Fiabilidade dos Sistemas Informáticos

Apresentação e Introdução à Engenharia da Fiabilidade e aos Métodos Formais

Simão Melo de Sousa

Computer Science Department
University of Beira Interior, Portugal

Outline

Plano

Focus

Introduzir os métodos e as ferramentas que permitam construir sistemas informáticos isentos de falhas

Evolução dos Sistemas Informáticos (SIs)

Hoje em dia:

- Difusão dos Sistemas Informáticos (em sectores essenciais da sociedade) cada vez maior;
- Complexidade dos SIs cada vez maior: tamanho e evolução técnica;
- Emergência de novas tecnologias, de novos mercados, assim como de novas exigências (segurança, escalabilidade, disponibilidade, etc.).

Evolução dos Sistemas Informáticos (SIs)

Exemplos:

Informática omnipresente, sociedade digital, e-government, sistemas embebidos, sistemas de segurança, dinheiro e transacções electrónicas, sistemas críticos, etc...

Mas...

Métodos actuais de desenvolvimento de SIs desactualizados frente aos novos desafios técnicos e científicos levantados pela emergência e massificação destes SIs.

Evolução dos Sistemas Informáticos (SIs)

SIs modernos = novos desafios

a { **fiabilidade**, **correcção**, **segurança**, **robustez**, **etc...** } dos SIs como desafio central

Engenharia de Software & Análise de Sistemas

Contribuição à problemática da fiabilidade:

- Identificação de algumas noções chaves, como a de ciclo de vida dum SI
- Produção: planeamento, análise, quantificação e gestão dos custos, manutenção, etc...
- Introdução de metodologia de produção, de exploração (etc...) do SI: $\Longrightarrow$ início de rigor na produção de SIs.

Engenharia de Software & Análise de Sistemas

Quantificação de custos de produção

- Manutenção de um SI $= \frac{2}{3}$ dos custos totais;
- Conserto de um erro de especificação: esforço 20 vezes maior se detectado após produção do SI.

Inadequação do ciclo Detecção-Conserto

A situação

- Um erro na especificação pode implicar mais do que um simples conserto (vidas humanas, impacto económico, confiança dos clientes, etc...);
- Existem (muitos) SIs em que o aparecimento de erros é inaceitável

Inadequação do ciclo Detecção-Conserto

mais uma vez. . .

SIs modernos = novos desafios

$\Downarrow$

necessidade da **fiabilidade** como desafio central

Alguns Factos e Números

Há mais de 10 anos, W. Gibbs dizia:

Despite 50 years of progress, the software industry remains years – perhaps decades – short of the mature engineering discipline required to meet the needs of an information-age society.

in: Trends in Computing: Software's Chronic Crisis - Scientific American - 1994.

Provocação? Passado?

Infelizmente, não. Veja por exemplo:

Comercialização = criação de um canal para bug reports

Alguns Factos e Números

Impacto económico das falhas nos SIs

- Ariane V : 2 biliões de $
- Avaliação pelo NIST dos custos dos erros em SIs na economia dos USA
  - 1995 : 81 biliões de $
  - 1996 : 100 biliões de $
  - 2002 : 59.5 biliões de $

Alguns Factos e Números

Impacto comercial das falhas nos SIs

- Ruptura de contrato por razões de falta de garantias de fiabilidade (exemplo da IBM)
  - US Federal Aviation Agency : 8 biliões de $
  - DoD : 2 biliões de $
- Confiança dos clientes. Exemplo da Intel e do Bug do Pentium: Impacto da perda de confiança dos cliente maior do que o impacto económico do conserto.

Alguns Factos e Números

etc. . .

Soluções?

Infelizmente,

Não há soluções universais!

No entanto, soluções adaptadas e satisfatórias:

repensar e adaptar o ciclo de produção de SIs de forma a integrar a **fiabilidade** como um requisito central

$$\Downarrow$$

Common Criteria

Soluções?

Infelizmente,

Não há soluções universais!

No entanto, soluções adaptadas e satisfatórias:

integrar e utilizar métodos de fiabilização :

- Testes e Simulação
- Métodos Formais

Testes e Simulação

Princípio Geral

Para um determinado **modelo** do SI alvo, **fornecer** um conjunto de dados **representativos** e **comparar** a resposta com o resultado esperado.

Não existe teste satisfatório:

exaustividade: geralmente impossível (conjunto de valores possíveis eventualmente infinito)

representatividade: em geral problemas ocorrem quando os dados têm valores inesperados (i.e. não representativos).

Testes e Simulação

Uma ilustração:

Littlewood provou, em 1914 e à surpresa geral de todos, que a função

$$\pi(n) - li(n)$$

onde

$$li(n) = \int_0^n \frac{du}{ln(u)} \quad \text{e} \quad li(n) = \text{número de primos} \leq n$$

muda de sinal infinitamente. Surpresa? sim, porque apesar de intensos testes sobre valores chegando a 10^{10} (antes da aparição dos computadores), nenhuma mudança de sinal foi detectada.

