Teste e Qualidade de Software

1ºSlide

Qualidade de Software definida

Qualidade: O grau em que um componente, sistema ou processo atende requisitos especificados e / ou necessidades e expectativas do utilizador / cliente.
Qualidade de Software: A totalidade das funcionalidades e caraterísticas de um produto de software que afetam a sua capacidade de satisfazer necessidades explícitas ou implícitas.

Fatores Internos e externos de Qualidade (S. Mcconell)

Externos: Aqueles que os utilizadores / clientes estão cientes ● Correção (construído livro de falhas) ● Usabilidade ● Eficiência (uso de recursos) ● Confiabilidade (longos tempos entre falhas) ● Integridade (estado é sempre constante) ● Adaptabilidade (utilizável em novos contextos) ● Robustez	**Internos**: Aqueles em que o programador está ciente ● Manutenção ● Flexibilidade + Portabilidade ● Reutilização ● Legabilidade + Compreensibilidade ● Testabilidade

Modelo de Qualidade: Fatores de Funcionalidade

Funcionais Conveniência: o Software é adequado para as tarefas destinadas; Precisão: resultados/saidas estão corretas e precisas Interoperabilidade Segurança: Resistir do acesso não intencional ou modificações	**Não-Funcionais** Confiabilidade Usabilidade Eficiência Manutenção Portabilidade

Qualidade de Software: Definição de Trabalho

O grau em que o produto de software:
- cumpre os requisitos funcionais definidos;
- atende às expectativas do cliente w.r.t para os atributos do sistema
- atende às melhores práticas da indústria

Qualidade do Produto requer (P. Farrel-Vinary)

- Uma versão muito inteligente do produto
- Pessoas inteligentes para construir e testar
- Um ambiente estável e útil para se construir

Qualidade de Software: Sistematização

- Não deixe ao acaso: implementar processo de garantia de qualidade de software
- **SQA: é um conjunto de atividades (metodologia) para controlar e monitorizar o processo de desenvolvimento de software para atingir os objetivos do projeto com um certo nível de confiança em termos de qualidade;**
- **Controlo de Qualidade de Software:** Remover os defeitos. Avalia se os produtos de software estão dentro dos padrões de qualidade definidos, recorrendo a inspeções formais e diferentes tipos de testes.

SQA (visa impedilos) != SQC (tem como objetivo detetar e corrigir defeitos)

Práticas SQA

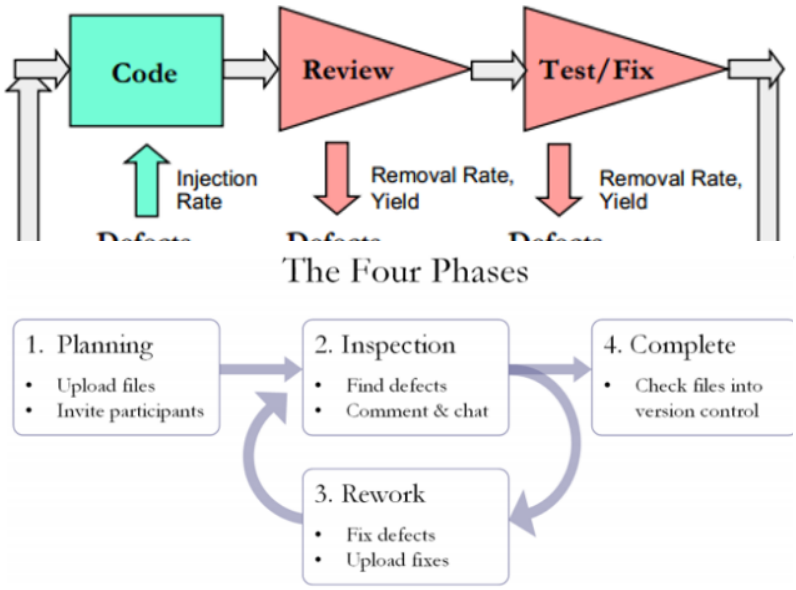
- Testes
- Gestão e configuração de software (Gestão de versões)
- Melhoria de Código (análise estáticas, ...)
- Emissão e Acompanhamento de tarefas de gestão
- Integração Contínua
- Os métodos formais

