Verificação & Validação (V&V)
Técnicas Estáticas

Eliane Martins

Criado: Set/2005
Modificado: Mar/2013

Verificação e Validação (V&V)

Verificação

é o processo de se avaliar um sw a cada fase para determinar se o produto dessa fase satisfaz ao que foi requerido no início da fase

estamos desenvolvendo o produto corretamente?

Validação

é o processo de se avaliar um sw, durante ou após o desenvolvimento, para determinar se o produto satisfaz aos requisitos

estamos desenvolvendo o produto correto?

V&V no ciclo de desenvolvimento

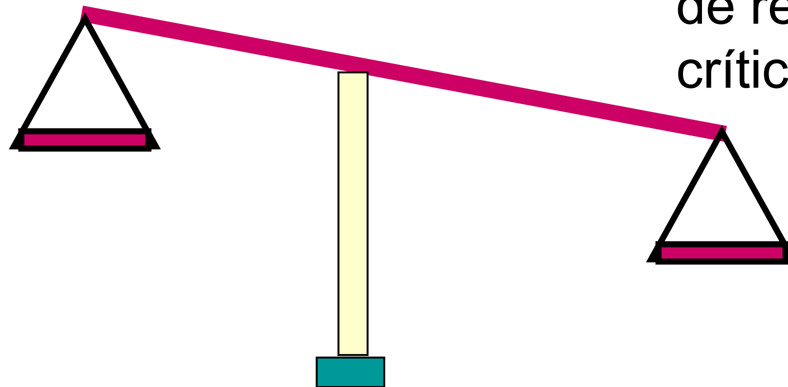
Validação
Verificação

Análise
Especificação
Projeto preliminar
Documento de projeto preliminar
Projeto detalhado
Documento de projeto detalhado
Codificação
Código
…

V&V: fazer ou não fazer ?

- ☺ Permite encontrar falhas mais cedo
- ☺ melhora a qualidade dos produtos
- ☺ torna os requisitos mais estáveis
- ☺ permite acompanhamento contínuo da qualidade e da produtividade $\Rightarrow$ facilita o gerenciamento

- ☹ Aumenta os custos do desenvolvimento
- ☹ aumenta a interação entre equipes
- ☹ Aumenta a documentação
- ☹ Requer compartilhamento de recursos (e/ou dados) críticos



V&V: fazer ou não fazer ?

☞ **quanto mais tarde uma falha é encontrada, maior o custo para corrigi-la**

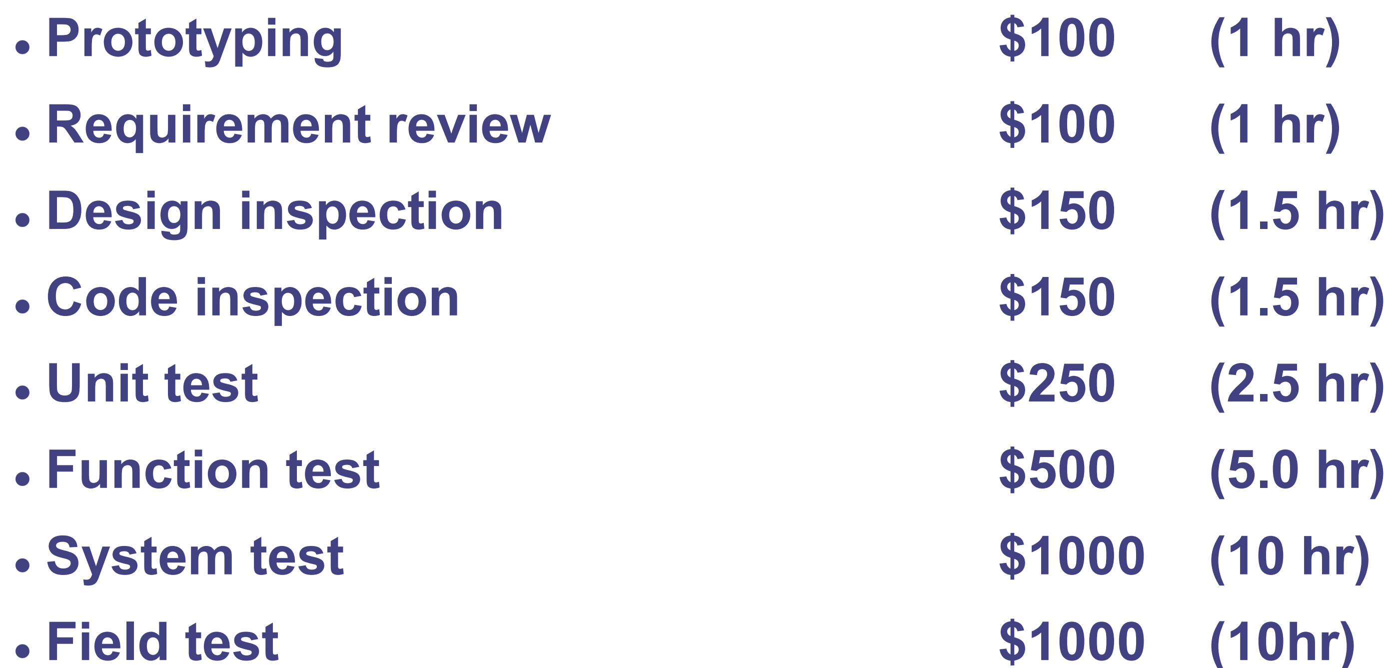
Luong&Stevens
HP, 2005

Bug Removal Efficiency

Typical Bugs Detection Profile

Reference: M.Dyer, "The Cleanroom Approach to Quality Software development", John Wiley & Sons, Inc., 1992

Industry Software Bug fixing Costs



- Prototyping $100 (1 hr)
- Requirement review $100 (1 hr)
- Design inspection $150 (1.5 hr)
- Code inspection $150 (1.5 hr)
- Unit test $250 (2.5 hr)
- Function test $500 (5.0 hr)
- System test $1000 (10 hr)
- Field test $1000 (10hr)

