Communications Security Establishment
Centre de la sécurité des télécommunications

Is Software Assurance an Oxymoron? Is Mathematics a Resolution?

Dan Craigen

Cryptographic Security

Architecture & Engineering

Presentation Outline

1. Motivation
2. SDLC
3. Definitions
4. FM Examples
5. Value propositions
6. Successes
7. Standards
8. Myths
9. Conclusions

Motivation

2005	Hudson Bay Co. (CA)	Problems with inventory system contribute to $33.3M loss
2004-5	Inland Revenue (UK)	Software errors contribute to $3.45B tax credit overpayment
1997	IRS (US)	Tax modernization effort cancelled after spending $4B
1996	Arianespace (FR)	Software spec/design error causes $350M Ariane 5 rocket to explode
1994	FAA (US)	Advanced Automation System cancelled after $2.6B [$700+/SLOC]

Charette

Motivation

Software Hall of Shame – partial list

Robert Charette
IEEE Spectrum September 2005

Illustrative Risks to the Public in the Use of Computer Systems and Related Technology

Peter Neumann
SRI International
www.csl.sri.com/users/neumann/illustrative.html

Motivation

- 55% of systems cost more than expected
- 68% were late
- 88% underwent significant redesign *IBM, 1994*

"The average company spends about 4-5% of revenue on information technology" *Charette*

… IT is now one of the largest corporate expenses outside of employee costs

Software developers spend approximately *80% of development costs* on identifying/correcting defects *NIST 2002*

Yet, few products other than software are shipped with such high error rates

Motivation

US economic cost of $59.5B annually

- 0.6% of GDP
- Half the cost borne by users

NIST 2002

Estimates that $22.2B in savings by improved testing ***earlier in life cycle***

"Increasing complexity of software, along with decreasing average product life expectancy, has increased the economic costs of errors."

NIST 2002

48% of requirements failures are due to misunderstandings or changes in the environment, not the system

Hooks and Farry, Customer Centered Products

Motivation

Software error-ridden in part because of growing *complexity*

Windows XP: 40M SLOC
Linux (some versions): 200M SLOC
Cell phones: 2M SLOC to 20M by 2010
General Motors: 100M SLOC/car by 2010

"I have always wished that my computer would be as easy to use as my telephone."

"My wish has come true. I no longer know how to use my telephone."

Bjarne Stroustrup

(Quoted by Daniel Jackson)

80% of the value of most systems is delivered by 20% of the features, and up to two-thirds of the features of most systems are rarely, if ever, used.

http://www.poppendieck.com/overview.htm#High_Productivity

Suggests that the best way to write reliable code faster is to write less code!

Motivation

"The cost of reworking errors in programs becomes higher the later they are reworked in the process, so every attempt should be made to find and fix errors as early in the process as possible."

"Rework done [earlier in the lifecycle] is 10 to 100 times less expensive than if it is done [later]."

With Fagan Inspections, "the measured increase in coding productivity of 23% is considered to validly accrue …"

Fagan 1976, 1999

Motivation

20-30 errors/1KSLOC in most software applications

Sustainable Computing Consortium

Formal design/code inspections average 65% in defect-removal efficiency. Most forms of ***testing less than 30% efficient***.

Caper-Jones

We find that if quality is integrated up-front, it actually costs less money

Payne

Motivation

"The demand for software has grown far faster than our ability to produce it. Furthermore, the Nation needs software that is far more usable, reliable, and powerful than what is being produced today."

We have become dangerously dependent on large software systems whose behaviour is not well understood and which often fail in unpredicted ways.

US PITAC, February 1999

Motivation

Software that is charged with

- protecting human life is safety-critical
- an essential task is mission critical
- protecting confidential information is security-critical

Larry Paulson

For ultra-criticality, both testing and software fault tolerance are inadequate

Butler & Finelli, 1993

10^{-9} failures/hr: Testing would take 10^{9} hours and error correction might seed new errors

Littlewood & Strigini, 1993

Motivation

State-wide Automated child Welfare Information System (SACWIS)

Florida

- Started 1990, estimated 8 years @ $32M
- By 2002, spent $170M and estimated at $230M

Minnesota

- Started in 1999, essentially the same system
- Completed in 2001 @ $1.1M

- Productivity difference of 200:1
- Standardized infrastructure, minimized requirements, 8 capable people

Jim Johnson, Chair Standish Group, 2002

Software Development Lifecycle

- Concern for “reliable software development” must start as early as possible in the SDLC
- SDLC as project “risk management”
- Industry best practice targeting higher assurance of systems and business value

Software Development Lifecycle

Scrum

Feature Driven Development

XP

UML

Dynamic Systems Development Methods

RUP

Agile, iterative, incremental, adaptive, empirical, produce business value, mitigate project risk

