OBS: Incertezas também estão presentes no BDD!

- BENEFÍCIOS DO BDD
  - → Redução do Tempo de Desenvolvimento
  - → Redução de Custos
  - → Mudanças mais fáceis e seguras
  - → Entregas Mais Rápidas //
- DESVANTAGENS DO BDD
  - → Demanda um elevado nível de colaboração com a equipe de negócio
  - → Estritamente vinculado a metodologias ágeis (cenários CONCRETOS!)
  - → Menos adequado para projetos que envolvem equipes independentes //

OBS$_1$: Os REQUISITOS FUNCIONAIS devem ser documentados desde que ESPECIFICADOS pelo cliente! (Mesmo se forem REQUISITOS NÃO FUNCIONAIS). Caso contrário, podem ser abordados no cenário de testes!

Linting e Padronização de Código

(≠ PADRÃO DE PROJETO)

→ Motivação: falhas de software são provocadas principalmente pelo erro humano, visto que o desenvolvimento NÃO É TRIVIAL e totalmente PROPENSO A ERROS // (independente das ferramentas)

→ Para mitigar essas falhas fazem-necessárias operações de verificação e validação //

* Verificação e Validação

- Verificação: se os requisitos dos clientes estão sendo apropriadamente atendidos //
- Validação: se o produto realmente atenderá a seu uso pretendido quando inserido no ambiente. //

ABRANGEM

- REVISÕES
- AUDITORIA DE QUALIDADE
- TESTES

compreendem dois tipos de análise //

ANÁLISE ESTÁTICA

↓

referente aos "artefatos" do software que não dependem de execução

Ex: DIAGRAMAS E DOCUMENTAÇÃO

ANÁLISE DINÂMICA

↓

dependem da constante execução do software para a análise e verificação de funcionalidades

Ex: TESTES UNITÁRIOS

- ANÁLISE ESTÁTICA

como fazer?

REVISÕES DE SOFTWARE TÉCNICAS E FORMAIS

- INSPEÇÃO
- WALKTHROUGH
- PEER-REVIEW

INSPEÇÃO

- → Analisam e verificam representações de sistema
- → Documento de Requisitos
- → AEC - Análise Estática de Código

Análise automatizada que detecta e destaca anomalias (erros de programação e omissões)

→ Podem não significar um defeito para o código, apenas justificam a análise humana //

→ Fornece feedback objetivo para os desenvolvedores

- Em 1978, Stephen Johnson escreveu o LINT - Uma verificação em C de erros não identificados pelo compilador, regras de tipagem, etc.
- VERIFICADORES ESTÁTICOS (LINTERS)
  - → verificam erros e regras de estilo (importante em uma empresa para facilitar a leitura e a manutenção!)
    - ordem de declaração
    - padrão de nomenclatura
    - espaços em branco, parênteses, etc.
  - → Ferramentas de Linting: checkstyle, pylint, sonarlint, eslint...

* CheckStyle - Linter para Java

- → NECESSÁRIOS: Dependência Maven
- → Dependência checkstyle no pom.xml
- → Módulos e Propriedades

Módulos → REGRAS DE ESTILO DE INTERESSE

Propriedades → ESPECIFICIDADES DE CADA MÓDULO

cada tem suas

→ Severidade: classificação do erro definido ~~pelo~~ de acordo com regra de estilo de interesse //

→ É possível criar um verificador estático de estilo personalizado, porém é importante estar ciente da verificação FORMAL de software.

* Verificador de Erros
  - Débito técnico
  - Bad Smell / Code Smell / Anti-Patterns

Sintomas no código que podem indicar um problema //
  - Ferramentas Ex: SonarQube, Deodorant, e SonarLint

FindBugs, etc.

- BAD SMELLS - EXEMPLOS:
  - código clonado / duplicado

(pode ser substituído por uma estrutura de herança ou extração de classes)
  - "God Class" - classe que faz tudo

(pode ser refatorada em pequenas classes)

- SEI CERT SECURE CODING
  - Computer Emergency Response Team do Software Engineering Institute //
  - Padrões seguros de codificação para dar suporte a uma variedade de linguagens de alto nível (C, C++, Java, etc.)
  - Conjunto de Regras e Recomendações