Verificação vs Validação

Verificação: estamos a fazer o sistema de maneira correta ?
Verificar a consistência dos módulos
Validação: estamos a fazer o sistema correto ?
Verificar subprodutos de trabalho em relação às necessidades e expetativas dos utilizadores

2ºSlide

Code Review: Destina-se a encontrar e a corrigir errps que se encontram na fase inicial do programa melhorando a qualidade de software.



Defeitos Code Review

- Desvios padrões
- Defeitos e Requisitos
- Defeitos de Design
- Manutenção insuficiente
- Especificações de interface

incorretas

Técnicas Leves para revisão de Código (5 tipos de revisão Cohen et al)

- Pros and cons of formal: revisão de processo pesado 3-6 pessoas numa sala com projetor
- Over-the-shoulder: um developer na estação de trabalho do autor, enquanto um autor revê alterações do código.
- E-mail pass-around: Colaboradores discutem os arquivos e fazem perguntas e debatem ideias com o developer
- Pair Programming: 2 developers na mesma estação de trabalho
- Tool assisted review: Autores e revisores usam ferramentas especiais projetadas para revisão de código a pares.

Programação em pares (XP Extrame Programming)

- Todo o código de produção é escrito por pares de programadores
- Cada par trabalho junto numa única estação de trabalho
- Revisão de código contínua
- Os pares são de curta duração
- Não se pode verificar o código que você mesmo produziu

Bad Smells

- Código duplicado
- Métodos muito longos
- Classes grandes
- Longa lista de parâmetros

3ºSlide

Análise Estática de Código

- Análise de tarefas sem executá-lo
- A análise sistemática do source-code / bytcode para vulnerabilidades conhecidas e potencias falhas

Problemas revelados pela Análise Estática

- Fazer referência a uma variável com um valor indefinido
- Interface inconsistente entre os módulos e componentes
- As várias não são usadas
- Código Inacessível
- Violações de padrões de programação
- Vulnerabilidades de Segurança
- Violção de sintaxe, modelos de código e software

Code Refactoring

Refactoring é uma técnica disciplinada para reestruturar um corpo existente de código, alterando a sua estrutura interna sem alterar componentes externos.

- Série de pequenas transformações que preservam a funcionalidade e correção
- Altera o desenho não a funcionalidade
- Código mais limpo/mais fácil de entender e manter

Casos de refactoring comum

Exemplos:

- ➢ Extração de métodos: Selecionar uma parte de um método para formar um novo e substituir a selecção por uma chamada para o novo
- ➢ Interface de Extrato: criar uma nova interface de uso de alguns métodos de uma classe que, em seguida irá implementar a nova interface
- ➢ Encapsular campo: criar/obter e definir métodos para o campo e usar apenas aqueles para aceder ao campo

NetBeans Refactoring

1. Rename
2. Pull Up
3. Push Down
4. Move Class
5. Copy Class
6. Encapsular campos
7. Mudar os parâmetros nos métodos

Estratégias de tratamento de erros em Java

- Programação defensiva
    - evitar causar erros
    - verificar se existe casos problemáticos antes de invocar funções
- Programação / Design por contrato
    - Adicionar restrições especificas no código
    - Os asserts devem ter valor verdadeiro
- Separar excepções lógicas do fluxo “normal”

Excepções Java

- Lidar com circunstâncias e excepções
- Separar fluxo do programa
- São muitas vezes mal utilizados
- Usando excepções economiza-se tempo e problemas

Melhores práticas para a manipulação de exceções

- 3 tipos de throwables
    - exceções verificadas
    - tempo de execução de exceções
    - erro
- Exceções verificadas:
    - Circulam em condições a partir do qual o chamador pode ser razoávelmente esperado para se recuperar
    - Ao lançar uma exceção verificada, você força o chamador para tratar a exceção num cláusula catch
    - Evitar o uso desncessário de exceções verificadas.
- Tempo de execução exceções
    - para indicar erros de programação

    - maioria das exceções de tempo de execução indicam violações de condições prévia
- Erros
    - Convenção forte que os erros são reservados para o uso pela JVM.

