# RA1 - Ver e Val

**Testar software não é fácil, mas não testar é catastrófico!**

**TESTE → processo de execução de um programa com a intenção de encontrar erros.**

- Adicionar algum valor a um programa para aumentar a qualidade/confiabilidade;
- Remover erros
- Suposição inicial → programa contem erros, testá-lo para conseguir encotrar o maior número possível

**Objetivo**: se for demonstrar que um programa **possui erros** → probabilidade maior de encontrá-los = mais valor ao programa.

## PRINCÍPIOS

1. **Teste demonstra a presença de defeitos, mas não a ausência desses**

   - permanência

   ≠ não existência

2. **Teste exaustivo é impossível**

   Verificar "riscos e prioridades"
   Focar nas necessidades
   Testar "tudo" é praticamente impossível

5. **Paradoxo do Pesticida**

   Revisar e atualizar testes para outros pontos do sistema
   Aumentar a possibilidade de encontrar mais erros

6. **Teste depende do contexto**

   Cada contexto, testes ≠ testes
   Não é possível aplicar o mesmo teste em diferentes tipos de sistema

7. **A ilusão da ausência de defeitos**

3. **Teste antecipado economiza tempo e dinheiro**

   Começar no início do desenvolvimento Planejamento, análise e modelagens Objetivos bem definidos

4. **Agrupamento de defeitos**

   Segue a definição do Princípio de Pareto: Um número pequeno de módulos (20%) contém a maioria dos defeitos descobertos (80%)

Encontrar e consertar defeitos não ajuda se o sistema construído não atende às expectativas e necessidades dos usuários

## ERRO / DEFEITO / FALHA

| | | |
|---|---|---|
| **Engano** | ... | programados trocou + por * |
| **Erro (info)** | desvios do que era esperado da definição | programa executará a * b ao invés de a + b |
| **Defeitos (físico)** | Instruções/Comandos incorretos (bugs) | o programa multiplica ao invés de somar |
| **Falhas (usuário)** | Problemas por processamento incorreto ou inconsistência | programa não conseguiu somar as duas entradas. |

## VERIFICAÇÃO x VALIDAÇÃO

| Processo | Produto |
|---|---|
| reduzindo riscos de falhas de interpretação | dentificar o maior número de erros |
| estruturar um processo sistemático de verificações! | Sucesso = forte planejamento dos testes |
| **Foco nas documentações - Revisões**<br>**Humano, alto treinamento** | **Caixa Branca**<br><br>Arquitetura interna |
| **Foco em atividades - Auditorias** | **Caixa Preta** |

| | |
|---|---|
| Equipes estão respeitando o processo de desenvolvimento? | Requisitos do sistema são atendidos? |

**É possível aplicar Ver e Val para todos os tipos de ciclo de desenvolvimento de software!**

| Principais Ganhos | Principais causas das falhas |
|---|---|
| Torna o ciclo confiável | Ausência de gerência de qualidade independente |
| Garante ações corretivas | Ausência de procedimentos de testes automatizados |
| Evita perda da capacidade de gerenciar o projeto | Aplicação tardia no desenvolvimento |
| Amplia as chances de sucesso do projeto | Ausência de profissionais capacitados |
| Amplia a produtividade do desenvolvimento | Deficiência no planejamento dos testes |
| Auxilia a diminuir o **fator desorganização** | **Sob pressão, os testes são sacrificados** |
| Auxilia a evitar o fator trabalho | Ausência de um ambiente de testes isolado |
| - | Dificultar o acesso do analista de testes ao software |

## NÍVEIS DE MATURIDADE

| | |
|---|---|
| **Unidade** | individualmente pequenas partes do código - funcionem isoladamente? |
| **Integração** | interação entre diferentes módulos/componentes - funcionem juntos? |
| **Sistema** | sistema todo - partes integradas funcioname/atendem aos requisitos? |
| **Aceitação** | sistema atende aos critérios e expectativas do cliente ou usuário final? |
| **Alfa** | interno - dentro da organização, antes do lançamento para o público externo |
| **Beta** | usuários finais fora da org - identificar problemas não encontrados durante o teste alfa. |
| **Regressão** | novas alterações/correções trouxeram novos defeitos em funcionalidades já existentes? |

## HEURÍSITICAS DE NIELSEN

1. **Visibilidade do Status do Sistema** → manter o usuário informado sobre o que está acontecedno, feedbacks
2. **Correspondência entre o sistema e o mundo real** → falar a linguagem do usuário, sem termos técnicos. Informações naturais e com ordem lógica
3. **Controle e liberdade para o usuário** → saída de emergência, “ctrl + z”, lixeiras...
4. **Consistência e Padronização** → consistência entre telas para evitar reaprender
5. **Prevenção de Erros** → evita deslize (quer fazer A mas faz B) ou engano (faz sem querer, entendeu errado): “tem certeza?”
6. **Reconhecimento em vez de memorização** → minimizar o uso da memória, apresentar objetos/ações visivelmente[
7. **Flexibilidade e eficiência de uso** → adaptação ao nível do usuário, teclas de atalho
8. **Design estético e minimalista** → manter apenas informações necessárias, evitar perda de tempo
9. **Ajudar os usuários a reconhecer, diagnosticar e recuperar erros** → mensagens de erro pequenas e simples (sem códigos). Indicar o problema e possível solução
10. **Ajuda e Documentação** → interface intuitiva sem a necessidade delas. Caso necessitadas, devem estar facilmente acessível on-line.