SCRUM

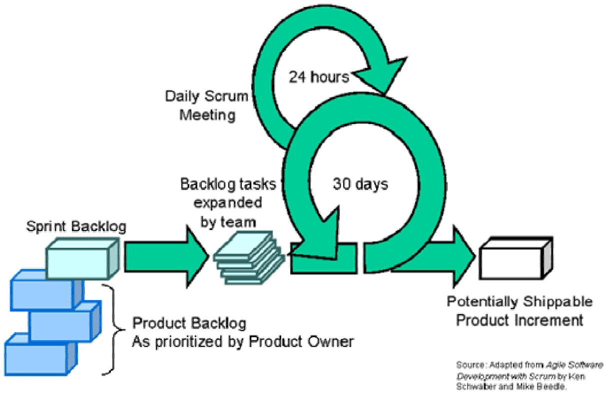


Diagram from www.methodsandtools.com

- BV earlier in cycle by focusing on the 20% of features prioritize features
- 24 hours: Done, Plan, Impediments
- Iterative release: Some requirements, analysis, design, development and testing
- Product owner, Scrum master, Project team (5-10)
- Provide business value every 30 days

SCRUM

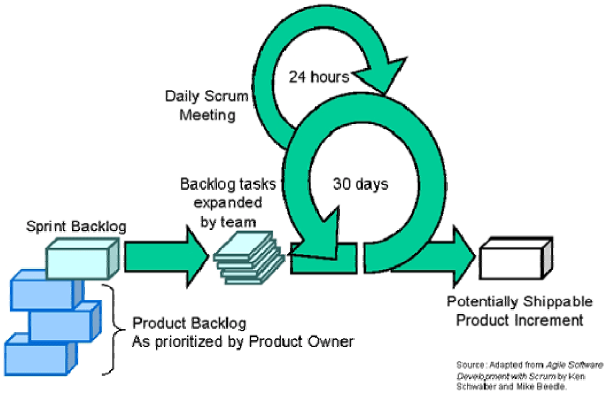


Diagram from www.methodsandtools.com

- Daily meeting: 15 minutes & standing; scrum master and project team are the only speakers
- Few artefacts: Product backlog, Sprint backlog, Burndown charts
- Iterative development accelerates drive to profitability

Feature Driven Development

Providing business value - $s

PRAISED:

- Productivity gains
- Reduced cost
- Avoided cost
- Increased revenue
- Service level improvements
- Enhanced quality
- Differentiation

- Client valued
- Inclusive methodology throughout
- Agile: features in 1-10 days
- Release meetings/cycles (2 weeks)

- Frequent, tangible working results
- Highly iterative
- Core set of industry best practice
- Quality built-in

Feature Driven Development

- Domain Object Modeling
- Developing by feature
- Individual class ownership
- Feature teams
- Inspections (design, code)
- Regular builds
- Configuration management
- High visibility

A feature is a small, client valued function expressed in a specific form

<action> the <result> <by|for|of|to> an <object>

Five processes:

- Develop an overall model
- Build features list
- Planning
- Design by feature
- Build by feature

XP

- Planning: release/iteration
- Small releases
- System metaphor
- Simple design
- Continuous testing

- Refactoring – eliminate duplicate code
- Pair programming
- Collective code ownership
- Continuous integration

- 40-hour week
- On-site customer
- Coding standards

- User stories – 3 sentences; 1-3 weeks development
- Etc.

Definitions

Verification:
Are we building the product right?

Validation:
Are we building the right product?

Formal Methods is the application of mathematical reasoning to establish properties about digital systems.

Rockwell Collins

Definitions

Formal methods are mathematically based approaches to software production that use mathematical models and formal logic to support rigorous software specification, design, coding and verification.

"Formal methods can be applied to a few or almost all software development activities: requirements, design and implementation. The degree to which formal methods are applied varies from the occasional use of mathematical notation in specifications otherwise written in English, …"

The goals of most formal methods are to:

Reduce the defects introduced into a product, especially during the earlier development activities …

Place confidence in the product not on the basis of particular tests, but on a method that covers all cases.

US National Cyber Security Partnership

Definitions

High assurance

Theorem proving/Model checking, etc.

Demonstrate routines meet functional specifications

Compilers, Type Checkers

Simple properties

Syntax/semantics

easy — hard

Strong Typing | Extended Static Checking | Partial Correctness | Total Correctness

Reasonably easy

Can require annotations

Some run-time errors prevented (e.g., array bounds)

Based on Homeier slide

Definitions

Business Cases

Technology Transfer

Adoption

CC, FIPS 140-2/3, DO-178B, MoD00-55

Proof theory

Model Checking

Symbolic Execution

Logic

Equivalence Checking

Extreme programming, Spiral, Waterfall, Agile programming, etc.