Métodos Formais

Uma Definição

The term formal methods refers to the use of mathematical modeling, calculation and prediction in the specification, design, analysis and assurance of computer systems and software. The reason it is called formal methods rather than mathematical modeling of software is to highlight the character of the mathematics involved.

J. Rushby, Formal Methods and their Role in the Certification of Critical Systems - SRI - 1995.

Métodos Formais

Algumas Propriedades

exaustividade: raciocínio **formal** (por oposição a informal) sobre o conjunto dos valores possíveis **na sua globalidade**

rigor: bases matemáticas bem assentes

adequação: solução avaliada como satisfatória para a **garantia de fiabilidade** (por exemplo em normas ISO como os Critérios Comuns)

Critérios Comuns

Origem.

Reflexão conjunta de várias entidades normativas sobre a

$$\left\{ \begin{array}{l} - \textbf{necessidade} \\ - \textbf{definição} \end{array} \right.$$ de um **modelo/métrica** para a avaliação da **fiabilidade/segurança** dos SIs:

Norma ISO-15408 designada por Common Criteria

Critérios Comuns

Critérios Comuns = métrica para segurança dos SIs

Definição

- Métrica organizada em 7 níveis de confiança (Evaluation Assurance Level – EAL)
- Para os mais altos níveis (de EAL5 a EAL7)
  - Entidade de certificação : Entidade pública (independência da certificação)
  - Métodos Formais requeridos (rigor da avaliação)

Plano

Desenvolvimento Formal de Software

Começemos por uma pequena digressão sobre a noção de

Desenvolvimento de Software (DS).

Objectivo: mostrar (de forma sucinta) que é preciso ajustar os métodos tradicionais de DS

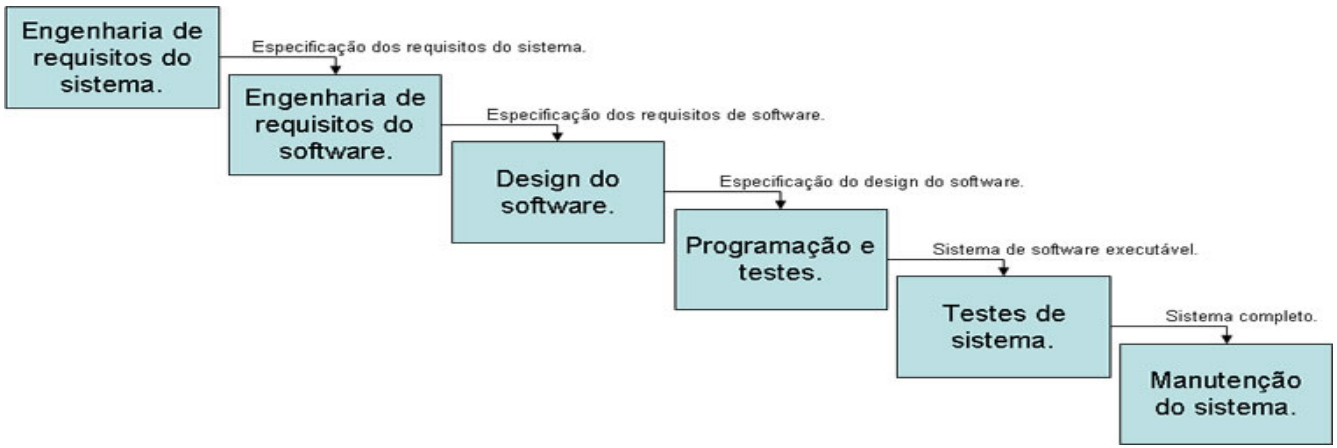
Desenvolvimento de Software

Ciclos de vida dos SIs: Os classicos

- Em V
- Em Cascata (Boehm, 1977)

Métodos Semi-Formais:

- SA, SADT, SSADM, MERISE, UML, etc...
- Análise feita, mas não há certezas relativamente aos requisitos de fiabilidade
- No entanto, contribuição positiva.



Desenvolvimento Formal de Software

Fiabilidade como requisito Central

Como já o referimos, estes ciclos, assim como os métodos referidos, não conseguem tomar convenientemente conta dos requisitos omnipresentes no desenvolvimento de SI modernos como a *fiabilidade*, a *correcção* a *robustez*. Esta apresentação advoga e introduz o

