-ed1d85b7df

Como vocês podem ver na imagem acima o modelo espiral é categorizado em quatro (4) fases:

- **Definição de objetivo:** Aqui rola o preparativa desta interação, marcos e objetivos são definidos para iniciar a interação.
- **Avaliação e Redução de riscos:** Nesta fase, temos um diferencial que não existia dentro do método de Waterfall (cascata) a analise de risco e prototipagem.
  - **Analise de Risco:** toda funcionalidade deveria ter a sua análise de risco identificada, pois assim antes mesmo de iniciar o ciclo é criado estratégias para contornar e/ou evitar que estas ameaças levantadas nesta análise ocorram realmente, ou seja, se encontrar determinado problema não será necessário voltar ao inicio do fluxo pois uma estratégia já foi criada para isso
  - **Protótipo:** a cada fase incremental, uma nova etapa de prototipagem é executada. Desta forma os protótipos estão sempre alinhados com as novas funcionalidades ou incremento de funcionalidades existentes.
- **Implementação e Validação:** Agora entramos na fase de desenvolvimento e testes. E aqui existe muitas etapas e cada uma delas roda em um ciclo do espiral, ou seja, conforme o espiral vai evoluindo, as técnicas e tarefas também evoluem.
- **Planejamento e Especificação:** Chegou a hora de verificar se os marcos e objetivos planejados realmente foram alcançados. Se alcançados, novos planejamentos são realizados, se não uma nova estratégia é definida.

Aula: Manifesto Ágil

Tudo começou em um encontro em 2004 pela Aliança Ágil, onde várias pessoas se reunirão para discutir sobre processos e documentações que melhorassem o desenvolvimento de software. Desta reunião saíram 4 valores centrais e 12 princípios que são seguidos.

Os 4 valores:

- Indivíduos e interações ao invés de processos e ferramentas
- Softwares executáveis ao invés de documentação
- Colaboração do cliente ao invés de negociação de contratos
- Respostas rápidas a mudanças ao invés de seguir planos

Os 12 princípios:

- Satisfazer o cliente
- Mudanças são bem-vindas

- Entregas frequentes
- Trabalho em equipe
- Comunicação face a face
- Pessoas motivadas
- Software funcionando com medida de progresso
- Ritmo sustentável
- Excelência técnica para melhorar a agilidade
- Simplicidade
- Melhores arquiteturas, requisitos e design veem de equipes auto organizáveis
- Refletir regularmente sobre o que está sendo feito

Aula: Diferenças do teste no ágil e tradicional

Quando analisamos em alto nível a principal diferença do tradicional para o ágil e a interação entre equipes e fases. Enquanto no tradicional, tudo é sequencial, necessitando que uma fase termine para que a outra comece, por outro lado no ágil como os softwares são desenvolvidos de forma incremental, ou seja, a cada nova interação uma melhoria é aplicada ao sistema.

Mas se olharmos mais afundo observamos muitas diferenças, e para isso a tabela abaixo irá mostrar um de/para comparando no tradicional e no ágil.

Tópico	Tradicional	Ágil
Equipe de Teste	Equipes independentes focadas apenas em testar o sistema, conhecidas como equipe de homologação	Testes realizados dentro do próprio time de desenvolvimento
Testes realizados	Testes baseados na interface (caixa preta)	Várias técnicas de testes podem ser aplicados
Requisitos de sistema	Requisitos e casos de testes são criados no inicio do ciclo	Interação constante da equipe com cliente para detalhar o sistema
Alteração do requisito	Deve retornar a fase inicial para reajustar o sistema	Adição incremental ao sistema, não necessário voltar a estaca zero

Padrão de escrita de testes	testes super detalhados, executados por um testador	testes são escritos para complementar a documentação
Papeis no time	diversos papeis definidos (analista de teste,testador, projetistca,etc..)	apenas um único papel é existente: representante da qualidade
Lidando com escopo do projeto	planejamento deve previnir mudanças	mudanças são incorporada ao escopo
Execuções de teste	testes planejados e executados no detalhes	foco em testes exploratórios
Automação de teste	escolher o que automatizar	foco em automatizar
Quantidade dos testes	muitos testes manuais	automatizar o máximo possível

Aula: Testes ágeis, uma nova era

Já entendemos a importância do ágil no nosso ciclo de desenvolvimento. Através da tabela do tópico anterior falando só do nosso dia a dia de trabalho como testador tivemos três grandes mudanças, sendo elas:

- testador está dentro do time, possibilitando assim o teste continuo
- o código desenvolvido é mais simples de ser testado, por se incremental e prezarem por técnicas de desenvolvimento orientados a teste (veremos mais para frente)
- serem incluídos ativamente no ciclo como um todo do desenvolvimento, indo desde a especificação do sistema até o deploy no cliente.

