VERIFICAÇÃO & VALIDAÇÃO

VERIFICAÇÃO & VALIDAÇÃO

- **Validação**: Estamos construindo o produto certo?
  - Verificação envolve checar se o software cumpre com suas especificações.
- **Verificação**: Estamos construindo certo o produto?
  - Validação é um processo mais genérico. É necessário assegurar que o software atende às expectativas do cliente.
  - Mostra que o software faz o que o cliente espera que faça, exatamente como foi especificado.

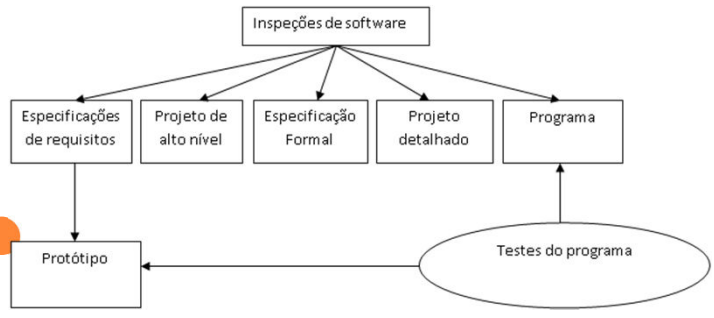
VERIFICAÇÃO & VALIDAÇÃO

Processo de Validação e Verificação (V&V):

- **Inspeções de software**: técnicas estáticas (Não requer que o sistema seja executado) consistem em verificação das representações do sistema, documentação, diagramas e código-fonte.
- **Teste de software**: Executar uma implementação do software com dados de teste e executar saída e comportamento operacional afim de verificar se está sendo executado conforme esperado.
  - Ex. Testes de defeitos, testes estatísticos.

VERIFICAÇÃO & VALIDAÇÃO

Inspeções de software



VERIFICAÇÃO & VALIDAÇÃO

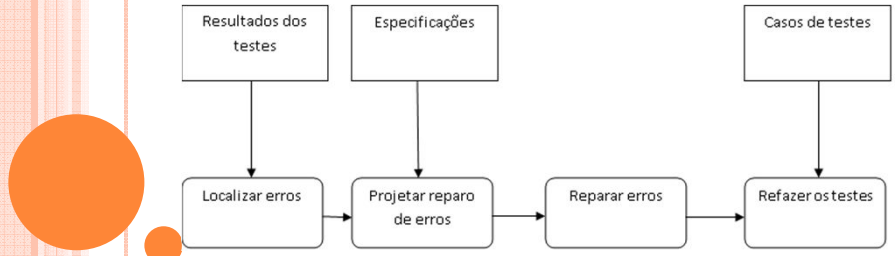
Processo de Validação e Verificação (V&V):

- A meta definitiva do processo de Validação e Verificação é estabelecer a **confiança** de que o sistema de software é adequado ao seu propósito, isto é, significa garantir que o sistema é suficientemente bom para o seu propósito.

VERIFICAÇÃO & VALIDAÇÃO

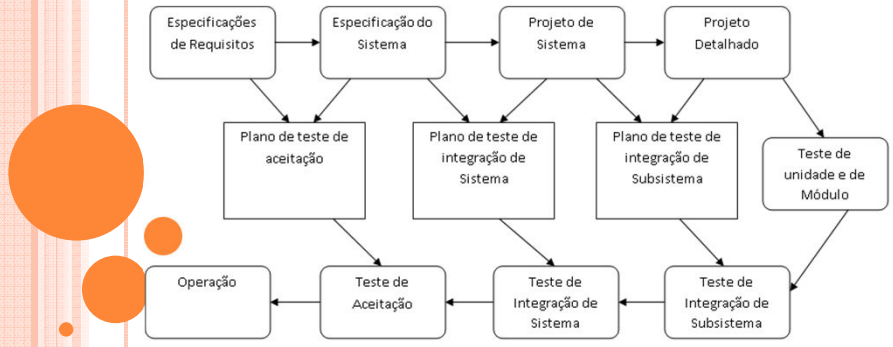
- **Nível de confiança**:
  - Função do software (depende de quanto é importante para uma organização)
  - Expectativas do usuário.
  - Ambiente de mercado.
- A Validação e Verificação são um processo que estabelece a existência de defeitos em um processo de software.
- A operação é um processo que localiza e corrige estes defeitos.

VERIFICAÇÃO & VALIDAÇÃO



VERIFICAÇÃO & VALIDAÇÃO

Plano de teste com planejamento de Validação e Verificação:



VERIFICAÇÃO & VALIDAÇÃO

Métodos formais:

- O uso de especificação matemática formal e da verificação associada é obrigatório nos padrões de defesa do Reino Unido para softwares críticos de segurança.
- Especificação formal pode ser desenvolvida e matematicamente analisada quanto a sua inconsistência. É eficaz para descobrir erros e omissões de especificação.
- Pode ser desenvolvida uma verificação formal de que código de um sistema é consistente com a especificação.
- É eficaz em descobrir erros de programação e alguns erros do projeto.