Reference: Caper Jones, "Applied Software Measurements", McGraw Hill, 1991

Luong&Stevens
HP, 2005

Técnicas de V&V

- Estáticas
  - não envolvem a execução do produto
  - visam determinar propriedades do produto válidas para qualquer execução do produto final

  exemplos:
    - análise de modelos (model checking)
    - análise de segurança (safety analysis)
    - prova de correção
    - leitura (documentos de requisitos ou de projetos)
    - construção de protótipos
    - análise estática
    - revisões técnicas

Técnicas de V&V

- Dinâmicas
  - envolvem a execução do produto (código ou modelo executável)
  - visam encontrar falhas ou erros no produto

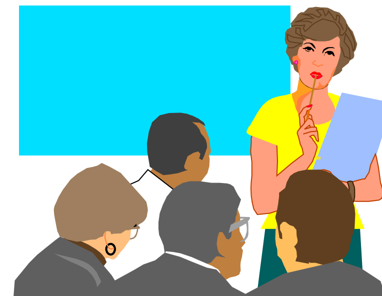
  exemplos:
    - simulação
    - execução simbólica
    - testes

Verificação Estática

Revisão por pares

Reunião organizada na qual um grupo de pessoas analisa um produto ou método com objetivo de melhorar sua qualidade

- Uma revisão tem por objetivos:
  - verificar se o artefato produzido satisfaz as especificações elaboradas em fases anteriores
  - identificar os desvios com relação aos padrões estabelecidos, inclusive fatores que afetem a manutenibilidade
  - sugerir melhorias
  - promover a troca de conhecimento entre os participantes
  - tornar o projeto como um todo mais viável



Revisões e normas de qualidade

- Revisões estão incluídas no SEI Capability Maturity Model (CMM) como uma **área-chave de processo** (KPA) de **nível 3**

- O Capability Maturity Model Integration (CMMI), que engloba sw, engenharia de sistemas e desenvolvimento integrado de produtos, inclui a Revisão como uma **área do processo** de Verificação do Produto

Tipos

- As formas mais comuns são:
  - **Passeio** (*walkthrough*)
  - **Inspeção**
  - **Revisão técnica**

Leitura

- O produto é lido por outra pessoa

☺ fácil de realizar

☺ eficaz quando o leitor for alguém ligado ao projeto (projetista, equipe de testes, equipe de controle de qualidade)

☹ resultados dependem muito do leitor

☹ baixo potencial para encontrar falhas de omissão e inconsistência, especialmente para sistemas mais complexos

Revisão

- O processo:
  - o processo de revisão é bem estruturado, dividido em fases e com procedimentos a serem realizados em cada fase
- Equipe:
  - o número de pessoas envolvidas deve ser pequeno (de 3 a 5)
  - pessoas têm papéis definidos na equipe: autor, revisor, ...
- Diretrizes:
  - os participantes devem receber o produto e a documentação associada dias antes da reunião
  - a reunião tem duração pré-fixada (≈ 2 horas)
  - saídas:
    - ata ou minuta da reunião
    - lista de itens a serem corrigidos

Passeio

Participantes

O autor seleciona os participantes.

Não há papéis específicos além do autor e revisores.

Entradas

o autor estabelece os objetivos da revisão

ex.: verificar completeza e correção; obter consenso quanto a abordagem utilizada no desenvolvimento do artefato

Passeio

Tarefa	Responsável
1. Selecionar os revisores, obter concordância dos mesmos e marcar reunião	Autor
2. Distribuir o artefato entre os revisores antes da reunião	Autor
3. Descrever o artefato para os revisores durante a reunião. Conduzir as discussões sobre os tópicos de interesse, conforme os objetivos da revisão	Autor
4. Apresentar comentários, erros e sugestões de melhorias	Revisores
5. Fazer os ajustes ou correções necessários ao artefato	Autor

Passeio

Saída — Artefato modificado.

Verificação — Não é necessário verificar se as modificações foram feitas conforme sugerido: fica a critério do autor.

Critério de término — O autor realizou as modificações sugeridas.

Inspeção

Participantes: Pelo menos 2 participantes, além do autor. Os participantes têm diferentes **papéis**:

- **Gerente**: gerente do projeto. É quem indica o Moderador.
- **Moderador**: coordena as atividades de inspeção:
  - Escolhe os revisores
  - Marca reuniões
  - Coordena reuniões
  - Produz relatório final
- **Leitor**: lê partes do artefato durante as reuniões de inspeção
- **Relator**: escreve a ata da reunião e classifica os pontos levantados
- **Inspetor**: qualquer revisor; revisa o artefato, registrando as falhas encontradas

Inspeção - Processo

Planejamento → Apresentação → Preparação → Reunião → Correção → Acompanhamento

Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

Inspeção - Processo

Planejamento → Apresentação → Preparação → Reunião → Correção → Acompanhamento

Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

O Autor apresenta o produto para os revisores (opcional)

Inspeção - Processo

Planejamento → **Apresentação** → **Preparação** → **Reunião** → **Correção** → **Acompanhamento**

Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

O Autor apresenta o produto para os revisores (opcional)

Os inspetores revisam o material, buscando as falhas listadas no *checklist* preparado pelo Moderador

Inspeção - Processo

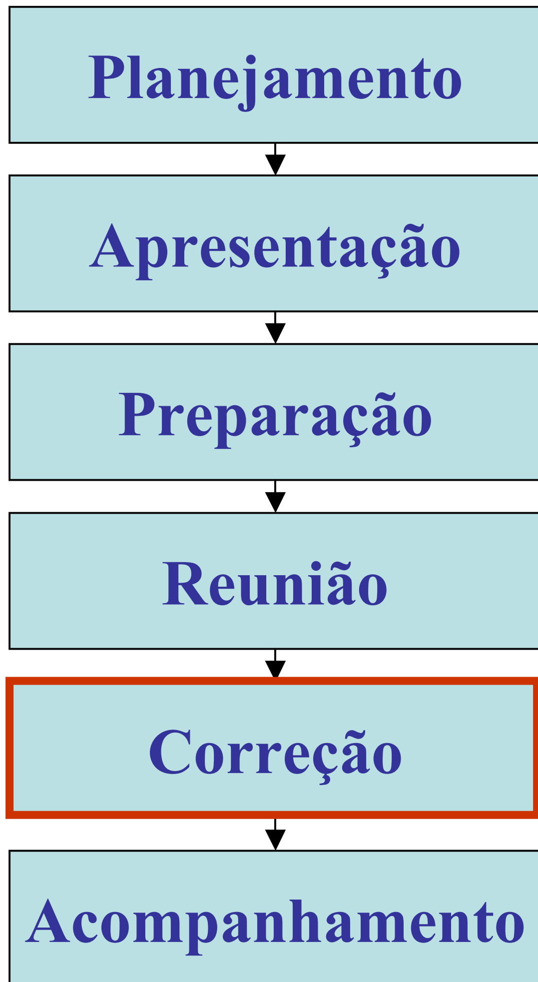
Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