Pode parecer besteira, porém estes três itens irão fazer você ganhar mais segurança no seu dia a dia de trabalho.

Com isso, o ágil também nos ajuda em estabelecermos processos de teste mais enxutos e eficazes, o famoso direto e reto como eu gosto de dizer. Com isso algumas abordagens são discutidas amplamente, como a **Pirâmide de teste** e **Quadrante de Teste Ágil** ambas técnicas serão discutidas no curso de *Padrões e Anti padrões de Teste*. Estes processos nos auxiliam a saber qual teste executar, porque utilizar, qual a quantidade e muitos outros dados.

Nas próximas aulas iremos entrar mais a fundo sobre estratégias de teste, como construir este processo, as skills esperadas de nós, em qual momento devemos automatizar e por fim a integração continua.

Aula: Estratégias de Testes para o Ágil

Quando falamos de estratégia dentro de Teste de Software, estamos buscando trabalhar de forma eficaz, agregar valor para o cliente e não gastar tempo com documentações extensas tornam-se três pontos importante para se trabalhar junto com o time. E para atingirmos estes pilares, três (3) estratégias são discutidas dentro do ágil, sendo elas:

- Testes baseados em risco,
- Testes regressivos automatizados e
- Testes reativos.

Testes baseados em risco

Basear testes em risco para mim é a melhor estratégia para priorizar e setorizar testes, e eu vou te explicar no passo a passo.

Primeiro passo: Saber listar as funcionalidades

Antes de qualquer coisa precisamos saber o que nos vamos testar, e para isso podemos começar pelo básico. Pegar a Funcionalidade pai, e listar tudo que precisa ser testado nela.

Vamos de exemplo: Nós vamos criar uma estratégia de risco para uma funcionalidade de login em qualquer aplicação. E para isso a nossa primeira tarefa é listar tudo que precisa ser testado.

Funcionalidade: Logar em uma aplicação
Utilizar usuário e senha correta
Utilizar usuário e/ou senha incorreta
Utilizar usuário bloqueado
Utilizar usuário inexistente

Agora que temos tudo que precisamos testar, nós vamos para a nossa segunda parte que é **Severizar** cada um destas validações que precisam ser feitas, mas o que é isso?

Severidade é dizer o quanto aquele ponto afeta a sua aplicação. A severidade é divida em três níveis Alta, média e baixa.

Quando temos um teste de alta severidade, quer dizer que se aquela validação falhar, o usuário irá deixar de usar a aplicação.

Em casos de testes com média severidade, são validações que apresentam uma falha porém você consegue utilizar a aplicação e/ou até mesmo aquela funcionalidade.

Já em casos de testes com baixa severidade, são defeitos que não impactam a aplicação e nestes casos podemos colocar os defeitos cosméticos como alinhamento, cor e outras tópicos relacionados mais ao designer da aplicação.

Depois deste bloco de explicação vamos começar a severizar as nossas validações.

Funcionalidade: Logar em uma aplicação	Severidade
Utilizar usuário e senha correta	Alta: pois se falhar o usuário não entra na aplicação
Utilizar usuário e/ou senha incorreta	Média: deve ser exibido um toast dos dados incorretos
Utilizar usuário bloqueado	Média: deve ser exibido um toast dos dados bloqueados
Utilizar usuário inexistente	Média: deve ser exibido um toast dos dados inexistentes

Agora que já temos a severidade, qual é o próximo passo Pri? Repetir este processo para todo o seu sistema, depois que finalizar isso vamos para a segunda parte que é agrupar as validações por severidade.

1. Lista da severidade alta: serão os primeiros cenários a serem executados
2. Lista de severidade média: serão executados após o de alta severidade
3. Lista de severidade baixa: serão executados somente se existir tempo hábil para isso

Quando focamos em executar os testes baseados em risco, procuramos focar os esforços em testes que impactam o uso do cliente em nossas aplicação.