5ºSlide
Teste de Unidade

A framework de teste de unidade para Java bem integrado com IDE's e ferramentas de compilação

Classe de Teste Separado: instância e carregadores de classe para cada teste de unidade para evitar efeitos colaterais

Anotações JUNIT: para fornecer a inicialização de recursos e métodos de recuperação: @after and @afterclass

Uma varieadade de **métodos de declaração** para tornar mais fácil de verificar os resultados dos testes.

Integração com ferramentas populares construção e IDE's populares.

Testes de Unidade: Propriedades de bons testes

➔ Automático: Pode ser executado por uma ferramenta de automação
➔ Minucioso: Atende aos objetivos de cobertura desajados (completo, cuidado)
➔ Repetitivo: capaz de ser executado repetidamente e continuar a produzir os mesmo resultados independemente do ambiente
➔ Independente: não depende ou interfere com outros testes.

6ºSlide
Estratégias Falsificadas

- Stubs
    - Fornece respostas enlatadas para chamadas feitas durante o teste
    - geralmente não responde a todos para qualquer coisa fora do que está programando em teste
- Mocks
    - Objetivo pré-programado com as expectativas ou seja, a especificação de chamada é esperada para receber
    - Verificação irá comparar as chamadas recebidas contra as expectativas

- Teste em recipiente
    - recipientes são ativados para permitir o ambiente de teste
    - requer mecanismos para implementar e executar testes num recipiente

8ºSlide

Teste de Software

- Teste: Uma tentativa de provar que alguma caraterística especifica do software e/ou hardware está ausente de modo que este pode ser estabelecido por meios humanos perceptíveis.
- Teste de software: o processo de execução de um sistema de software para determinar se ele corresponde a sua especificação no ambiente pretendido.
- Especificações são cruciais: Estabelecer o que é o comportamento correto

Software bugs

Falhas são de código ou design de erros
Uma falha é uma discrepância com o comportamento observável pretendido (manifestação de falhas)

Testamos para:

- Obter informações valiosas sobre o processo de construção
    - Quão bem que estamos a fazer ?
    - Não é um processo de verificação para falhar !
- Gerir riscos (ganhar confiança nos resultados)
    - P.Ex: Temos cobertura de testes suficientes ?
- Responder à pergunta final
    - O produto está pronto para lançamento ?

A natureza dos testes

Teste ≠ Depuração

- A **depuração** é o processo que os developers passam por identificar a causa de erros ou defeitos no código e fazer correções.
- O **teste** é uma exploração sistemática de um componente ou sistema, com o principal objectivo de encontrar e relatar defeitos. (não correção de defeitos)

Estática vs dinâmica

- **Static**: testes onde o código não é exercido
- **Dinâmico**: execução de software, utilizando os dados de entrada

Princípios gerais - a filosofia de testes

- Teste mostra a presença de erros
    - O teste não podem mostrar que o software é livre de erros
- Testes exaustivos é (geralmente) impossível
    - Teste todas as combinações possíveis de dados e caminhos de código em código?
    - Desafio: saber quando parar.
- Testes iniciais
    - Teste lhe dá informações valiosas sobre o processo de construção
    - Quanto mais cedo um problema (defeito) é encontrado, a menos que os custos para corrigir
    - Se o teste for atrasado, existe um risco real de que não será feito
- O teste é dependente de contexto
    - Teste diferente é necessário em circunstâncias diferentes

Níveis de teste

Teste Unidade

- O código escrito para a unidade atende à sua especificação, antes da sua integração com outras unidades
- O teste de unidade requer acesso ao código que está ser testado

Teste de Integralão

- Objetivo: expor os defeitos nas interfaces e nas interações entre componentes integrados no sistema
- Teste de objetos: Código de interface principalmente.