Desenvolvimento Formal de Software

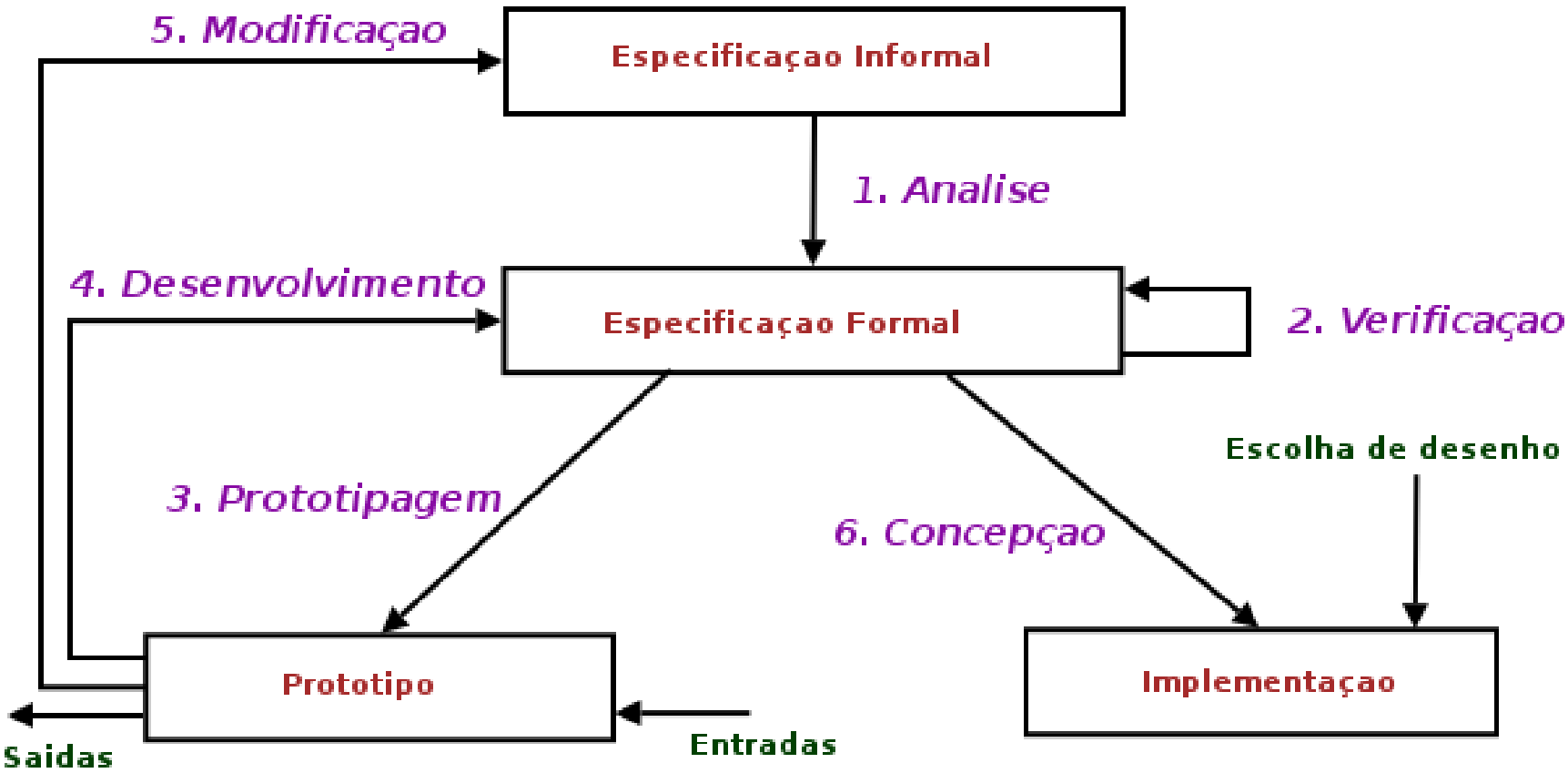
Especificação Formal e Engenharia de Software: Que benefícios?

Mais uma vez: Desenvolvimento Formal de Software. Mas como integrar estas abordagens nos mecanismos clássicos de desenvolvimento de Software?