Quando deve ser usado: Você possui uma entrega apertada? esta estratégia é perfeita para isso, pois você consegue focar no que vai impactar na entrega do seu sistema.

Testes regressivos automatizados

Aqui mora um perigo nesta frase, pois automatizar tudo não é a solução. Mas se você já está a algum tempo trabalhando ou até mesmo estudando sobre Teste de Software, você já escutou a frase "*Temos que automatizar TUDO!"* e isso é a maior besteira que você pode escutar de alguém, e no curso de padrões e antipadrões de teste irei te explicar o porque mais detalhadamente além de te apresentar a estratégia correta de automação dentro de uma empresa.

Mas agora que eu explique para vocês o conceito básico do "Automatizar tudo" vamos entrar no tema de regressivos automatizados.

Testes de alta severidade e/ou valor agregado alto ao cliente, devem ser automatizados. Mas Pri o que é alto valor agregado para o cliente? Alto valor agregado ao cliente, as vezes é um cenário que não bloqueia a aplicação, porém o cliente tem tanto apego aquela funcionalidade, que quando apresentado um defeito pode gerar um crise interna com o cliente, por isso nestes casos devem ser automatizados também.

Nestes dois casos, eles devem ser automatizados, tanto a parte funcional que são os clicks em tela (FrontEnd) quando as regras de negócio que se encontram no backEnd. Todos estes testes automatizados devem estar disponíveis em uma pipeline junto com o código do desenvolvedor, pois assim a cada alteração de código, estes testes serão executados validando assim se ocorreu alguma quebra (defeito) na funcionalidade existente. Desta forma as novas implementações não devem afetar funcionalidade antigas, a não ser é claro que seja uma alteração, ai é normal quebrar e uma correção deve ser realizada na automação.

Testes reativos

São testes que não exigem documentação para serem executados, este tipo de teste também possui o nome de teste exploratório que como o nome mesmo diz é explorar o sistema. A vantagem de utilizar esta estratégia de testes é que você consegue encontrar defeitos na aplicação onde as suas validações e automações não encontram, e isso acontece porque você não está seguindo um roteiro e uma passo a passo já viciado (que você sempre executa).

Aula: Construindo um processo de teste no ágil

Vamos começar a falar do nosso processo de teste dentro do ágil? Das etapas do nosso teste e os papeis que exitem?

Planejamento ágil

Como falamos no manifesto ágil, o foco sempre será em pessoas então não deve existir processos e planejamentos robustos e com muitas documentações. Então o planejamento são realizados de forma macro, abordando o que será feito através de planilhas, checklists, ferramentas de rastreio (JIRA, Azure) e mapas mentais.

E como lidamos como o dia a dia de trabalho? Check points diários são realizados para fazer o tracking das atividades e validar o alinhamento e é claro que todo o fluxo deve possuir integração continua (veremos mais a frente), assim se ganha agilidade, inteligência e automação as tarefas.

Plano de teste

Ao contrario do que muitos pensam de que a qualidade é responsabilidade do QA, quando nos encontramos no ágil a responsabilidade é do time como um todo e para isso o planejamento do que será testado e como será testado é discutido entre todos os membros da equipe. Nós como QA devemos trazer para o grupo estratégias, técnicas e tipos de testes que devem ser implementados para assim termos um debate interno de como será feito.

Casos de teste

Agora que foi definido estratégia, técnicas e tipos de teste junto com a equipe, chegou o momento de nós criarmos a nossa validação e neste momento devemos sempre pensar no teste em risco pois assim não geramos possíveis gargalos dentro do teste.

Em conjunto com isso devemos criar a validação de cenário válidos e inválidos das nossas funcionalidades e juntamente com isso utilizarmos a estratégia dos testes reativos ou exploratórios, para conseguirmos validar possíveis defeitos tantos nos cenários previstos quanto nos caminhos alternativos do sistema.

E por último mais não menos importante automatizar cenários que forem criticos ou de alto valor ao cliente.

Papéis

Dentro do ágil cargos tem o nome de papel e cada papel desempenha uma função e atividades especificas para ele, vamos conhecer estes papeis no ponto de vista do teste.