Teste de Sistema

- Concentra-se no comportamento do sistema inteiro / produto, em um ambiente vivo representante.
- Objeto de teste: o sistema.

Teste de Funcionalidade

- Envolver os utilizadores finais para validar o sistema que irá funcionar de acordo com as suas expectativas
- Teste de objetos: o sistema totalmente integrado; formas e relatórios

É bom o suficiente? O teste de aceitação

Deve ser concluída com êxito

- ➔ Antes que o novo sistema possa ir ao vivo ou substituir um sistema legado.
- ➔ Conclusão pode ser um requisito contratual antes do sistema ser pago.

Caixa Fechado: pelo cliente

- ➔ Todo o sistema é testado como um todo
- ➔ A ênfase é sobre se o sistema atende aos requisitos
- ➔ Utiliza dados reais em situações reais, com utilziadores reais, administradores e operadores

Como testamos

O teste funcional

- ➔ Teste baseado em especificação aka, testes comportamentais, o teste de caixa-preta
- ➔ O teste funcional precisa de uma especificação

Testes não-funcionais

- ➔ Teste para as qualidades do sistema / atributos (por ex.: usabilidade, performance, ...)
- ➔ Precisa de uma especificação

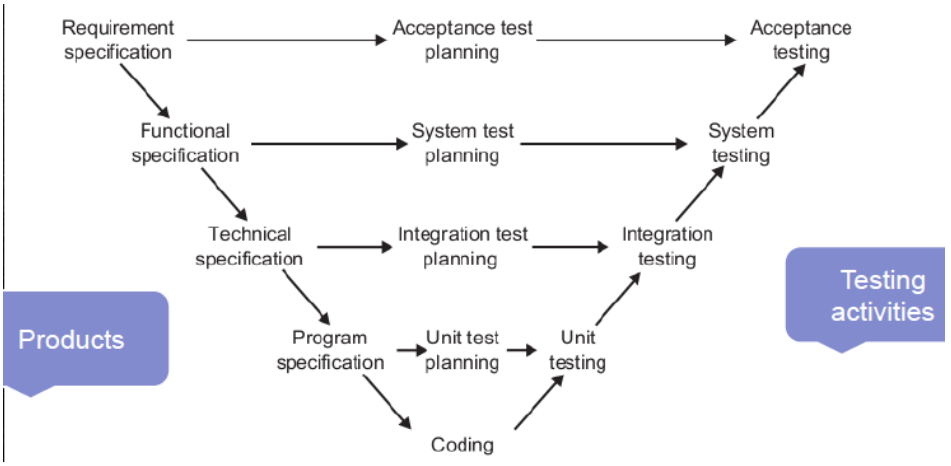
O teste estrutural

- ➔ Medir o quantos testes foram realizados contra algum conceito estrutural
- ➔ Por exemplo: user stories testadas/ todas as stories ; servlets / testados todos os servlets

Teste após o código alterado.

- ➔ Quando Hanges são intruduces, o sistema deve ser testado novamente
- ➔ Testes de regressão: verificações que corrige erros não apresentam funcionalidade inesperadas no sistema (correções foram bem sucedidos)

V-Model



No V Model a execução dos processos acontece de forma sequencial em forma de V. Também é conhecido como um modelo de Verificação e Validação.
V - Model é uma extensão do modelo em cascata e baseia-se em associação de uma fase de testes para cada fase de desenvolvimento correspondente. Isto significa que, para cada fase do ciclo de desenvolvimento, há uma fase de teste diretamente associada. Este é um modelo altamente disciplinados e a próxima fase começa apenas após a finalização da fase anterior.