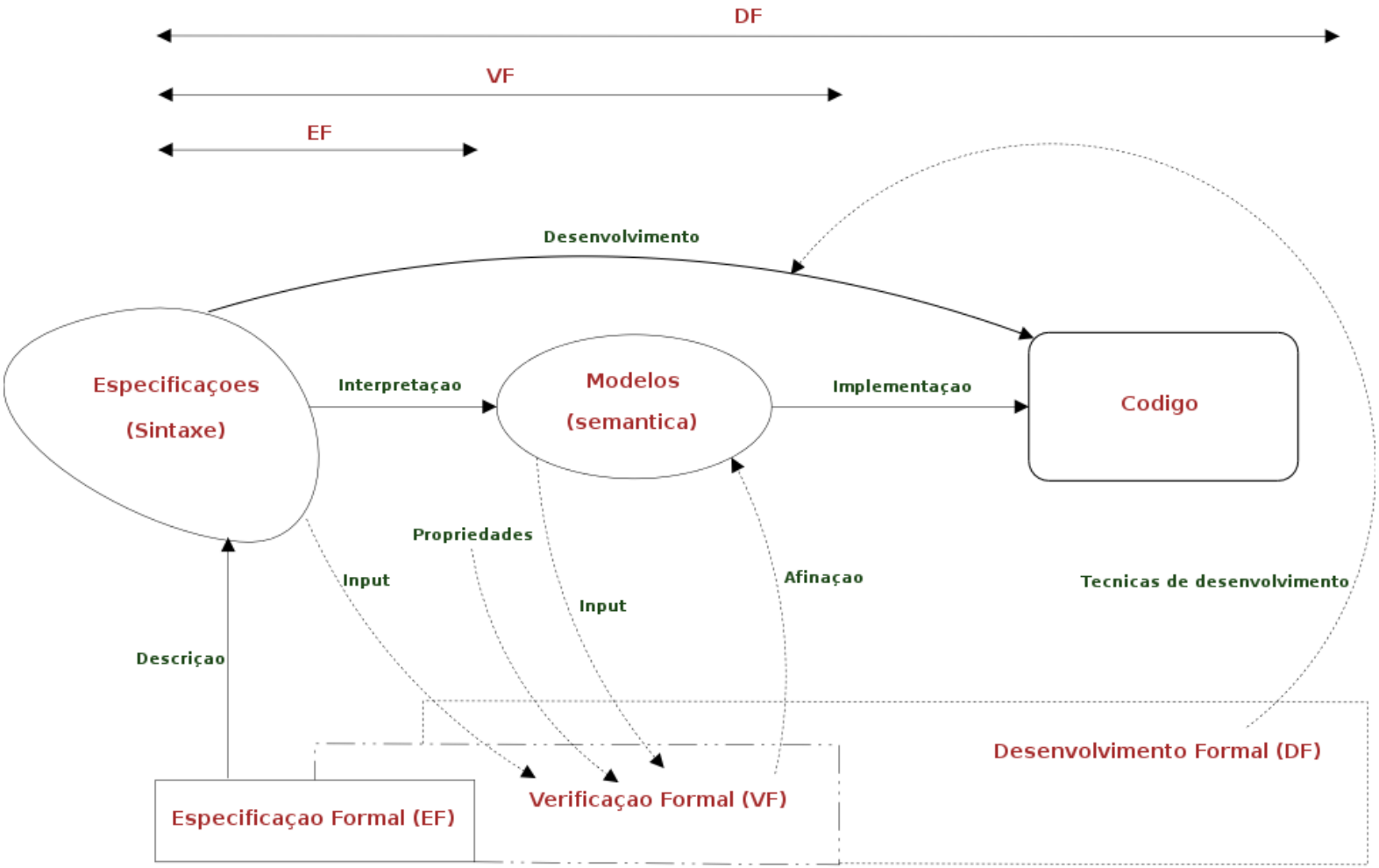
Lembrete

Modelar, raciocinar, derivar (transformação sistemática do modelos em implementação: refinamento, extracção).

Solução: repensar e adaptar os ciclos de vida do Desenvolvimento (Formal) de Software



Ciclo de Vida de Balzer



Desenvolvimento Formal

Plano

O Problema Central

Os Métodos Formais pretendem:

Garantir que um SI tenha um determinado (bom) comportamento

O Problema Central

Os Métodos Formais pretendem:

Garantir que um SI tenha um determinado (bom) comportamento

Noção central de $\left\{ \begin{array}{l} - \textbf{modelo} \\ - \textbf{especificação} \end{array} \right.$ para levar o objecto do discurso para a matemática.

O Problema Central

Os Métodos Formais pretendem:

Garantir que um SI tenha um determinado (bom) comportamento

Subdivisão do problema principal em dois sub-problemas

Garantias de comportamentos: Como garantir/verificar ao nível do modelo o comportamento desejado

Confronto "modelo vs. implementação":

1. Como obter de um modelo uma implementação usufruindo desse comportamento?
2. Como garantir que uma implementação tem o mesmo comportamento que um modelo?

Visita Guiada

O que vem a seguir

Visita dos formalismos e das ferramentas organizada em camadas de funcionalidades:

- Especificar e analisar modelos
- Especificar e demonstrar propriedades
- Especificar e derivar implementações
- Especificar e transformar especificações

Especificar e Analisar

Princípios Subjacentes

Formalismos/linguagens/ferramentas para expressar modelos/especificações.

Benefícios Imediatos

- Linguagens Formais = Rigorosos e sem Ambiguidades
  - Obriga a uma reflexão aprofundada e a uma compreensão detalhada dos mecanismos modelados (e.g. D. Syme, CISC/IBM);
  - "Esperanto" ideal para a documentação e comunicação entre as diferentes partes envolvidas na produção do SI;

Especificar e Analisar

Princípios Subjacentes

Formalismos/linguagens/ferramentas para expressar modelos/especificações.

Benefícios Imediatos

- Linguagens Formais = Rigorosos e sem Ambiguidades
- No caso da existência de uma implementação do formalismo, observação directa do comportamento sem necessidade de implementação: prototipagem
- Enquadramento formal propício ao raciocínio matemático

Construir Modelos/Especificações

Vamos aqui nos debruçar sobre o primeiro sub-problema, em particular sobre a construção de especificações

Definição Informal

- Especificação Formal: Descrição Formal do "QUÊ" dum sistema por desenvolver (por oposição ao "COMO").
- $\Longrightarrow$ resultado da fase de analise.