O Autor apresenta o produto para os revisores (opcional)

Os inspetores revisam o material, buscando as falhas listadas no *checklist* preparado pelo Moderador

O Leitor lê as partes do artefato revisado
Os inspetores apontam as falhas encontradas
O Autor esclarece dúvidas
O Relator anota as falhas e outros pontos relevantes

Inspeção - Processo



Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

O Autor apresenta o produto para os revisores (opcional)

Os inspetores revisam o material, buscando as falhas listadas no *checklist* preparado pelo Moderador

O Leitor lê as partes do artefato revisado
Os inspetores apontam as falhas encontradas
O Autor esclarece dúvidas
O Relator anota as falhas e outros pontos relevantes

O Autor corrige as falhas apontadas

Inspeção - Processo

Planejamento

Seleção do Moderador e da equipe
Cronograma
Seleção dos artefatos a serem inspecionados

Apresentação

O Autor apresenta o produto para os revisores (opcional)

Preparação

Os inspetores revisam o material, buscando as falhas listadas no *checklist* preparado pelo Moderador

Reunião

O Leitor lê as partes do artefato revisado
Os inspetores apontam as falhas encontradas
O Autor esclarece dúvidas
O Relator anota as falhas e outros pontos relevantes

Correção

O Autor corrige as falhas apontadas

Acompanhamento

O Moderador verifica se as falhas foram devidamente corrigidas

Exemplo de checklist para especificação

- Todas as informações a serem exibidas para o usuário estão descritas?
- As interfaces externas e internas estão adequadamente definidas?
- As interações entre o sistema e o usuário em situações de erro estão devidamente descritas?
- Cada requisito está descrito de forma clara, concisa e não ambígua?
- Todos os requisitos (funcionais e não-funcionais) são verificáveis?
- O grau de precisão dos cálculos foi especificado?
- Os requisitos não contêm detalhes de projeto?
- Os casos de uso mais complexos foram detalhados em diagramas de seqüência, diagrama de estados, diagrama de atividades ou outro modelo?
- O modelo de dados ou diagrama de classes reflete adequadamente os objetos, seus atributos e relações?

Uma classificação das Falhas (código)

Lista de falhas	**Lista de verificação (*checklist*)**
Falhas nos dados	Todas as variáveis foram inicializadas antes de serem usadas? Todas as constantes têm nomes? O limite inferior para índice de variáveis indexadas deve ser 0, 1 ou qualquer? O limite superior para índice de variáveis indexadas deve ser igual ao tamanho máximo ou igual ao tamanho máximo - 1? Os delimitadores de fim de cadeia de caracteres precisam ser atribuídos explicitamente?
Falhas de controle	As expressões condicionais ou lógicas estão corretas? Todas as iterações (“loops”) terminam? Em comandos do tipo “case”, todas as situações possíveis foram previstas?
Falhas de entrada ou saída	Todas as variáveis de entrada foram usadas? Foram atribuídos valores a todas as variáveis de saída?
Falhas de interface	Todas as funções e procedimentos têm o número certo de parâmetros? Os parâmetros estão na ordem certa? Os tipos de parâmetros formais e reais são compatíveis? Se há acesso a dados globais, todos os módulos têm o mesmo modelo para esses dados?
Falhas de armazenamento	Se uma estrutura dinâmica é utilizada, o espaço foi alocado corretamente? O espaço é liberado após a estrutura não ser mais necessária? Os ponteiros são atualizados corretamente quando uma estrutura dinâmica é alterada?
Falhas de controle de exceção	Todas as condições de erro possíveis foram consideradas?

Informações sobre as falhas

Falha	Descrição
Origem	Fase do desenvolvimento em que a falha foi introduzida
Tipo	• Omissão: algo que devia estar mas está faltando • Comissão: algo errado ou inconsistente com outro • Extra: algo desnecessário • Usabilidade • Desempenho • Outros aspectos: questão, problema de estilo, sugestão
Severidade	• Grave: pode causar defeito ou tem custo de correção mais tarde muito alto • Irrelevante: não causa maiores danos
Localização	Página do documento ou linha do código
Descrição	Descrição concisa do problema

[Wiegers03]

Exemplo de artefatos de inspeção

Sumário de Inspeção

Identificação da Inspeção:

Moderador: ______________________________ Data da reunião: ____ / _____ / ____

Identificação do produto:

Material revisado: ____________________________________

Autor: ____

Descrição resumida: ____

Falhas assinaladas: _________________ falhas (total de falhas assinaladas)

Tamanho do item: _______________ KLOC ou páginas

Taxa de falhas: ______________ (% de falhas por KLOC ou páginas)

Esforço:

Planejamento: _________ pessoas-hora Preparação: _____________ pessoas-hora

(inclui o tempo gasto por todos os participantes dessas atividades)

Apresentação: _________ pessoas-hora Inspeção: _____________ pessoas-hora

(duração da reunião de apresentação ou inspeção × número de participantes)

Resultado da inspeção:

1. Material aceito sem modificações
2. Material aceito com pequenas modificações. Prazo final para correções: ___ / ___ / ___
3. Material rejeitado. Data da re-inspeção: ___ / ___ / ___
4. Inspeção interrompida. Razão: ____
   Data da próxima reunião: ___ / ___ / ___

Material suplementar:

5. Lista das falhas assinaladas.
6. Material revisado com anotações das falhas.

Acompanhamento:

7. Material aceito. Todas as falhas assinaladas foram corrigidas.
8. Material rejeitado. Há necessidade de correções adicionais.
   Prazo para entrega das correções: ___ / ____ / ____

Exemplo - Lista de Problemas Encontrados

Identificador da revisão: Exemplo-Ver-05-04

Data da revisão: 21/5/2004

Material revisado: Espec Req   Revisor: J.F.