Vocabulário

Condição de Teste

Alguma caraterística do nosso software que pode verificar um teste ou um conjunto de testes (por exemplo, uma função, transação, caraterística, atributo de qualidade, ou elemento estrutural)

Caso de Teste

Recebe o sistema como ponto de partida para o ensaio (condições de execução); em seguida, aplica-se um conjunto de valores de entrada que deve alcançar um determinado resultado (resultado esperado), e sai do sistema como ponto final (pós-condição de execução).

Procedimento de Ensaio (scripts de teste a.k.a. manuais)

As ações necessárias em sequência para executar um teste

Fluxo de trabalho de design de teste Essencial

1) Decidir sobre uma condição de teste, o que normalmente seria uma pequena parte da especificação para o nosso software em teste;
2) Projetar um caso de teste que vai verificar a condição de teste;
3) Escrever um procedimento de teste para executar o teste

Stories e Scenarios

Unidade básica da funcionalidade, e, portanto, de entrega.

- Captura uma caraterística do sistema
- Define o âmbito do recurso
- Os seus critérios de aceitação.

Título Como [papel] Eu quero [recurso] Assim que [beneficio]	Critérios de Aceitação Given And When Then And

Código Fonte

- **Distribuído**: são vários repositórios autónomos e independentes para casa developer
- **Centralizado**: Segue a topologia de estrela, havendo apenas um único repositório central, mas existindo várias cópias de trabalho, uma para cada developer.

Cobertura

▪ Cobertura de teste é uma medida do grau ao qual um teste exerce alguma caraterística (s) ou de código.
- Minimizar o risco de não ter coberto todas as possibilidades de ter um bug fatal no sistema lançado
- Medidas podem fazer parte dos critérios de conclusão definidos no plano de teste e usados para determinar quando parar

Testes de Cobertura:
- A cobertura de teste de funcionalidade.
- A cobertura de teste de estrutura
- A cobertura de teste de cenário

Estratégias caixa preta vs branca

Caixa Preta	**Caixa Branca**
Várias estratégias - Divisão de equivalência - Análise do valor limite - Erro de adivinhação - Causa-efeito gráfico - Testes aleatórios - Testes de partição	Várias estratégias - Verificação formal - A revisão de código / inspeção - Cobertura ▪ testes Caminhos ▪ cobertura Decisões ramo ▪ testes de Dados de fluxo

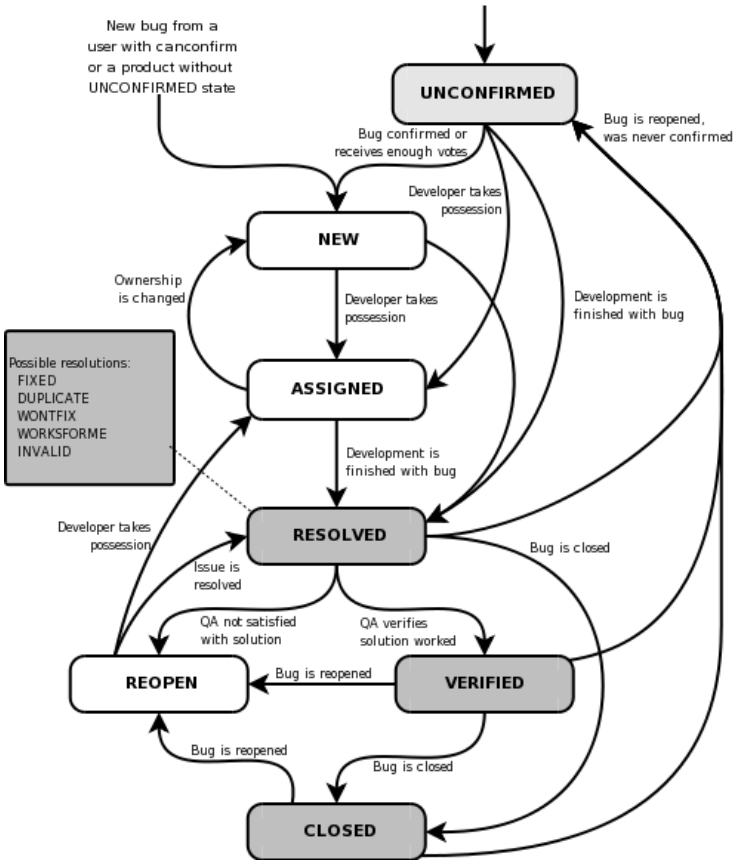
- Testes de valor de limite - Testes baseados em tabelas de decisão	▪ teste orientado a objeto