- Desenvolvedores: desenvolvem e realizam testes pela visão de código
- Cliente: Realizaram testes sob a visão da funcionalidade
- Tester: Possui a visão com o todo do sistema e com isso realiza testes em todos os níveis junto com os devs e o cliente.

Report de erros

Ao iniciar a validação dos testes, será inevitável você encontrar defeitos e saber lidar com eles pode aparecer apavorante no inicio mais você vai ver com o passar do tempo que é algo normal e tranquilo de se lidar.

Vou enumerados alguns pontos básicos que irão de ajudar:

1. Defeitos devem possuir severidade: isso irá ajudar na priorização da sua correção.
2. Correção: defeitos abertos devem ser corrigidos na mesma sprint.
3. Escrita clara: saber escrever de forma clara e com uma boa evidência irá ser de fácil entendimento a qualquer pessoa que pegar para correção.
4. Rastreabilidade: indicar onde o defeito foi encontrado, aqui pode informar funcionalidade e/ou caso de teste criado.

Não se preocupe com estes quatro itens, vocês aprenderão com mais detalhes no curso de Bugtracking, onde entraremos no detalhe do assunto sobre bugs.

Aula: O testador ágil

Já falamos sobre funções e atividades, mas o que é esperado de você QA? Vamos descobrir agora!

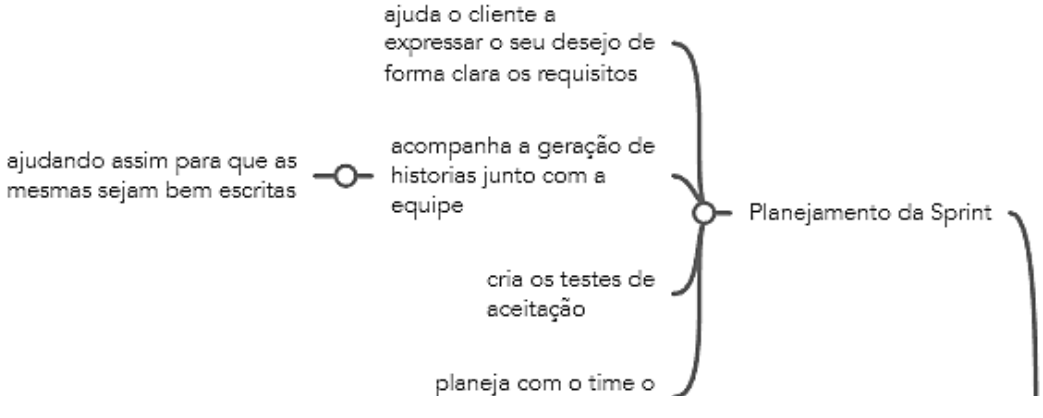
Você como testador dentro de um time ágil é espero os seguintes pontos:

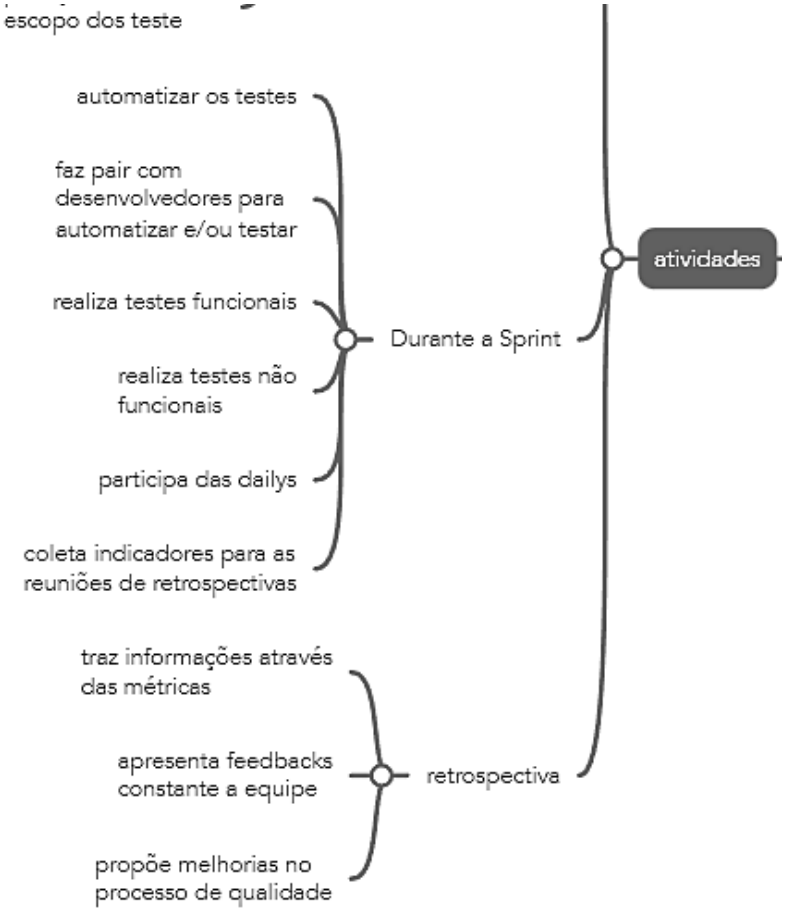
- Ser orientado a qualidade
- Ser pró ativo
- Executar as suas atividades de formas transparentes
- Auxiliar o desenvolvedor a gerar código simples de testar
- "*Se um construtor tem o perfil de **construir** então o testador de **destruir***"
- Ter uma visão critica, comunicação ativa e respeito ao time
- Possuir experiência