Descrição dos problemas:

1. A descrição do caso de uso C-1-A (Gestão de Usuários) não está completa. Falta a descrição dos fluxos de exceção.
2. A descrição do caso de uso C-1-B (Gestão de Fornecedores) deve ser expandida. Sua complexidade requer um diagrama de interação para torná-lo mais fácil de entender.
3. No diagrama de classes falta um relacionamento entre as classes Fornecedor e Mercadoria para que seja possível realizar o caso de uso C-1-B.
4. Erro tipográfico na descrição da classe Usuário. Veja anotação no texto.
5. O requisito não-funcional “o tempo de resposta deve ser adequado” é impreciso e não verificável.

Coleta de dados

- Dados (ou medidas) tanto sobre o produto quanto sobre o processo são coletados durante a inspeção
- Dados sobre o processo: qual a eficácia do processo?
  - número de horas gastas para encontrar falhas
  - número de horas gastas na correção de falhas
  - ...
- Dados sobre o produto: informações e medidas sobre o produto:
  - número de falhas encontradas
  - tipo de falhas encontradas
  - número de falhas / página ou linhas de código
  - ...

Exemplo de métricas [O'Neill 2000]

Sessões	Esforço Prep	Tempo Reunião	Falhas Graves	Falhas Irrelev.	Tamanho Real
2669	161.139min	61.547min	2116	10.926	903.979

Métricas:

Esforço.Prep / Falhas	12,36min
Esforço.Prep / Falhas.Graves	76,15 min
Esforço.Prep. / Esforço.Reunião	0,65
Falhas.Graves / KLOC	2,34
LOC / Hora de reunião	881,26
LOC / Sessão	388,70
Taxa.Falhas / Sessão	4,89

Inspeção OO

- Dificuldade:
  - Problema do espalhamento (*delocalization*):
    - a informação necessária para se entender uma determinada funcionalidade é geralmente espalhada ao longo do código

      $\Downarrow$
    - para entender um trecho de código é necessário seguir uma seqüência de invocação de métodos, navegando através de métodos e classes e percorrendo hierarquias de herança
  - Conseqüências:
    - Técnica sistemática usada no paradigma procedimental é difícil de aplicar
    - Visão estática do código não é suficiente

Técnicas propostas

- Dunsmore, Roper e Wood propõem três técnicas para leitura e inspeção de código OO:
  - **Dirigida por abstração** (*abstraction driven*): o inspetor deve fazer uma Engenhria Reversa de cada método, obtendo uma especificação mais abstrata

Revisão por abstração

public boolean isRegistered (String e) {
int I = 0;
while ((i < theUsers.size ( ))
&& ((((Person)
theUsers.elementAt (i)
.getEmail ( )). equals (e)))
i++;
return I < theUsers.size ( );
}
Passo 1
	– (Person) theUsers . elementAt (i): obtém o iº elemento do vetor theUsers, mudando seu tipo para Person.
	– (Person) theUsers . elementAt (i).getEmail ( ): obtém o email (tipo String) do o iº elemento do vetor theUsers.
	– (Person) theUsers . elementAt (i).getEmail ( ).equals (e): compara e (tipo String) com o email do o iº elemento do vetor theUsers, usando a função String equals ( ) que retorna verdadeiro se duas instâncias de String têm caracteres idênticos.
	Passo 2
		O método isRegistered retorna verdadeiro se a cadeia de entrada e é igual a um dos endereços de email de um dos elementos do tipo Person em uma coleção de usuários denominada theUsers; caso contrário, retorna falso.
		Passo 3

Técnicas propostas

- Dunsmore, Roper e Wood propõem três técnicas para leitura e inspeção de código OO:
  - **Dirigida por abstração** (*abstraction driven*): o inspetor deve fazer uma Engenhria Reversa de cada método, obtendo uma especificação mais abstrata
  - **Dirigida pelos casos de uso** (*use-case driven*): o inspetor lê um código OO usando os modelos dinâmicos usados para descrever os casos de uso, como os diagramas de seqüência
  - **Baseada em lista de verificação** (*checklist-based*): similar ao que já foi visto.
    - A diferença está no checklist, que procura enfocar características de OO: herança, redefinição de operadores, polimorfismo, entre outras.
    - O checklist está organizado de forma a guiar o revisor para que este adquira progressivamente conhecimento sobre a classe.

Exemplo de checklist

	Característica	Questão
	Para cada classe:	
1	Herança	A herança requerida pelo projeto está implementada na classe?
2		A herança é apropriada?
3	Construtor	Todos os atributos são inicializados com os valores apropriados?
4		Se o construtor requer a chamada à super-classe, esta chamada está presente?
	Para cada método	
5	Referência a dados	Todos os parâmetros são usados no método?
...		
14	Comportamento	Todas as atribuições e mudanças de estado estão corretas?
15		Em um *return*, o valor e o tipo retornado estão corretos?
16		O método está de acordo com o que foi especificado?
	Para polimorfismo:	
17	Sobrecarga de método	Se os métodos herdados precisam ter um comportamento diferente, eles são sobrescritos?
18		Todos os usos de um método sobrecarregado estão corretos?

Revisões Técnicas e Inspeções

☞ IEEE 1028-1997- "IEEE Standard for Software Reviews“, além do passeio e inspeção menciona também as revisões técnicas

- A revisão não requer pessoal treinado.
- A revisão não tem papéis específicos para os participantes.
- A revisão não necessita de *checklist*.
- A única saída da revisão é uma lista de ações e ata das reuniões
  - Na inspeção tem-se a lista de falhas, métricas, ...
- Na revisão não há critério rígido para o início
  - Na inspeção, o Moderador deve garantir que a fase de Preparação foi completada pelos revisores
- Na inspeção, cada qual age conforme seu papel, na revisão, não há essa exigência
  - Ex.: o artefato só pode ser lido pelo leitor
- Na revisão, o término é decidido pelo líder, enquanto na inspeção, pode ser necessária uma fase de Acompanhamento

Inspeções e testes

☞ Inspeções e testes são atividades complementares, e não mutuamente exclusivas!

