CISTER - Research Center in
Real-Time & Embedded Computing Systems

Verificação Formal de Software

CISTER Summer Internship, ISEP, 04-09-2017

David Pereira

dmrpe@isep.ipp.pt

CISTER – ISEP, IPP & INESC TEC

Erros em Software

Software está presente em “todos os cantos” do nosso quotidiano…

… e, diariamente, somos confrontados com falhas que afetam as nossas vidas (em certos casos, colocando as mesmas em risco!)

Há erros de software que ficam muito caros...

NASA's Mars Climate Orbiter

Perdas : 125,000,000$

- Foi lançada em 1998 e acabou por ser perdida no espaço;
- Erro na conversão entre sistemas métricos diferentes

Ariane 5

Perdas : 8,000,000,000€ (foguete) + 500,000,000€ (payload)

- 36 segundos após o lançamento, a equipa auto-destruíu o foguete;
- Reutilização de software do Ariane 4 que levou a um overflow que bloqueou o computador principal e o seu backup (ambos executavam o mesmo software)

Knight's $440 Million Error

Perdas : 440,000,000$ em 30 minutos

- Entre as 9:30AM e as 10AM, a aplicação procedeu a negociações de ações de forma errática (aproximadamente em 150 ações;
- O comportamento errático consistiu em comprar caro e vender barato!

Há erros de software que ficam muito caros...

Pentium FDIV bug

Perdas : 475,000,000$

- Erro do processador Pentium em cálculos com vírgula flutuante, mas considerado muito raro de acontecer
- A IBM mais tarde mostrou que o erro acontecia a cada 24 dias, e os utilizadores responderam em massa

The Mariner 1 Spacecraft

Perdas : 18,000,000$ em 1962

- Após a descolagem foi de imediato auto-destruído por ter assumido uma trangetória que ameaçava uma queda em terra;
- A falta de um "-" no código originou a geração de instruções de direção errados

Há erros de software que são catastróficos!!!

"WW3"

Perdas : quase, quase, milhões de vidas!!!

- Em 1980, o sistema NORAD reportou que os USA estavam a sofer um ataque de mísseis; na realidade, um erro no hardware não foi capturado pelo software.
- Em 1983, um satélite Soviético indicou a existência de mísseis por parte dos USA; o comandante decidiu não reagir (suspeita-se que a causa foi por efeitos Solares)

Therac-25

Perdas : 3 + 21 pessoas perderam a vida

- O sistema baseado em radiação Therac-25 administrou em vários pacientes, entre 1985 e 1987, doses de radiação 100 vezes superiores ao indicado
- Algo similar aconteceu na cidade do Panamá, onde 21 pessoas perderam a vida pois os sistemas administravam dados diferentes consoante o tipo de dados inseridos.

Há erros de software que são catastróficos!!!

Misseis Patriot

Perdas : 28 soldados mortos, 100+ feridos

- Uma bateria de mísseis antiaéreos Patriot, instalada em Dharan, Arábia Saudia, falou a interseção de um míssil Scud (primeira guerra do Iraque, em 1991)
- Resultado de um erro de arredondamento em aritmética de 24 bits, usado para calcular o tempo para a interceção do míssil inimigo.

Conduta de Gás Soviética

Perdas : ???

- Maior explosão não-nuclear observada do espaço;
- Resultado do uso de software danoso, escrito de forma propositada (a.k.a. CIA) por uma empresa Europeia, adquirido pela União Soviética durante a Guerra Fria.

Fontes:

https://raygun.com/blog/10-costly-software-errors-history/

http://royal.pingdom.com/2009/03/19/10-historical-software-bugs-with-extreme-consequences/

Qual a razão para tantos erros???

Erros de programação

O código é escrito por humanos e, portanto, naturalmente sujeito a falhas!!!