Além de possuir as softs skills acima, você terá as seguintes responsabilidades:

- Incentivar o time a busca a qualidade
- Critérios de aceite sejam definidos pelo cliente
- Critérios de aceitação devem ser claros
- Testes devem ser planejados durante o levantamento das histórias
- Os testes devem ser executadas em par: devs + QAs

Quando falamos de atividades que são realizadas por nos testadores dentro do ágil podemos dividir em três (3) fases, planejamento, dentro da sprint e na retrospectiva, veja com mais detalhes no mapa mental abaixo.





Referência da imagem: da própria autora.

Aula: Automação no ágil

Chegamos ao tópico que você irá ou já esta ouvindo e muito dentro da sua jornada, hoje eu vou mostrar para você o que realmente é, os desafio que irá enfrentar e o porque você deve automatizar, vamos nessa?

Objetivo

Ganhar agilidade, segurança e autonomia para seus testes e validações do sistema.

Desafios

Dentro da automação você irá ter vários desafio para entre eles:

- criar uma arquitetura clean code
- saber criar pré condições e steps efetuvis para não ter duplicidade de código
- possuir um ambiente de testes estável para seus testes

Porque automatizar o seu sistema

Automatizar os cenários corretos irá diminuir o seu trabalho repetitivo, acelerar seu processo de teste além de pode implementar várias estratégias diferentes em um único lugar garantindo assim multiplicas validações.

Automação não pode ter inteligência

Isso mesmo, automação é para você não realizar mais processos repetitivos manuais. Se você precisa adicionar cálculos, tratamentos de dados (isso é adicionar inteligência) é sinal que você não deveria estar automatizando este fluxo.

Mas porque eu não devo fazer isso? Se você adicionar inteligência ao seu código para cumprir a regra de negócio, sempre que a mesma for alterada será necessário você refatorar a sua automação para satisfazer a nova alteração da regra de negócio.

Aula: Integração continua

Muito se fala sobre automação, mais não se fala sobre o depois. E vou te contar uma coisa, o fluxo de automação não acaba após você automatizar e sim inicia, pois após finalizar a automação, estes testes devem ser adicionados ao integração continua, ou como falamos muito CI/CD e meus caros alunos, quando adicionamos ao CI/CD a brincadeira começa a ficar interessante, e abaixo foi descrever tudo o que acontece em uma integração continua.

- Identificar mudanças: com automação junto a pipeline do código, nenhuma mudança irá passar despercebido. Código implementado, afeta uma funcionalidade antiga? a automação pode sofrer quebra.
- Build e Deploy: não é só de teste que uma pipeline de integração continua vive, ela gera builds da aplicação, ou seja, gera o pacotinho e o sistema fica disponível ao publico.

- Métricas: muitas métricas podem ser extraídas do código com auxilio de plugin da pipeline, entre eles, cobertura de testes unitários, duplicidade de código, saúde de dependências e muito mais.