Construir Modelos/Especificações

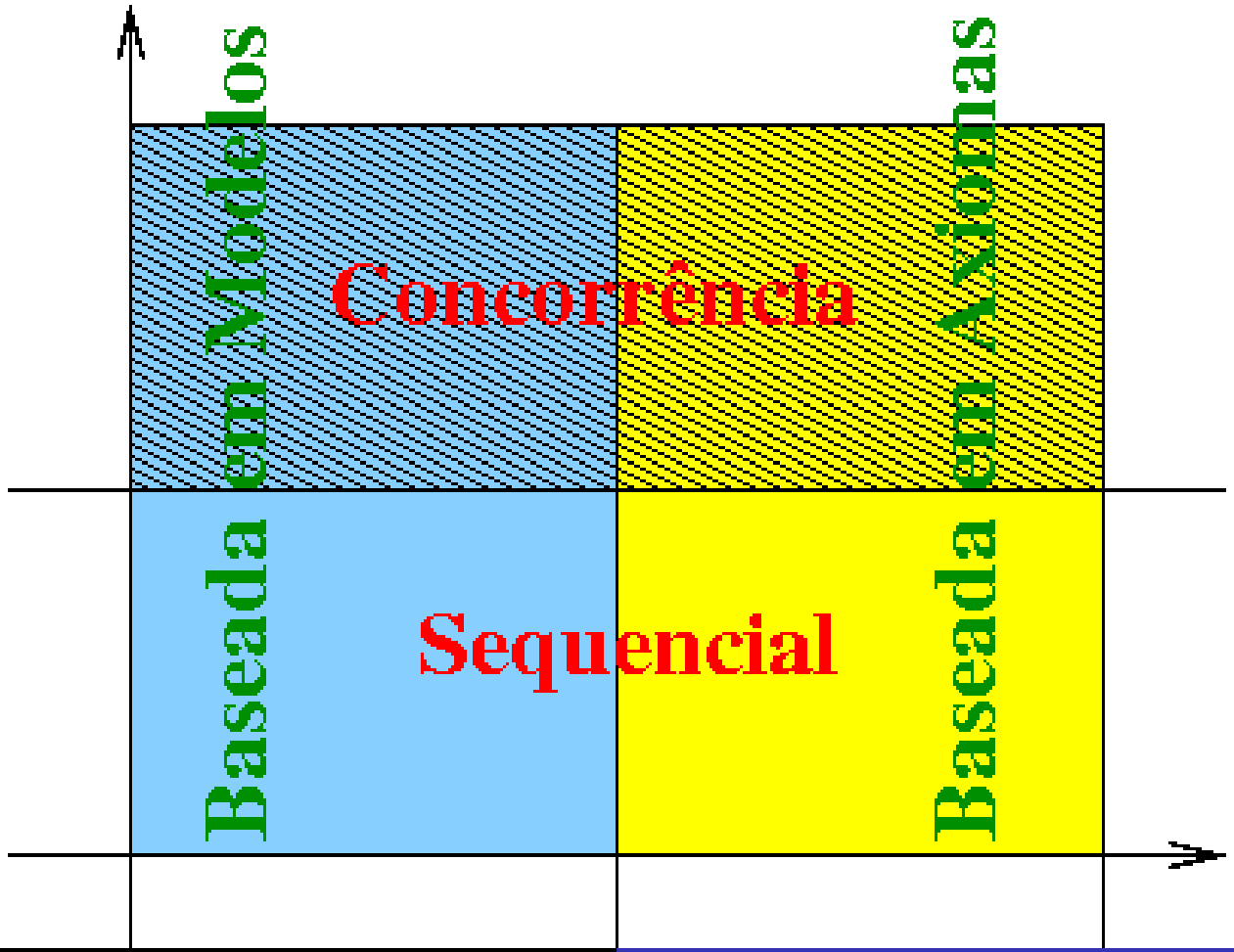
Abordagens à Especificação

Objectivo: dar uma formulação matemática (rigorosa, formal) do comportamento do sistema por modelar.
Comecemos por distinguir duas grandes famílias de especificações formais. Ênfase:

- nos dados manipulados. O comportamento do sistema alvo pode ser expresso via os dados que manipula e pela forma com que estes dados evoluem ou se relacionam: especificação algébrica, ou ainda, baseada em axiomas .
- nas operações. O comportamento pode ser expresso via as operações, mecanismos de que dispõe ou ainda as acções que é capaz de executar. Aqui o cerne da especificação é as modificações que as operações efectuam sobre um *estado* interno: especificação baseada em estados ou modelos.

Existe igualmente uma catalogação ortogonal: modelos sequenciais ou modelos concorrentes (CSP, CCS, Álgebra dos Processos, Π – *calculus*, etc...).

Construir Modelos/Especificações



Especificação Orientada a Estados/Modelos

Princípios

Determinar a noção de estado para o sistema alvo e privilegiar a descrição de que como as operações do sistema alteram o estado.