Razões não-técnicas

Sistemas postos em funcionamento sistemas, mas não testados o suficiente:

- Prazos de entrega
- Redução de custos
- Pressões políticas/estratégicas

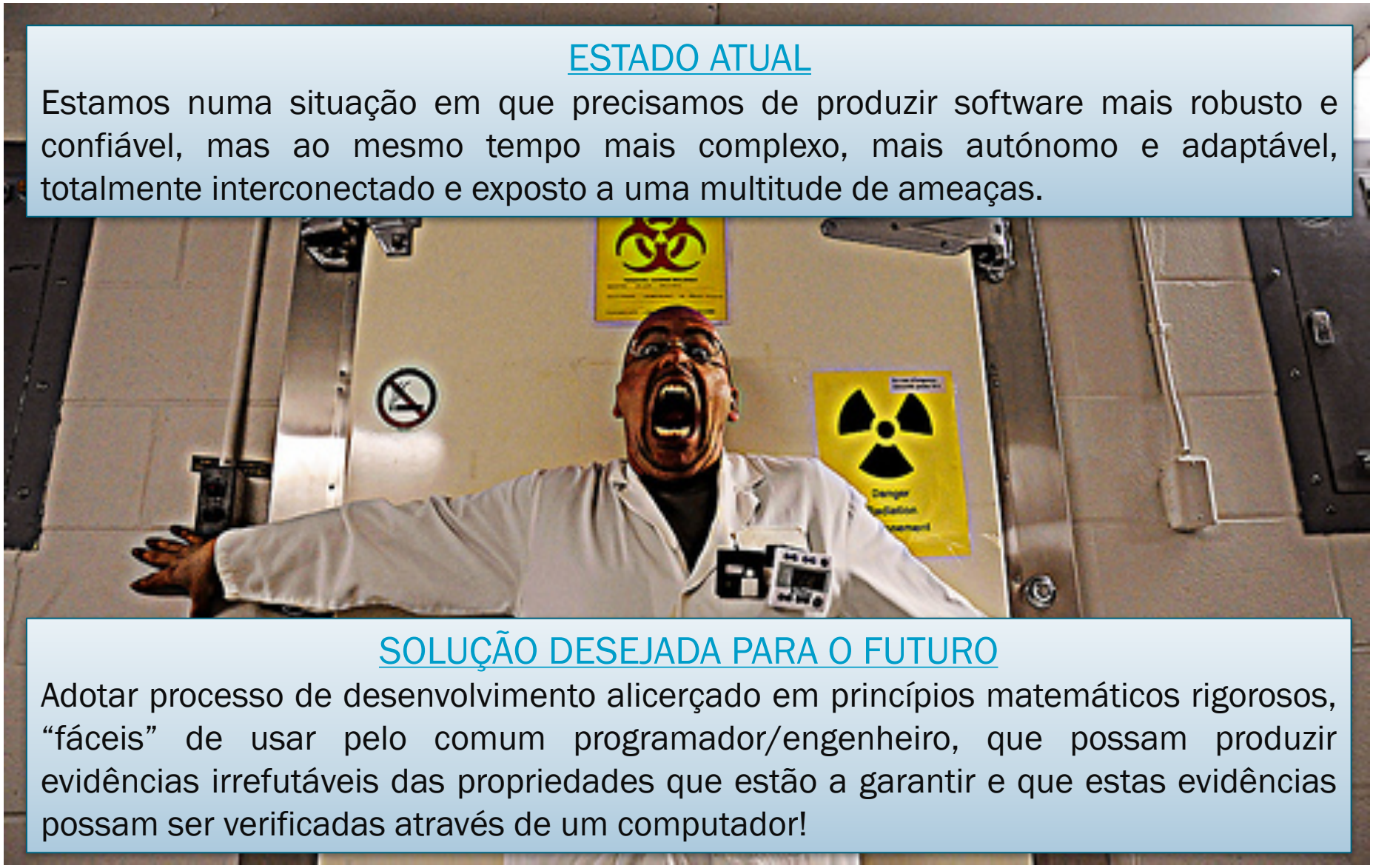
Abordagem Técnicas

Mecanismos atuais baseados em teste e simulação:

- Testes tentam encontrar erros, e não a ausência dos mesmos
- Propriedades não previstas num modelo, nunca serão simuladas;

ESTADO ATUAL

Estamos numa situação em que precisamos de produzir software mais robusto e confiável, mas ao mesmo tempo mais complexo, mais autónomo e adaptável, totalmente interconectado e exposto a uma multitude de ameaças.