Exemplos de casos de teste seleções

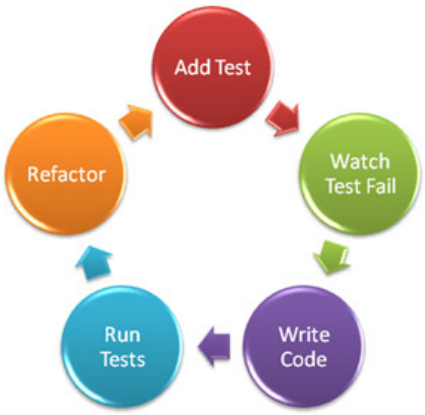
Cobertura de inputs - O objetivo é testar todas as classes de entrada - Aulas de dados - principais categorias de transacções de dados e entradas. ▪ Intervalos de dados - valores típicos, extremos ▪ Dados inválidos ▪ Reversões, recargas, reinicia após falha	Cobertura de todas as funções - O objetivo é testar todas as funções de cada programa de computador ▪ Caminhos através dos programas de computador - Análise de demonstração - Testes de ramo - Fluxo do Gráfico do Programa ▪ Verifique se todos os caminhos são executado pelo menos uma vez - Se todas as declarações e todos os ramos são testado é o programa correto?

Vida de um bug

- Erros são registrados / abertos
- Pode ser aceito / rejeitado
- Seja atribuído
- Código com correções apresentadas
- Retest
- Relatórios, gráficos de defeito, ...

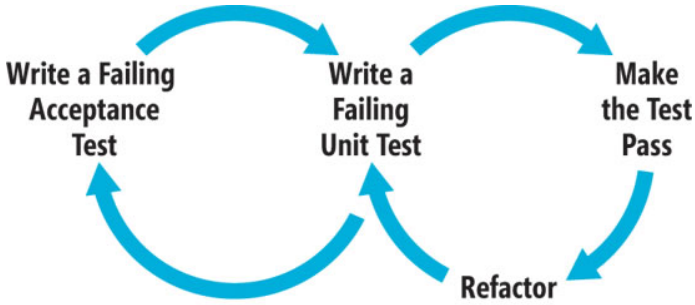


TDD (Test Drive Development): é uma técnica de desenvolvimento de software que se baseia num curto circulo de repetições.



1. Adiconar Testes
2. Verificar se ocorre falhas
3. Escrever código
4. Correr testes
5. Refatorar Código
6. Repetir

BDD (Behavior Drive Development): é uma técnica de desenvolvimento ágil que encoraja a colaboração de developers, setores de qualidade e pessoas não técnicas.



9ºSlide

Botoom line

Inferno Integração Grande esforço, imprevisível para integrar estado da aplicação, não é executável na maioria das vezes.	**Integrar cedo e frequentemente** Propriedade do código de erros pontuais anteriormente partilhado todo mundo é co-responsável.

A essência do que está na prática simples onde toda a equipa se integra frequentemente.

