Aula 6

Qualidade de Software

Profª Maristela Weinfurter

1

Conversa Inicial

2

Testes e qualidade de software

- Testes ágeis:
  - Qualidade no desenvolvimento
  - Orientação do desenvolvimento (BDD, por exemplo)
  - Inclusão de atividades de testes
  - Utilização das retr.

3

- Os principais testes que podemos automatizar são:
  - testes de regressão
  - de tarefas repetitivas
  - de funcionalidades críticas
  - testes de cálculos matemáticos

4

- Embora automatizamos os testes, ainda assim será necessário executar os testes manuais. Testes ágeis e automação tem uma grande relação, pois garantimos um *feedback* contínuo e rápido, bem como entrega de *software* com qualidade

5

Estratégias de testes

6

Estratégias de testes

- Plano mestre de testes:
  - Fornecer uma estrutura para testes dentro do desenvolvimento do ciclo de vida para que o esforço permaneça focado e dentro do cronograma
  - Identificar as tarefas necessárias para preparar e conduzir testes manuais e automatizados de nível de lançamento entre as equipes

7

Estratégias de testes

- Plano mestre de testes:
  - Renunciar a uma abordagem tradicional de testes em silos pela comunicação clara e frequente de coordenação com todos os membros da equipe de lançamento ágil

8

Estratégias de testes

- Plano mestre de testes:
  - Os itens a serem comunicados incluem quais equipes adicionaram histórias específicas para o *backlog* do produto e, posteriormente, para sua lista de pendências de iteração. Isso inclui a coordenação de atividades de teste para evitar a duplicação de esforços e reduzir a oportunidade de ocorrência de erros

9

Estratégias de testes

- Plano mestre de testes:
  - Garantir o compartilhamento de dados de teste em todas as camadas de teste
  - Representam requisitos funcionais e não funcionais do produto/componente e garantir se esses requisitos foram atendidos

10

Estratégias de testes

- Plano mestre de testes:
  - Se o projeto tiver um componente de terceiros ou de código aberto, então as histórias derivadas devem indicar claramente quem realizará testes funcionais, de estresse e de sistema e deve especificar os critérios de aceitação para o componente

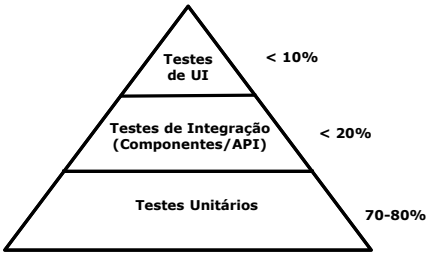
11

Estratégias de testes

- Plano mestre de testes:
  - Fornecer uma estrutura para testes dentro do desenvolvimento ciclo de vida para que o esforço permaneça focado e dentro do cronograma
  - Identificar as tarefas necessárias para preparar e conduzir testes manuais e automatizados de nível de lançamento entre as equipes

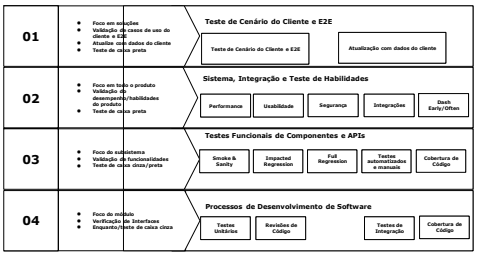
12

Pirâmide de testes em camadas:



13

Cobertura de testes em camadas



14

Tipos de testes ágeis

15

Tipos de testes ágeis

- Teste de unidade
- Teste de recurso/funcional
- Teste de regressão
- Teste de documentação
- Teste do sistema
- Teste de integração

16

Tipos de testes ágeis

- Teste do cenário do cliente E2E
- Testes não funcionais
- Teste de acessibilidade
- Teste de interoperabilidade
- Teste de segurança
- Teste de localização

17

Tipos de testes ágeis

- Teste de prontidão para internacionalização
- Teste de localização
- Teste de suporte
- Teste de capacidade de atualização
- Teste de usabilidade
- Testes de desempenho e escalabilidade

18

- Evolução de testes ao longo do tempo
  - Internet das Coisas (IoT)
  - Big Data
  - Usuários móveis e automação de testes
  - API e microsserviço

19

- Evolução de testes ao longo do tempo
  - Adoção de ferramentas de código aberto
  - IA e aprendizado de máquina
  - Teste de *crowdsource*

20

Testes ágeis segundo o manifesto de testes ágeis

21

Testes ágeis segundo o manifesto de testes ágeis

- Manifesto de testes ágeis:
  - Testar cada etapa e não somente no término do projeto
  - Prevenir *bugs* e não encontrar *bugs*
  - Testar o entendimento e não verificar as funcionalidades
  - Construir o melhor *software* e não deixá-lo quebrar
  - O time é responsável pela qualidade, não somente o time de SQA

22

23

- Q1:
  - Testes de unidade
  - Testes de componentes

24

- **Q2:**
  - **Testes funcionais**
  - **Testes de *story***
  - **Protótipos**
  - **Simulações**