- Muitas falhas podem ser descobertas em uma única reunião de inspeção.
- Inspeções servem para verificar conformidade com especificações.
- Inspeções servem para verificar se convenções e padrões de desenvolvimento são seguidos.

- Os testes necessitam de várias execuções para descobrir várias falhas, pois uma falha pode mascarar outras.
- Os testes servem para verificar conformidade com requisitos do usuário.
- Os testes servem para verificar requisitos de qualidade (não funcionais).

Outras técnicas de Verificação Estática

Análise Estática Automatizada

- Analisadores estáticos de programas:
  - Ferramentas que varrem o código fonte à procura de falhas e anomalias ou para obter métricas
  - Complemento muito útil para a inspeção
  - Falhas que podem ser encontradas:
    - Erros de gramática e ortografia
    - Violação de sintaxe
    - Desvio com relação a padrões e convenções estabelecidos
    - Anomalias de fluxo de controle e de dados
    - Vulnerabilidades que podem ser exploradas e resultar em um ataque ao sistema
    - Falhas que comprometem o serviço oferecido pelo sistema

Análise estática automatizada

- Realizada sobre código fonte ou algumas formas de código objeto
- Informação obtida varia desde a indicação de possíveis erros de codificação (ferramentas lint) até métodos formais que matematicamante provam propriedades do programa (se o comportamento está de acordo com a especificação)
- Uso comercial crescente na verificação de propriedades de programas usados em sistemas críticos e na localização de código vulnerável

[Agostinho Sousa 2006 – Notas de aula. Obtido ago/2010 em: http://paginas.fe.up.pt/~jpf/teach/TQS0607/TP_O1.pdf]

Analisadores estáticos e compiladores

- Compiladores:
  - Realizam análise sintática e semântica da linguagem de programação
  - Convertem código fonte para o executável

- Analisadores estáticos:
  - Complementam análises feitas pelos compiladores
  - Usam técnicas mais sofisticadas de análise (ex.: análise de fluxo de dados) para detectar:
    - Falhas de segurança: vulnerabilidades que podem ser exploradas e resultar em um ataque ao sistema
    - Falhas funcionais que comprometem o funcionamento esperado do sistema (ex.: acesso a posições inexistentes de vetores; buffer overflow)

Classificação das ferramentas (1)

- Os analisadores estáticos podem ser classificados de acordo com suas funcionalidades [Pfleeger]:
  - Análise de código
  - Verificação estrutural
  - Análise de dados
  - Verificação de seqüência

Classificação das ferramentas (2)

- Análise de código
  - Verificação de erros de sintaxe
  - Procura de construções propensas a erros
  - Verificação de itens não declarados
- Verificação estrutural
  - Mostra relações entre os elementos
  - Aponta possíveis fluxos do programa
  - Detecta loops e partes de código não utilizados

Classificação das ferramentas (3)

- Análise de dados
  - Verificação da atribuição de valores
  - Revisão das estruturas de dados
  - Verificação da validade das operações

- Verificação de seqüência
  - Verifica seqüência de eventos
  - Verifica ordem das operações

Falhas mais comuns

- Inicialização de dados
- Exceder limites de espaço (ex.: Overflow)
- Acesso fora dos limites (ex.: ArrayOutofBound)
- Divisão por zero
- Atribuição de valor a variável em expressão condicional
- Má gestão do uso de memória
- Critérios de paradas de loops/recursão
- Falta de validação de parâmetros
- Valor de retorno não utilizado ou não validado
- Injeção de comandos (ou SQL)
- Tratamento de exceção inexistente
- Estilo

Exemplo

Fonte: M.Williams, M. Gaudêncio, UFCG

- Injeção de SQL:
  - Dados de entrada em consultas a bancos de dados são usados para a introdução de acesso não permitido
  - Veja, por exemplo, a consulta abaixo, que visa listar alunos com um determinado nome:

```
String n = input.readLine();
 String statement =
 "SELECT * FROM alunos WHERE nome=" + n;
```

Entrada: a; DROP TABLE Notas

```
SELECT * FROM alunos WHERE nome=a; DROP TABLE Notas;
```

☹ A tabela Notas é removida!

- Ferramenta Splint
  - (http://splint.org/)
  - Substituto do lint
  - Código C

Splint 3.1.1 14 Dec 2006

badcode.c:9:17: Format argument 1 to printf (%d) expects int gets int *:pointer

Type of parameter is not consistent with corresponding code in format string.

(Use formattype to inhibit warning)

badcode.c:9:11: Corresponding format code

badcode.c: (in function main)

badcode.c:13:2: Variable c used before definition

An rvalue is used that may not be initialized to a value on some execution path. (Use usedef to inhibit warning)

badcode.c:15:22: Null storage passed as nonnull

param: undefined_parameter (NULL)

badcode.c:11:14: Parameter argc not used

A function parameter is not used in the body of the function. If the argument is needed for type compatibility or future plans, use /*@unused@*/ in the

argument declaration. (Use paramuse to inhibit warning)

badcode.c:7:1: Definition of unused_function

badcode.c:8:6: Function exported but not used outside badcode: undefined_parameter

A declaration is exported, but not used outside this module. Declaration can

use static qualifier. (Use exportlocal to inhibit warning)

badcode.c:10:1: Definition of undefined_parameter

Finished checking 7code warnings

Analisadores estáticos

Fonte: (Ago/2010) http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

- 1 Historical products
- 2 Open-source or Non-commercial products
  - 2.1 Multi-language
  - 2.2 .NET (C#, VB.NET and all .NET compatible languages)
  - 2.3 Java
  - 2.4 JavaScript
  - 2.5 C
  - 2.6 Objective-C
  - 2.7 Perl
  - 2.8 ActionScript
- 3 Commercial products
  - 3.1 Multi-language
  - 3.2 .NET
  - 3.3 Ada
  - 3.4 C/C++
  - 3.5 Delphi
  - 3.6 Java
  - 3.7 SQL
  - 3.8 Uncategorized
- 4 Formal methods tools
- 5 See also
- 6 External links
- 7 References