A matemática subjacente

Matemática discreta, teoria dos conjuntos, teoria das categorias, lógica

Exemplos de formalismos - sistemas

Sequencial: Z, VDM, B, Charity, Perfect Developer,
Concorrência: Redes de Petri, CCs, CSP, UNITY, B evenementiel etc...

Máquinas de Estados Abstractos

- Formalismos da família dos sistemas de transição baseado nas noções de estado e de transformações de estado,
- SI = máquina abstracta = conjunto de estados + conjunto de transições não determinísticas
- Transição = condição de eleição + transformações ao estado da máquina
- Turing-equivalente
- Implementações existentes: ASM_Gopher, Sistema B

Teoria dos Conjuntos e das Categorias

- estado = conjunto + funções + relações
- transição = invariantes + pre/post condições
- formalismo particularmente expressivo e modular
- alvo de numerosas implementações : Z, VDM, RAISE, SPECWARE, CHARITY, CAMILA, etc...

Especificação Algébricas

Princípios

- Definir o domínio do discurso (os dados sobre os quais nos irmos debruçar)
- Fornecer uma descrição axiomática do comportamento das operações que podem manipular o alvo do discurso

A matemática subjacente

Âlgebra, Indução, Lógica equacional

Exemplos de formalismos - sistemas

Sequencial:ACT-ONE, CLEAR, OBJ, CASL, SPECWARE, etc...
Concorrência: LOTOS, etc...

Especificação Híbridas: Princípios

Misturar as especificações por modelos com especificação algébrica.
Exemplos: RAISE (VDM+CSP), Extended LOTOS, ZCCS, etc...

Especificação Algébrica

- baseado nas álgebras "multi-sorted"
- tipos de dados do SI alvo = conjuntos
- funções do SI alvo =
  - símbolos de funções introduzidos por assinaturas
  - propriedades dos símbolos estabelecidos por axiomas
- implementações : OBJ, CLEAR, ACT-ONE

Outras Famílias de Especificação formal

Ortogonalmente a estas duas famílias de metodologias e ferramentas existem outras categorias cada vez mais populares e que tendem em complementar ou em integrar-se no universo existente da especificação formal.

Modelização por Autómatos

- Outro formalismo da família dos sistemas de transição
- transição = evento
- formalismo particularmente adaptado para a modelação do comportamento do SI alvo: hardware, SI concorrente, SI reactivo, SI comunicante, protocolo etc...

Modelização Declarativa

Uma classe importante de formalismos:

Formalismos lógicos

baseado na noção de predicados de primeira ordem

- tipos de dados do SI alvo = estrutura de dados (simples) da linguagem alvo
- função = descrição via propriedades comportamentais
- implementações: Prolog,λ-Prolog, etc...
- animação e execuções

Modelização Declarativa

Formalismos lógico-funcionais

baseado no λ-cálculo (teoria das funções de ordem superior)

- tipos de dados do SI alvo = tipos de dados (arbitrariamente complexos) da linguagem alvo (tipos indutivos, dependentes, etc..)
- função = funções de ordem superior
- implementações:

  linguagens: Scheme, SML, Haskell, OCaml,
  sistemas de prova: ACL2, DECLARE, COQ, PVS, HOL, ISABELLE etc...
- Possibilidade de animação e de execução

Modelização Declarativa

Sistemas de reescrita

baseada na noção de redução

- tipos de dados do SI alvo = estrutura de dados da linguagem alvo
- função = descrição computacional via equações
- implementações: ELAN, SPIKE etc...
- animação e execução

Modelização Declarativa

Semântca das linguagens de programação

família de formalismos adaptados para a expressão matemática de linguagens de programação.

- ambiente para a expressão de semânticas (expressão dos símbolos e do comportamento semântico destes)
- animação e execução
- implementações: Action-Semantics, LETOS, ASF+SDF, Centaur, RML, etc...

Especificar e Demonstrar

Garantir um comportamento

- animar ou executar é insuficiente
- necessidade de demonstrar: **verificação formal**

Especificar e Demonstrar

3 tipos de suporte a verificação formal (J. Rushby)

1. Ferramentas que só fornecem um enquadramento formal, não disponibilizam nenhum suporte adicional
2. Ferramentas que fornecem um sistema formal para a expressão rigorosa de raciocínios
3. Ferramentas fornecendo um sistema formal e um suporte computacional para a expressão de demonstrações

demonstração "manual", em língua natural que é válida se convencer a comunidade

Focus

$\Longrightarrow$ **Panorama dos sistemas de nível 3**

Especificar e Demonstrar

3 tipos de suporte a verificação formal (J. Rushby)

1. Ferramentas que só fornecem um enquadramento formal, não disponibilizam nenhum suporte adicional
2. Ferramentas que fornecem um sistema formal para a expressão rigorosa de raciocínios
3. Ferramentas fornecendo um sistema formal e um suporte computacional para a expressão de demonstrações

demonstração "manual" e uso exclusivo de linguagem formal para a expressão da demonstração

Focus

$\Longrightarrow$ **Panorama dos sistemas de nível 3**

Especificar e Demonstrar

3 tipos de suporte a verificação formal (J. Rushby)

1. Ferramentas que só fornecem um enquadramento formal, não disponibilizam nenhum suporte adicional
2. Ferramentas que fornecem um sistema formal para a expressão rigorosa de raciocínios
3. Ferramentas fornecendo um sistema formal e um suporte computacional para a expressão de demonstrações

$\Longrightarrow$ **Mais rigoroso** (e.g. Lamport, etc...)
i.e. no mesmo enquadramento:
expressão formal de um modelo e da sua demonstração, **verificação mecânica** de demonstrações de propriedades do modelo.