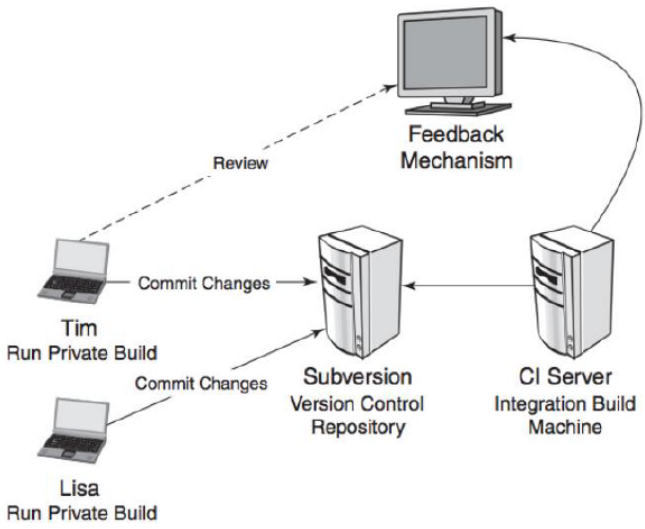
Integração Contínua (CI)

- CI consiste em construir um projeto regular e automaticamente com execução de testes automatizados, testes de qualidade, mecanismos de integração e implementação de binários em máquinas de execução ou de partilhamento de repositórios.
- CI faz com que o processo de desenvolvimento seja mais suave, mais previsível e menos arriscado.

Ferramentas:

Hudson / Jenkins, CruiseControl, Apache Continuum, Atalassian Bambu, …

1. Checkout (ou atualização) de SCM
2. Codificar um novo recurso
3. Execute compilação automatizada na máquina local
   a. Repita # 2 e # 3 até testes passam
4. Juntar a cópia local com as mais recentes mudanças de SCM
   a. Fix e reconstruir até testes passam
5. Commit
6. Executar uma compilação numa máquina limpa
   a. Imediatamente corrigir bugs e problemas de integração



Práticas de Fawlor

1. Manter uma fonte de repositório único
2. Automatizar contrução
3. Manter construção rápida
4. Todos vem o que esta a acontecer
5. Desenvolvimento automatizado

Feedback Contínuo

- ➔ O processo de integração é o mais frequente e o menos doloroso
- ➔ Os erros são mais fáceis de detectar numa fase anterior, perto do ponto em que foram introduzidas:
  - ◆ O mecanismo de detecção de tais erros torna-se mais simples, porque o passo natural no diagnóstico do problema é verificar qual foi a última alteração submetida.
  - ◆ Problemas seguidos de commits atômicos são mais fáceis de corrigir do que corrigir vários problemas de uma só vez.
- ➔ Deve haver um mecanismo eficaz que informa automaticamente programadores, testers, administradores de banco de dados e gerentes sobre o estadoda construção
- ➔ Feedback tem como objetivo gerar reação de uma forma mais precisa e rápida.

Construção "contínua": o processo de construção

- Processo Build é uma série de etapas que transforma os vários componentes do projeto numa aplicação pronta para ser implementada
- As instruções de construção estão descritas num ou mais arquivos de descrição
    - exemplo: pom.xml
- ENTRADA
    - Outro código-fonte, código de teste, as dependências, a documentação, etc
- SAÍDA
    - Outros arquivos executáveis, a documentação do utilizador, bibliotecas embalados, relatórios.

Testes Contínuos

- Verificações de qualidade em todos os níveis do sistema que envolvem todos os indivíduos, e não apenas os elementos da equipa de QA
- A maioria dos testes podem ser automatizados e devem ser executado no pipeline CI a ser realizado repetidamente:
    - Testes unitários, testes de integração, testes de regressão, testes de sistemas, testes de carga e desempenho, etc
- Ferramentas de construção podem ter um papel fundamental a automatizar testes

Maven: É uma ferramenta de gestão de projetos de software e compressão. Baseado no conceito de um modelo de objeto de projeto (POM).

SonarQube: É uma plataforma open source de inspeção contínua da qualidade do código

Compilar: é o ato de transformar código fonte em código objeto.