Outras verificações

- Trechos de código não utilizados
- Complexidade elevada
- Baixa testabilidade
- Baixa manutenibilidade

Ferramenta para análise de testabilidade

- Testability Explorer (http://code.google.com/p/testability-explorer/)
  - Ferramenta da Google que mede testabilidade através de análise estática
  - Identifica códigos difíceis de aplicar testes de unidade
  - Fornece métricas:
    - Complexidade ciclomática
    - Escore de testabilidade

Uso de Análise Estática

- Inúmeras ferramentas → escolha as mais adequadas para o projeto
- Ter uma equipe dedicada ao uso e atualização da ferramenta
- Atualizar a ferramenta com freqüência
- Escrever código que seja tratável pela ferramenta
  - Reduzir número de falsos positivos
- Corrigir apenas o que é mais importante para o bom funcionamento do produto
  - As ferramentas costumam indicar nível de alarme
  - Cuidado com falsos positivos

Prova de correção de programas

- A verificação é baseada em argumentos matemáticos que demonstram que a implementação é consistente com a especificação
  - a semântica da linguagem de programação deve ser definida formalmente
  - a especificação deve ser formal
- Histórico:
  - McCarthy (1962)
  - Floyd (1967)
  - Hoare (1973) e Dijkstra (1976)

Prova de correção

- Para que possa ser realizada é preciso
  - uma especificação formal do sw
  - uma semântica bem definida da linguagem de programação
- Uma linguagem de especificação tem dois aspectos:
  - estrutura (**sintaxe**): trata dos símbolos utilizados e de como combiná-los
  - significado (**semântica**): trata do significado dos símbolos
- Especificação formal:
  - Possui uma sintaxe rigorosa e uma semântica precisa

Exemplos de linguagens de especificação formal

- **Lógica de predicados**
  - as operações que o sistema realiza são representadas em termos das condições válidas antes e após a execução das mesmas
  - ex.: OCL
- **Algébricas**
  - o sistema é especificado em termos das operações que realiza e das relações entre elas
  - ex.: Larch, OBJ (sist. seqüenciais); LOTOS (sist. concorrentes)
- **Baseadas em modelos**
  - o sistema é definido em termos de estados e de operações que modificam seu estado. Modelo matemático representando estados é construído usando conjuntos e seqüências
  - ex.: VDM, Z (sist. seqüenciais); CSP, RdP (sist. concorrentes)

Lógica de predicados

- Predicado:
  - expressão booleana que pode assumir valores verdadeiro (V) ou falso (F)
  - contém os operadores lógicos usados em linguagens de programação (**and**, **or**, **not**, **xor**), e também os quantificadores para todo ($\forall$) e existe ($\exists$) bem como o operador pertence ($\in$)
  - Ex.:

    $x > y$ and $y > z$

    $\exists\ i, j, k \in M .. N \mid i^2 = j^2 + k^2$

    $\forall x\ \exists y \mid x > y$

Uso da lógica de predicados

- Pré-condição
  - especifica condições válidas para as entradas de uma função
- Pós-condição
  - especifica condições válidas para as saídas de uma função
- Invariante
  - Especifica condições que devem ser verdadeiras durante toda a execução
  - Invariantes de laços: relaciona as variáveis usadas em laços. Devem ser verdadeiras: (i) antes da primeira iteração (na entrada do laço) (ii) após cada iteração (iii) ao final da iteração (iv) se o laço não é executado
  - Invariantes de classes: condições que devem ser satisfeitas por todos os objetos da classe

Exemplo

```
// função que calcula o maior (max) e o menor (min) elementos
// de um vetor x[ nx ]
…
assert ( nx > 0 )
min = x[0];
max = x[0];

tam = nx;
for ( int i=0; I < nx; ++i) {
    if ( x[i] < min ) min = x[i];
    if ( x[i] > max ) max = x[i];
    assert ( (min < = max) && (x[i] <= max) && (x[i] > = min) &&
              (nx = tam) );
}
…
```

pré-condição

invariante de laço

Prova de correção

pré-condição

```
{
   assert ( true );
   read (a, b)
   x = a + b;
   write ( x );
   assert ( x = a + b );
}
```

pós-condição

Prova de correção

pré-condição

```
{
  assert ( true );
  read (a, b)
  x = a + b;
  write ( x );
☞ assert ( x = a + b );
}
```

pós-condição

Dado que:

1. Antes de executar **write**: x = a + b
2. **write** não altera o valor de x
3. Atribuição atribui a x o valor de a + b, sendo que a e be são os valores lidos
4. **read** sempre retorna valores válidos de entrada, e não condições de erro ou valores inválidos

Então o programa está correto

substituição por retrocesso (backward substitution)

Prova de correção: considerações finais

- Não existem provas para todos os programas.
  - programas que tratam interrupções;
  - comandos com efeitos colaterais
- Desenvolver prova de correção tem custo alto (precisa de ferramenta) por isso não é usada na prática na maioria dos projetos de sw.

Processo Sala Limpa

- Objetivo:
  - prevenção de falhas: evitar falhas usando métodos rigorosos e precisos
- De onde vem?

Sala-limpa

- “Sala na qual é controlado o nº de partículas em suspensão, sendo construída e usada de forma a minimizar a introdução, geração e retenção de tais partículas na sala”.