Focus

$\Longrightarrow$ **Panorama dos sistemas de nível 3**

Especificar e Demonstrar

3 tipos de suporte a verificação formal (J. Rushby)

1. Ferramentas que só fornecem um enquadramento formal, não disponibilizam nenhum suporte adicional
2. Ferramentas que fornecem um sistema formal para a expressão rigorosa de raciocínios
3. Ferramentas fornecendo um sistema formal e um suporte computacional para a expressão de demonstrações

$\Longrightarrow$ **Mais rigoroso** (e.g. Lamport, etc...)
i.e. no mesmo enquadramento:
expressão formal de um modelo e da sua demonstração, **verificação mecânica** de demonstrações de propriedades do modelo.

Focus

$\Longrightarrow$ **Panorama dos sistemas de nível 3**

Sistemas de Prova

Conceito

sistema formal = sistema dedutivo

Isto é: permitam a expressão de modelos e a demonstração de propriedades

a variedade dos formalismos desta família explica-se pelo balanço entre dois factores antagónicos por considerar:

expressividade lógica

versus

automatização da dedução

Demonstradores de Teoremas

. . . ou Automated Theorem Provers

Privilegia: automatização da dedução

Vantagens:

- Após parametrização do motor de dedução, a demonstração é automática
- Possibilidade de raciocínio sobre conjuntos infinitos (e.g. por indução)

Desvantagem: Nem tudo o que pretendemos expressar pode ser expresso

Sistemas: Prolog, SPIKE ELAN, ACL2, etc...

Sistemas de Prova Assistida

. . . ou Proof Assistants

Privilegia: expressividade lógica (em geral lógica de ordem superior)

Vantagens: Muitas propriedades/demonstrações precisam dessa expressividade Raciocínio próximo do raciocínio matemático standard

Desvantagem: lógica indecidível $\Longrightarrow$ intervenção do utilizador necessária

Sistemas: Coq, PVS, DECLARE, HOL, ISABELLE

Verificação de Modelos

. . . ou Model Checking

- Abordagem alternativa aos sistemas de prova com muitas aplicações bem sucedidas
- Técnica de verificação propriedades de SI modelados como sistemas (concorrentes) de estados finitos (e.g. autómatos)
- Propriedades esperadas expressas por fórmulas de uma determinada lógica temporal
- Verificação de propriedade traduzida numa determinada exploração do sistema de transição

Verificação de Modelos

. . . ou Model Checking

Vantagens:

- Demonstração Automática
- Exibição de um contra-exemplo em caso de erro

Desvantagens:

- Problema da explosão do espaço de estados (crescimento exponencial do grafo de estado em relação ao tamanho do sistema modelado)
- Nõ sabe lidar com conjuntos infinitos

Sistemas: SMV, Murphy, SPIN, Kronos, Design/CPN, etc. . .

Anotações de programas e Lógica de Hoare

Consiste em

- Anotações lógicas do programa fonte (pre/post condições, invariantes)
- construção automática de um modelo w.r.t. semântica axiomática da linguagem alvo
- suporte para a demonstração de propriedades (delegação da demonstração a verificadores de modelos, sistemas de prova etc...)

Anotações de programas e Lógica de Hoare

. . . ou design by contracts

Vantagens: Democratiza a verificação formal de SIs ao colocar o papel da verificação na mão de quem desenha/desenvolve o SI alvo

Desvantagens: Desenhar anotações é uma arte que nem todos dominam (Democratização elitista :))

Sistemas: fortemente ligados ás linguagens alvo.

- **Java:** JML, LOOP tool, ESC/JAVA, Bandera, Jive, Jack etc...
- **C♯:** Spec♯

Especificar e Derivar

Segundo Subproblema central dos Métodos Formais

Obter uma implementação de uma especificação aprovada

Soluções?

- A própria linguagem de especificação é uma linguagem de programação $\Longrightarrow$ Q.E.D.
- Refinamento
- Extracção

Especificar e Derivar

Refinamento

Técnica que permite a síntese "passo a passo" de implementações a partir de especificações.