25

- **Q3:**
  - **Teste exploratório**
  - **Cenários**
  - **Testes de usabilidade**
  - **UAT (Teste de Aceitação do Usuário, Alfa/Beta)**

26

- **Q4:**
  - **Teste de desempenho e carga**
  - **Teste de segurança**
  - **Teste de ilidade**

27

- **Os quatro quadrantes servem como diretrizes para garantir que todas as situações possíveis sejam cobertas no processo de teste e desenvolvimento**

28

29

BDD, *code coverage* e testes unitários

- **Ele garante que projetos permaneçam sempre focados na entrega do que o negócio realmente precisa e que todas as necessidades do usuário estejam atendidas**

30

BDD, *code coverage* e testes unitários

- **BDD:**
  - **Exemplos para escrita de comportamento da aplicação ou unidades de código**
  - **Automatização dos exemplos para que o *feedback* seja rápido**

31

BDD, *code coverage* e testes unitários

- **BDD:**
  - **Escrita de comportamento com responsabilidades bem claras**
  - **Funcionalidades questionadas**
  - **Uso de *mocks*, *stubs*, *fakes* e *dummies***

32

Funcionalidades escritas através do GWT (*given, when, then*)

GWT - Given, When, Then	
FUNCIONALIDADE	Enviar um PIX
COMO	funcionalidade segundo normas do BACEN
EU QUERO	efetuar um pagamento instantâneo
PARA	outra pessoa física ou jurídica.

33

Funcionalidades escritas através do GWT (*given, when, then*)

GWT - Given, When, Then	
CENÁRIO	Cliente envia PIX
DADO	que o cliente tem a chave de pix correta
QUANDO	entrar com a chave do pix
ENTÃO	o aplicativo deve validar a chave de pix na outra empresa financeira.

34

- ***Code Coverage*:**
  - **É uma métrica dentro dos testes automatizados e indica o percentual de código em produção que está coberto por testes**

35

- ***Code Coverage*:**
  - **A métrica de *code coverage* deve servir como um indicador de que as coisas estão indo mal e o percentual muito baixo como um grande alerta de que o código está pouco testado, mas busque identificar a raiz do problema**

36

- **Testes unitários:**
  - ***Unit Test* é de responsabilidade dos desenvolvedores durante o processo de implementação do código**

37

- **Testes unitários:**
  - **A redução da quantidade de *bugs* é considerável ao longo da implementação do *software*. Estes funcionam através da comparação de resultados esperados das funcionalidades com o código escrito**

38

- **Ferramentas para Unit Test:**
  - **Unit Testing Framework, Pytest e Locust para linguagem Python**
  - **XCTest para Swift**
  - **Test::Unit, RSpec e Minitest para Ruby**
  - **Mocha, Jasmine, Jest, Protractor e Qunit para JavaScript**
  - **PHPUnit para PHP**
  - **NUnit para C#**
  - **JUnit para Java**

39

Testes automatizados e testes de vulnerabilidade

40

Testes automatizados e testes de vulnerabilidade

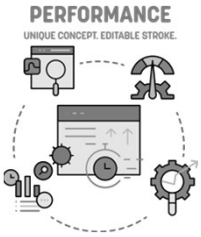
- **Ambiente de teste:**
- **Equipes multifuncionais com desenvolvedores *front-end*, *middleware*, *back-end*, administrador de banco de dados e DevOps**
- **A governança descentralizada faz com que os times escolham suas ferramentas de acordo com suas necessidades**
- **Testes, *deploy* e infraestrutura altamente automatizados, com quase nenhuma intervenção manual**

41

- **Os métodos disponíveis para arquiteturas de *microservices* são:**
  - **testes de *containers***
  - **testes de *containers* de banco de dados**
  - **testes de *containers* de virtualização de serviço**
  - **testes de *containers* de serviços terceiros**

42

- **Dentre as ferramentas de APM encontradas no mercado temos:**
  - **New Relic**
  - **Datadog**
  - **AppDynamics**
  - **Scouter**
  - **Pintpoint**
  - **Loupe**
  - **Stackify Retrace**



Gribiriadam/Shutterstock

43

Gorodenkoff/Shutterstock Magele-picture/Adobestock

44

- **Ferramentas para testes de vulnerabilidade:**
  - **APIsec (SaaS - comercial)**
  - **AppScan (Windows - comercial)**
  - **AppTrana Website Security Scan (Free - multiplataforma)**
  - **CloudDefense (SaaS / OnPremises - comercial - integra CI/CD)**
  - **Grabber (Open Source)**
  - **Zed Attack Proxy (Open Source, multiplataforma)**

45

- **Vulnerabilidades mais comuns:**
  - **Falhas de criptografia**
  - **Injeção**
  - **Quebra de controle de acesso**
  - **Configuração insegura**
  - **Projeto inseguro**
  - **Falha da integridade dos dados**
  - **Falha de identificação e autenticação**
  - **Falsificação de solicitação do lado do servidor**

46

- **Para cobertura dos testes de vulnerabilidades temos:**
  - **Avaliação de rede e *wireless***
  - **Verificação de aplicativos**
  - **Avaliação de *host***
  - **Avaliação de banco de dados**

47

48