  Ref: International standard ISO14644-1 1ª ed. 1999-05-01
- Usadas em algumas manufaturas e em ambientes de pesquisa
- Uma sala comum contém tipicamente 35 milhões partículas por m^3, com tamanho $\geq 0,5\mu m$ → escala ISO9
  – Escala ISO8 → 100 mil partículas/ m^3, com tamanho $\geq 0,5\mu m$ ou 700 partílucas / m^3, com tamanho $\geq 5\mu m$

Sala limpa usada na produção de micro-sistemas

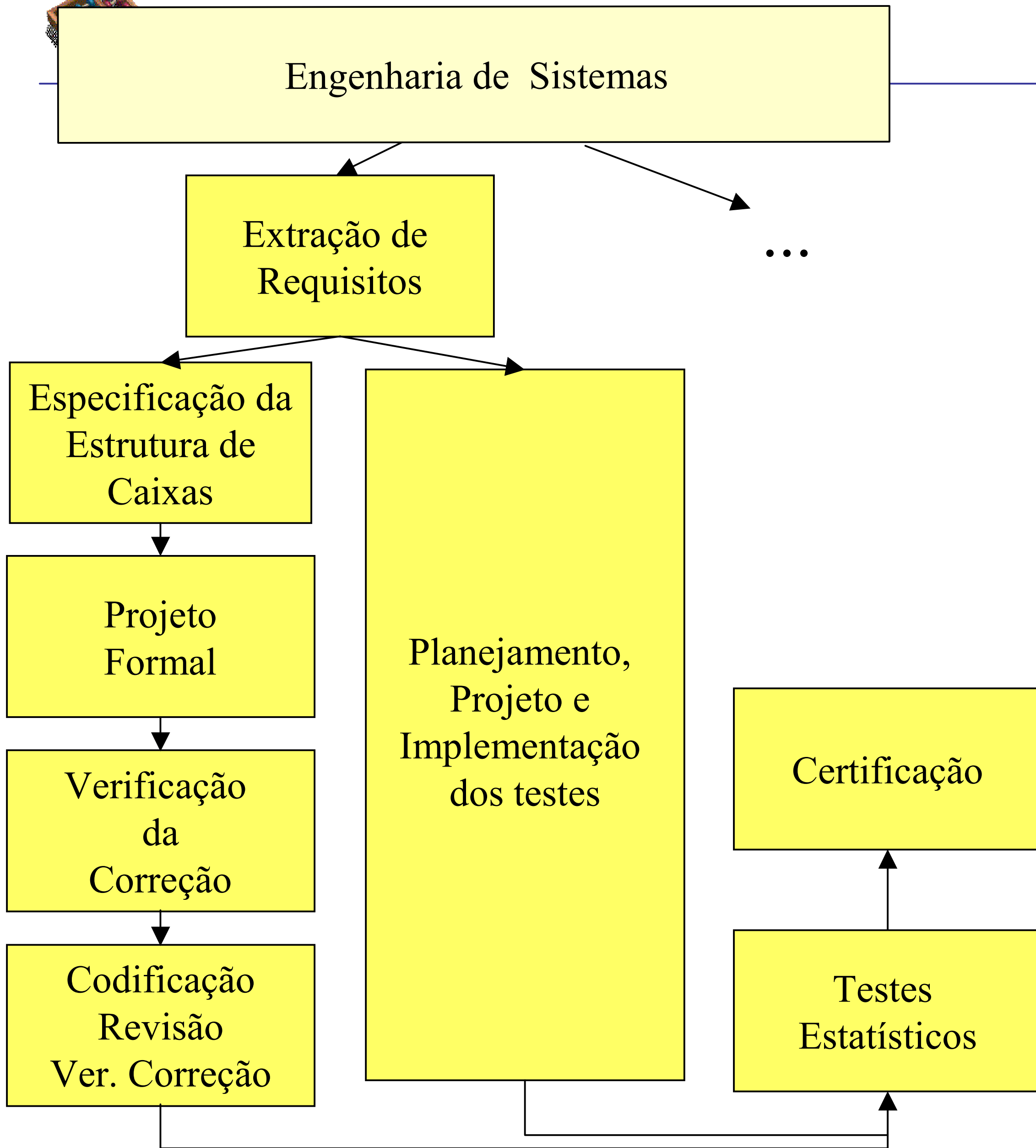
Sala limpa usada na manufatura de micro-eletrônicos

Fonte: http://en.wikipedia.org/wiki/Cleanroom, ago/2010

Características

- Objetivo:
  - produção de sw livre de falhas pelo uso de métodos formais + verificação rigorosa ao longo do desenvolvimento + testes estatísticos
- O processo:
  - cobre todo ciclo de vida: planejamento, medição, especificação, projeto, codificação, teste, certificação
  - combina especificação e projetos formais, verificação da correção baseada em funções matemáticas, testes estatísticos para medição e certificação da confiabilidade
  - é baseado no desenvolvimento incremental

O processo para um incremento



[Pressman 97]

Estratégia (1)

- Planejamento dos incrementos:
  - definem-se as funções que cada incremento deve realizar
  - planejam-se as atividades para o desenvolvimento sala limpa de cada incremento
- Extração de requisitos de cada incremento
- Especificação da estrutura de caixas:
  - especificação funcional estruturada em **caixas**
- Projeto formal
  - refinamentos da especificação para se obter projeto preliminar e detalhado
- Verificação da correção
  - verificação rigorosa realizada em todas as fases, podendo fazer uso de métodos formais

Estratégia (2)

- Geração, Revisão e Verificação de Código
  - as caixas são convertidas para a linguagem de programação escolhida
  - o código produzido é revisado (passeio ou inspeção)
  - verificação rigorosa da correção do código é realizada
- Planejamento dos Testes
  - definição do perfil de uso do sw (distribuição de probabilidade)
  - projeto e implementação de testes de acordo com a distribuição de probabilidades identificada
- Testes estatísticos
  - testes exercitam cada entrada de acordo com sua probabilidade de ocorrência
- Certificação
  - medição da confiabilidade para cada incremento e para o sistema

Equipes

- Equipe de especificação
  - responsável pelo desenvolvimento e pela manutenção da especificação do sistema, tanto da especificação orientada ao cliente (definição de requisitos) quanto da especificação matemática para verificação
- Equipe de Desenvolvimento
  - responsável pelo desenvolvimento e verificação estática do sw
- Equipe de Certificação
  - responsável pela definição do perfil de uso do sistema, bem como pelo planejamento, projeto e implementação dos testes estatísticos

Especificação Funcional

- Organizada hierarquicamente utilizando estrutura de **caixas**
- Caixas englobam processamento e dados (como em OO)
- Caixas vão englobando mais detalhes a medida em que avançam as fases de desenvolvimento (refinamentos sucessivos):
  - **caixa preta (CP)**: especifica o comportamento do sistema (ou de um incremento) em termos de estímulos (ou eventos) e respostas
  - **caixa de estados (CE)**: encapsula dados e serviços (ou operações), da mesma forma que um objeto
  - **caixa branca (CB)**: detalha os serviços definidos na caixa de estados