- 1 passo = uma escolha de implementação (e.g. um conjunto por uma lista ligada, uma ordenação pela ordenação "quicksort", etc...)
- cada passo tem de ser formalmente justificado (e.g. não altera nem o comportamento e nem as propriedades da especificação fonte)

Sistemas: Z, VDM, B, SPECWARE

Especificar e Derivar

Extracção

Lógica inerente aos sistemas de prova assistida = lógica intuicionista $\Longrightarrow$ Isomorfismo de Curry-Howard

Isomorfismo de Curry-Howard

λ-calculo	lógica	programação
tipo	enunciado lógico	especificação
termo	demonstração	programa

Especificar e Derivar

Extracção

Conceito: Seja T o teorema

$$\forall (x, y) \in \mathbb{N}^2 . \exists (q, r) \in \mathbb{N}^2 . (y = (q \times x + r) \wedge r < x)$$

Na lógica intuicionista na na interpretação de Curry-Howard, uma demonstração de T é uma função que calcula o par (q, r) que testemunha da validade de $T \Longrightarrow$ Q.E.D.

Sistemas: COQ

Especificar e Transformar

Abstrair e Refinar

A situação: Necessidade de mecanismos de transformação de modelos (e.g. definir uma variação, uma simplificação, extensão etc...) para expressar provas complexas, modulares em especificações de tamanho grande etc...

Exemplo: Abstracção de especificações para focar a especificação sobre um determinado comportamento alvo da demonstração

Sistemas: ASF+SDF, JaKarTa

Especificar e Transformar

Potencial das Combinações

Não existe nenhuma ferramenta que permite resolver de forma global e satisfatória o problema central

$\Downarrow$

pertinência das plataformas de especificação permitindo a colaboração de vários métodos formais: JaKarTa

Uma Contribuição: JaKarTa

Objectivo: permitir um tratamento satisfatório e global.

- uma arquitectura centrada numa plataforma de especificação própria;
- mecanismos de comunicação com diferentes métodos formais (sistemas de prova, demonstradores de teoremas, linguagens de programação);
- mecanismos de administração e automação de demonstrações (em COQ);
- mecanismos de transformações de especificação;
- mecanismos de prova de correcção das transformações realizadas.

Plano

Algumas aplicações bem sucedidas

Muitos exemplos, dos quais:

- hardware, protocolos, redes, SOs, sistemas críticos, plataformas de execução, software, etc...
- Áreas de aplicações:
  - Aviação (NASA, FAA, ARIANE, etc...),
  - Caminhos de ferro (Metro, etc...)
  - Sistemas nucleares
  - Sistemas médicos
  - Sistemas embebidos

Algumas aplicações bem sucedidas

Uma contribuição

Verificação formal da plata-forma JavaCard (em conjunto com G. Barthe, G. Dufay):
Definição pioneira de uma metodologia para automaticamente obter (1) da Especificação do ambiente de execução de JavaCard e (2) da Demonstração de (citação de Milner)

Well-typed (JavaCard) Programs cannot go wrong

uma implementação da plataforma e de um módulo de verificação estática de programas (BCV) demonstrados correctos

Algumas aplicações bem sucedidas

Caso (antigo) da verificação formal do projecto CISC da IBM (Huxley Park- UK & Oxford): Actualização de um SI.

- de 800 000 linhas de código, 268 000 reescritas, 37 000 via Z (só especificação, sem provas)
- resultados:
  - custos de desenvolvimento -9%
  - 2.5 vezes menos de bugs e os bugs detectados foram menos importantes (logo implicaram menos custos de reparação).

Considerações Finais e Perspectivas

Tendência

Emergência de um mercado "aliciante" (i.e. $$!) ainda por liderar (por mais quanto tempo?):
Intel, Microsoft, IBM, NASA, Esterel Technology, Prover Technology, Compaq/HP, SIBS (pt), Sidereus (pt), etc...

Considerações Finais e Perspectivas

Tendência

- Importância estratégica dos Common Criteria cada vez maior
- Perspectivas: 2 eixos.
  - Desenvolvimento conceptual e técnico de novas soluções (formalismos e ferramentas)
  - A tendência actual: Utilização cada vez mais intensiva dos métodos formais (caso FORMAVIE, Trusted Logic etc. . .)
- { - competência valiosa e procurada
  - o informático, além de saber produzir SIs, deverá igualmente saber validá-los, produzir garantias

Considerações Finais e Perspectivas

Tendência

- Importância estratégica dos Common Criteria cada vez maior
- Perspectivas: 2 eixos.
  - Desenvolvimento conceptual e técnico de novas soluções (formalismos e ferramentas)
  - A tendência actual: Utilização cada vez mais intensiva dos métodos formais (caso FORMAVIE, Trusted Logic etc. . .)
- { - competência valiosa e procurada
  - o informático, além de saber produzir SIs, deverá igualmente saber validá-los, produzir garantias

Considerações Finais e Perspectivas

Tendência

- Importância estratégica dos Common Criteria cada vez maior
- Perspectivas: 2 eixos.
  - Desenvolvimento conceptual e técnico de novas soluções (formalismos e ferramentas)
  - A tendência actual: Utilização cada vez mais intensiva dos métodos formais (caso FORMAVIE, Trusted Logic etc. . .)
- { - competência valiosa e procurada
  - o informático, além de saber produzir SIs, deverá igualmente saber validá-los, produzir garantias