SOLUÇÃO DESEJADA PARA O FUTURO

Adotar processo de desenvolvimento alicerçado em princípios matemáticos rigorosos, "fáceis" de usar pelo comum programador/engenheiro, que possam produzir evidências irrefutáveis das propriedades que estão a garantir e que estas evidências possam ser verificadas através de um computador!

Verificação Formal de Software

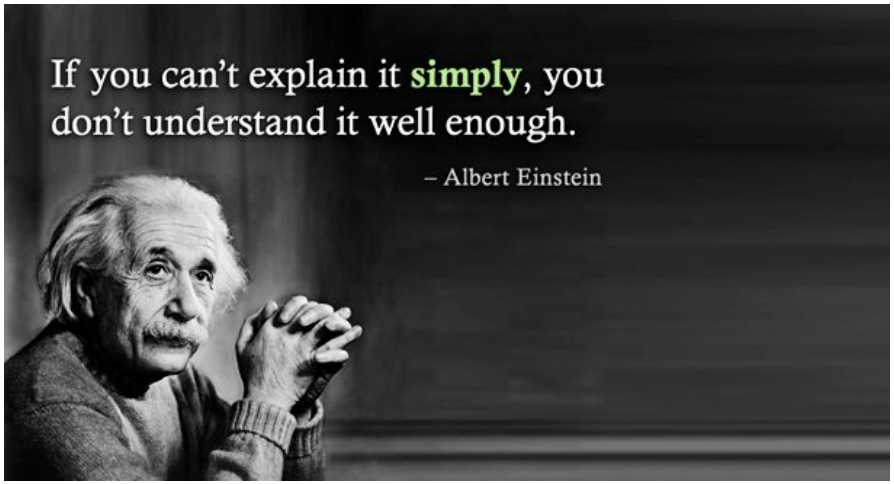
OBJETIVO

Utilizar linguagens formais para especificar um sistema e mostrar que o sistema está correto, no sentido de satisfazer as especificações.

INOVAÇÃO

Testes e simulações substituídas por **demonstrações matemáticas!!!**

- Servem de evidência e também de documentação para o comportamento computacional em causa;
- São construídas num sistema formal com sintaxe, semântica, e sistemas de dedução íntegros, cujas demonstrações são verificáveis (por humanos e computadores)



Verificação Formal

É o processo de demonstrar ou falsificar propriedades usando métodos formais (i.e., linguagens, algoritmos, ou ferramentas matemáticas).

Especificação Formal

Especificações escritas numa linguagem formal onde as asserções são bem-formadas, de acordo com as regras sintaticas dessa linguagem. São usadas para descrever o que o **software deve fazer**. São geralmente linguagens com base axiomática, ou algébrica, ou temporal/temporizada, ou concorrentes.

Demonstração

"Objeto" matemático que garante que nenhum teste/simulação pode violar uma determinada propriedade, logo elimina a necessidade de se proceder a mais testes/simulações. Necessitam de métodos de inferência/dedução, pois cada "passo" de uma demonstração tem que corresponder a uma regra bem definida, não ambígua e consistente.

Limite Fundamental

Os métodos formais não garantem a validade das especificações (responsabilidade do programador/designer/engenheiro):

ABORDAGENS

Existem dois tipos de abordagens à Verificação Formal de Software:

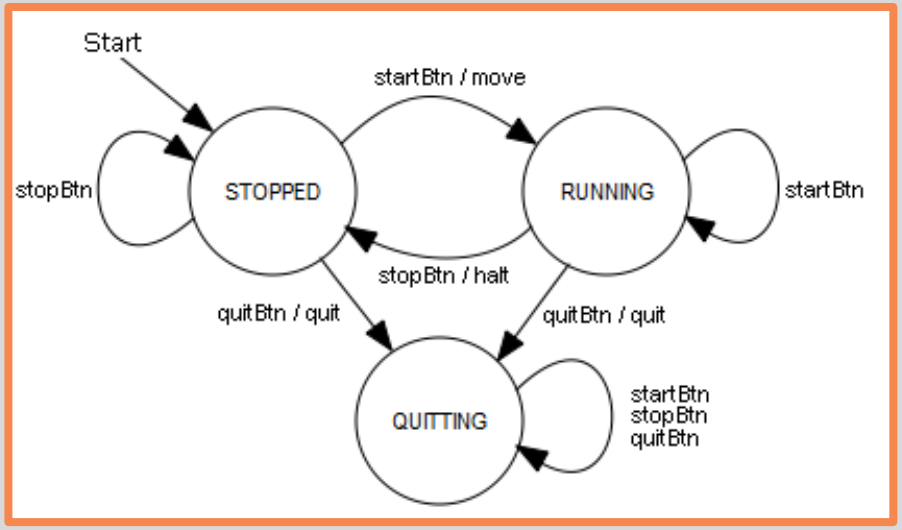
Estática: o processo de verificação ocorre logo após o processo de escrita do software, ou em simultâneo com o mesmo. Uma vez feita a verificação, o software é considerado correto e não são esperados quaisquer erros durante o tempo que se mantenha em operação;

Dinâmica: embora a especificação formal do sistema seja feita antes ou durante a escrita do software, a sua verificação ocorre em simultâneo com a operação do sistema. Tal permite lidar com dados reais, resultantes e controlar propriedades não funcionais mais facilmente (e.g., tempo de execução, consumo de energia, etc.)

Model Checking

- Propriedades especificadas numa lógica temporal/temporizada;
- Produzido modelo (abstrato) do sistema alvo;
- Pesquisa no modelo, guiada pela especificação;
- Reporta contraexemplos;

Especificação Exemplo
Informalmente: *"Existe no futuro um cenário onde a aplicação está a executar, mas que num próximo estado termina"*.
Formalmente, em Linear Temporal Logic: **EF**(*RUNNING* **->** **X**(*QUITTING*))



Demonstração "Mecânica" de Teoremas

- Construir a demonstração é difícil, mas é computacionalmente eficiente e decidivel verificar a sua boa construção;
- Demonstração automática:
  - Baseada em SAT e SMT (Satisfiability Modulo Theory – SAT + teorias)
  - SAT é decidivel mas exponencial no pior caso;
  - teorias SMT podem não ser eficientes ou expressivas para poderem garantir decidibilidade (ou seja, automação)
- Demonstração interativa/assistida:
  - Sistemas formais extremamente expressivos, com validação de termos automática (proof checking / type checking);
  - Táticas de demonstração ala Dedução Natural da Lógica de Primeira Ordem
  - Uma asserção " **P : Q** " pode ser vista de duas formas:
    - Um programa "**P**" retorna um valor do tipo "**Q**", ou;
    - "**P**" é uma demonstração de "**Q**"
    - Isomorfismo de Curry-Howard => extração de programas a partir de formalizações!!!!

Runtime Verification

- Abordagem dinâmica proeminente
- as propriedades que necessitam de ser verificadas são específicadas numa linguagem formal (normalmente numa lógica temporal/temporizada)
- Propriedades são transformadas em máquinas de estados (monitores) que reconhecem a “linguagem aceite” por essa propriedade propriedade;
- Sequências de eventos são lidas pelos monitores que decidem se estas satisfazem ou não as propriedades;
- Alerta o sistema de propriedades violadas ou ativa mecanismos de recuperação de falhas
- Aumenta a carga computacional do sistema alvo (desafio para sistemas embebidos).

Exemplos de Sucesso

- CompCert: compilador verificado (código assembly gerado tem a mesma semântica do que o código C suportado);
- seL4: microkernel formalmente verificado;
- Teorema das 4 cores (teoria de grafos)
- Teorema de Feit-Thompson
- Terno Pitagórico Booleano:
  - *"existe uma forma de separar todos os números naturais (1, 2, 3, 4, ...) em dois conjuntos, de forma que nenhum dos dois possua ternos pitagóricos?"*
  - demonstração com o tamanho de 200-terabytes, construída de forma completamente automática

Exemplo Hands-on...