VERIFICAÇÃO & VALIDAÇÃO

Métodos formais

Vantagens:

- Força uma analise detalhada da especificação.
- Pode revelar inconsistência ou omissões em potencial.

VERIFICAÇÃO & VALIDAÇÃO

Métodos formais

Desvantagens:

- Requer notações especializadas.
- Só são utilizados por pessoal especialmente treinado e não podem ser compreendidas por especialistas do domínio.
- Problemas com requisitos podem ficar ocultos pela formalidade.
- A especificação pode ser matematicamente consistente, mas não especificar as propriedades do sistema realmente requeridas.

VERIFICAÇÃO & VALIDAÇÃO

Métodos formais

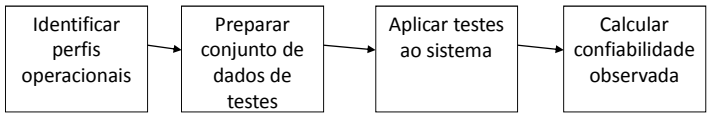
Desvantagens:

- Verificação não trivial toma bastante tempo e requer ferramentas especializadas como provadores de teoremas e perícia matemática.
- O tamanho que o sistema aumenta, os custos de verificação formal crescem desproporcionalmente.
- A prova pode supor um padrão incorreto de uso.

VERIFICAÇÃO & VALIDAÇÃO

Validação de requisitos de confiabilidade

- **Processo de requisitos de confiabilidade**:



VERIFICAÇÃO & VALIDAÇÃO

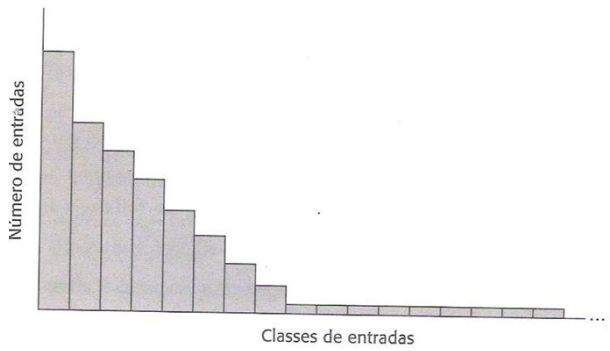
Validação de requisitos de confiabilidade

- **Processo de requisitos de confiabilidade**
- O principal objetivo do teste estatístico é avaliar a confiabilidade do sistema (enquanto os testes para detecção de defeitos têm como objetivo detectar erros).
- **Principais dificuldades**:
  - Incerteza do perfil operacional.
  - Altos custos de geração de dados de teste.
  - Incerteza estatística quando uma alta confiabilidade é especificada.

VERIFICAÇÃO & VALIDAÇÃO

Perfil Operacional:

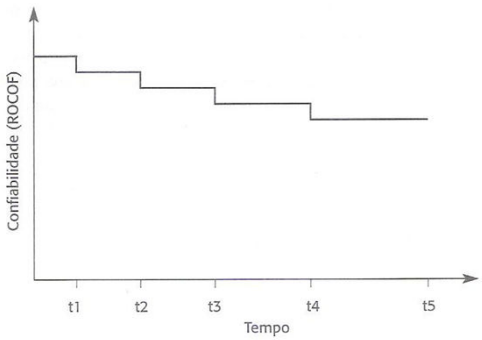
- Perfil operacional muda conforme o usuário conhece o sistema e confia na própria experiência de utilização.



VERIFICAÇÃO & VALIDAÇÃO

Modelos de Confiabilidade:

Modelo de aumento de confiabilidade por etapas iguais.

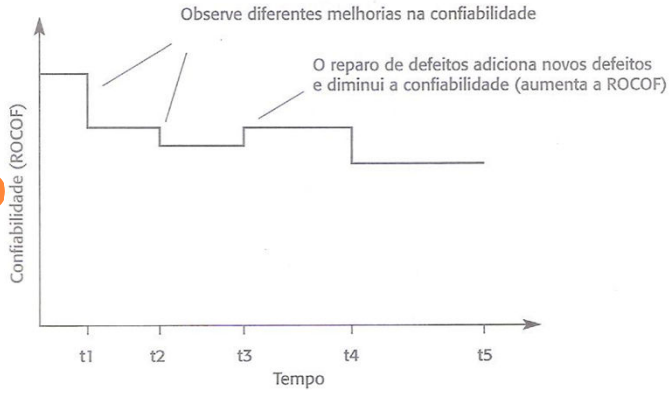


ROCOF: Rate of failure occurrence

VERIFICAÇÃO & VALIDAÇÃO

Modelos de Confiabilidade:

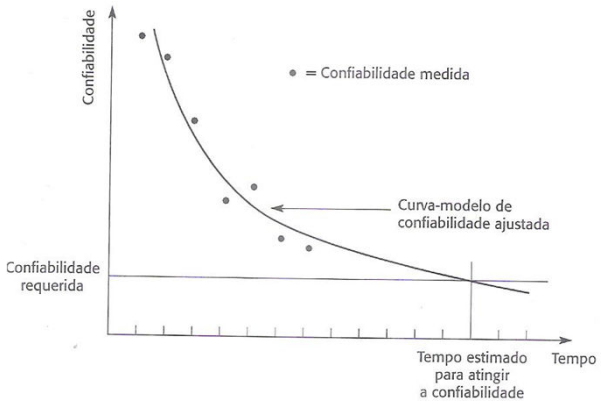
Modelo de aumento de confiabilidade de funções por etapas aleatórias



VERIFICAÇÃO & VALIDAÇÃO

Modelos de Confiabilidade:

Modelo de Previsão de confiabilidade



VERIFICAÇÃO & VALIDAÇÃO

- Devido às taxas de falhas extremamente baixas, que são requeridas em muitos sistemas críticos, os testes estatísticos nem sempre podem fornecer uma estimativa quantitativa da confiabilidade do sistema em razão do elevado número de testes exigidos.

ARGUMENTO DE SEGURANÇA

- É possível desenvolver um argumento de segurança que demonstre que o programa cumpre com suas obrigações de segurança.
- Não é necessário provar que o programa cumpre com sua especificação. **O importante é demonstrar que a execução do programa não pode resultar em um estado inseguro**.

ARGUMENTO DE SEGURANÇA

- A técnica muito utilizada é a prova de contradição.
- Significa supor que o estado inseguro (identificado pela analise de perigo), pode ser alcançado com a execução do programa.
- O código é analisado e demonstra-se que as pré-condições para esse estado de perigo são contraditórias às pós-condições de todos os caminhos conduzem àquele estado. Assim a suposição inicial de um estado inseguro é incorreta.

ARGUMENTO DE SEGURANÇA

Ex. Bomba de Insulina.

Argumento de Segurança:
Demonstrar que a dosagem aplicada de insulina nunca será maior que a dosagem máxima.

Estado inseguro:

CurrentDose(Dosagem Atual) > MaxDose(Dosagem Máxima)

Demonstração: Todos os caminhos do programa levam a uma contradição de asserção de insegurança.

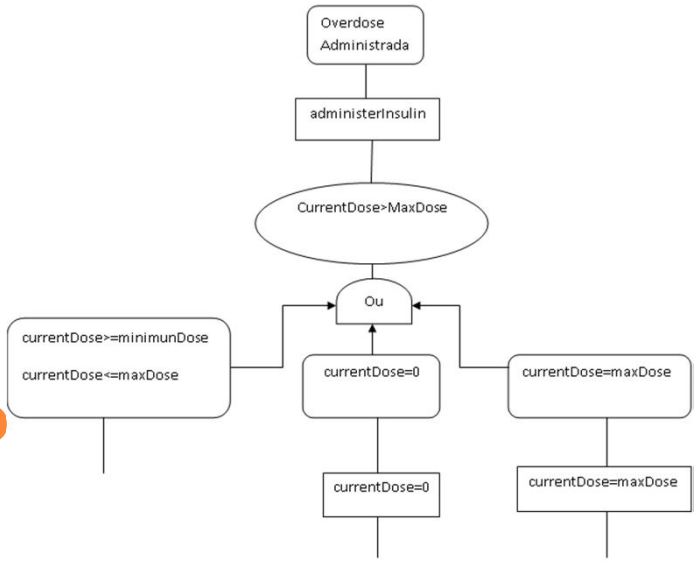
Código de fornecimento de Insulina

```
...
currentDose=computeInsulin();

//checar segurança – Ajustar currentDose se necessário.

IF(previousDose==0)
   {
     IF (currentDose>16)
            {
              currentDose=16;
            }
   }
else
   {
     IF(currentDose>(previousDose*2))
            {
              currentDose=previousDose*2;
            }
    }

IF(currentDose<minimunDose)
   {
    currentDose=0;
   }
else
   {
            IF(currentDose>maxDose)
            {
              currentDose=maxDose;
              }
    }
administrateInsulin(currentDose);
```



Notas de Aula retiradas do Livro

Livro:
ENGENHARIA DE SOFTWARE
SOMMERVILLE, IAN
6 Ed.