Exemplo com predicados

pré:[$x \geq 0$]

[calcula y, a parte inteira da raiz quadrada de um valor inteiro de entrada, x]

pós: [x não muda e $y^2 \leq x \leq (y + 1)^2$]

⇨

pré: [$x \geq 0$]

[calcula y, a parte inteira da raiz quadrada de um valor inteiro de entrada, x]

do
[inicializa y com 0]
init: [$x \geq 0$ e $y = 0$]

[y recebe a parte inteira da raiz quadrada de x]
end

pós: [x não muda e $y^2 \leq x \leq (y + 1)^2$]

⇨

pré: [$x \geq 0$]

[inicializa y]
do
$y \leftarrow 0$;
init: $x \geq 0$ e $y = 0$

[y recebe a parte inteira da raiz quadrada de x]
loop: $y^2 \leq x$
while $y^2 \leq x$
sim: $(y+1)^2 \leq x$
$y \leftarrow y+1$;
cont: $y^2 \leq x$
end

pós: [x não muda e $y^2 \leq x \leq (y + 1)^2$]

[Linger94, Pressman97]

Limitações das técnicas de V&V

O problema

- Na Engenharia Civil:
  - É possível provar, através de cálculos matemáticos, que uma ponte suportará o volume de tráfego requerido (especificado)
- Na Engenharia de Software:
  - Dada uma especificação precisa e um programa, é possível mostrar ou provar que o programa satisfaz às propriedades especificadas?

A motivação

- Na Engenharia Civil:
  - É possível provar, através de cálculos matemáticos, que uma ponte suportará o volume de tráfego requerido (especificado)
- Na Engenharia de Software:
  - Dada uma especificação precisa e um programa, é possível mostrar ou provar que o programa satisfaz às propriedades especificadas?
    - Para algumas propriedades simples ...
    - Para alguns programas simples ...
      - Algoritmos, não sistemas

    ... é possível, a um custo elevado
    - Em geral, não é possível → indecidibilidde, intratabilidade

Problemas intratáveis

- Para todos os algoritmos que resolvem um dado problema:
  - Se existir algum que tenha complexidade polinomial → problema tratável
  - Em caso contrário → problema intratável
    - Ou seja, o problema não pode ser resolvido de forma eficiente, seja em termos de tempo ou de uso de memória
    - Nesse caso, pode-se utilizar o algoritmo para instancias reduzidas do problema
    - Ex.: problema do caixeiro viajante
      - Problema muito comum em técnicas de teste baseadas em grafos

Problema indecidível

- Alan Turing: provou que alguns problemas não podem ser resolvidos algoritmicamente
- De maneira simplificada ... [Queiroz 2012]
  - Um **problema de decisão** é uma questão sobre um sistema formal com uma resposta do tipo **sim** ou **não**:
    - Dado um número inteiro x, x é primo?
    - Dado um programa P, ele é correto?
  - Uma família de problemas de decisão é dita **indecidível** se não existe algoritmo que dê a resposta correta para todo o problema da família
    - Dados dois programas P1 e P2, eles são equivalentes?
    - Dado um programa P e uma entrada qualquer x, P pára quando roda com essa entrada? → **Problema da parada**

Relação com V&V

- Dado que quase toda propriedade importante relativa ao **comportamento** de programas incorpora o problema da parada:
  1. O programa satisfaz à propriedade?
  2. O programa pára para toda e qualquer entrada?

  Sempre se pode achar um caso em que a questão 1 tem reposta SIM, mas a 2. não
- Dado que a verificação de algumas propriedades recai em um problema intratável ...
- O que fazer, então?
  – Utilizar propriedades que sejam mais fáceis de verificar
  – Restringir a classe de programas a verificar
  – Aceitar que as técnicas são imprecisas

Imprecisão das técnicas

- Técnicas otimistas:
  - Podem aceitar programas (ou sistemas) que não satisfazem à propriedade, i.e, não detecta algumas violações → **falsos negativos**
  - Ex.: testes
- Técnicas pessimistas:
  - Podem não aceitar um programa, mesmo que este satisfaça à propriedade analisada (*análise conservativa*)
  - Indicam violações que na verdade não o são → **falsos positivos**
  - Ex.: analisadores de código
- Em alguns casos, a técnica pode oferecer a resposta "não sei" (**inconclusivo**)

Simplificação de propriedades

- Supondo que desejamos verificar uma propriedade S, mas as técnicas de análise estática disponíveis geram muitos falsos positivos (técnicas conservativas)
  - Selecionar uma propriedade S', que é uma condição suficiente, mas não necessária, para S (i.e, a validade de S' $\Rightarrow$ S mas não o contrário)
  - Dado o trecho de código C abaixo:

```
1. int j, soma;
2. int prim = 1;
3. for (j=0; j<10; j++) {
4.     if (prim)
5.         { soma = 0; prim = 0; }
6.     soma += j;
7. }
```

- S: toda variável deve ser inicializada com um valor antes de ser usada em uma expressão
- Análise conservativa: o código é válido ou não?
- Proponha S'

Preparação para a verificação

- Adquira os recursos necessários: equipamentos, ferramentas
- Adquira ou desenvolva ferramentas para armazenamento (*logging*) e rastreio (*tracking*) das falhas encontradas
- Use os resultados obtidos para uma análise visando a melhoria da qualidade do processo ou do produto
  - Classificação das falhas encontradas por fase de desenvolvimento ajuda
- Analise periodicamente a verificação: o processo precisa de ajustes? Precisa-se de novas ferramentas?
- Analise as técnicas de análise utilizadas:
  - Que tipo de imprecisão elas apresentam?
  - Elas são adequadas para o domínio de aplicação?
    - Maioria das técnicas voltadas para programas procedimentais e OO
    - O que fazer com características como: concorrência, distribuição, interfaces gráficas,etc?

Principais pontos

- Qual a diferença entre verificação e validação?
- Porquê é importante realizar V&V desde cedo?
- Qual a diferença entre técnicas estáticas e dinâmicas de verificação?
- Qual a diferença entre revisão e leitura?
- Qual a atividade de V&V que é mais utilizada?
- Quais as principais limitações das técnicas de V&V?