Methodologies

Standards

Analysis

Theorem Proving

FM is Multidisciplinary

Language design

Application areas

Z/EVES, EVES, PVS, Spark, SMV, SPIN, ESC/Java, Blast, ACL^2, NQTHM, HOL, Isabelle, B

Operational Semantics

Specification languages

Network analysis

Critical systems

Denotational Semantics

Programming languages

Hardware

Axiomatic Semantics

Z, VDM, Verdi, PVS, Spark Ada, Eiffel, JML, Cryptol, etc.

Signalling systems, etc.

Z

$[User, Word]$

LogSys

$$password : User \pfun Word$$
$$reg, active : \mathbb{P}\, User$$
$$\overline{\quad}$$
$$active \subseteq reg = \operatorname{dom} password$$

InitLogSys

$$LogSys'$$
$$\overline{\quad}$$
$$password' = \emptyset$$
$$active' = reg' = \emptyset$$

Register

$$\Delta LogSys$$
$$u? : User;\ p? : Word$$
$$\overline{\quad}$$
$$password' = password \cup \{u? \mapsto p?\}$$
$$active' = active$$

LogIn

$$\Delta LogSys$$
$$u? : User$$
$$p? : Word$$
$$\overline{\quad}$$
$$u? \notin active$$
$$p? = password(u?)$$
$$password' = password$$
$$active' = active \cup \{u?\}$$

- Formal notation for specifying and designing computer systems and software
- Based on set theory
- Blackboard ready
- Oxford
- Standardized and “broadly” adopted
- CBIS, CICS, many other examples …

The Way of Z

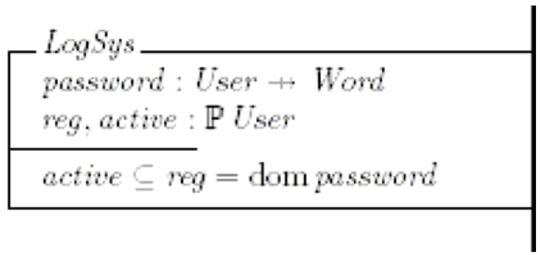
Practical Programming with Formal Methods

Jonathan Jacky

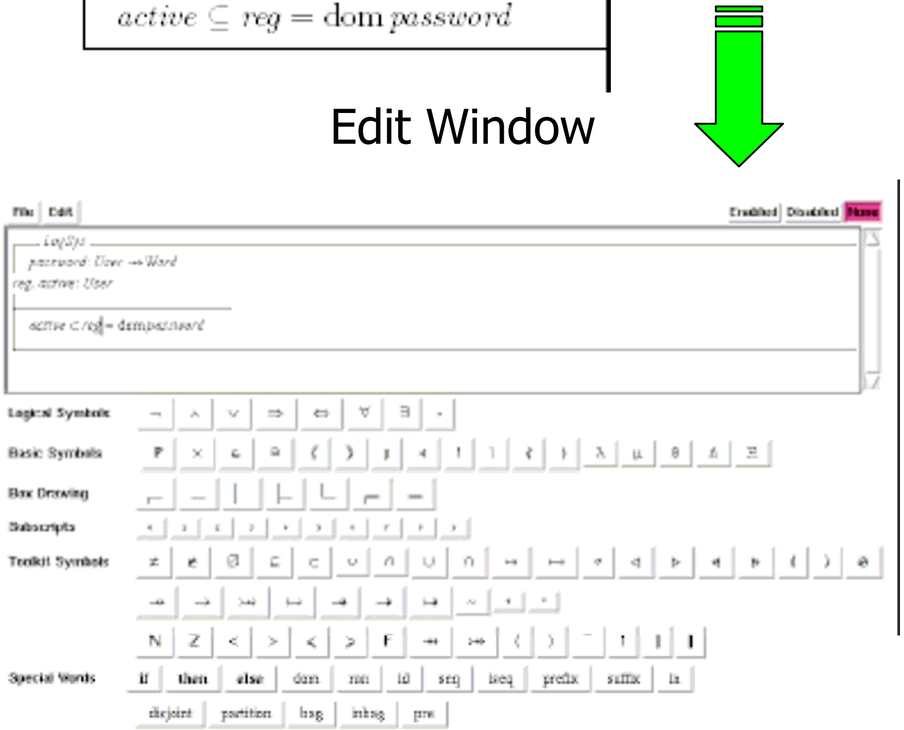
Potter, Sinclair, Till

Z/EVES

- GUI-based system that supports the analysis of Z specifications in several ways:
  - ❑ Syntax and type checking
  - ❑ Schema expansion
  - ❑ Precondition calculation
  - ❑ Domain checking
  - ❑ General theorem proving
- Incremental adoption
- 63 Countries
- ORA Canada/NSA/DND
- CBIS, Crypto protocols …



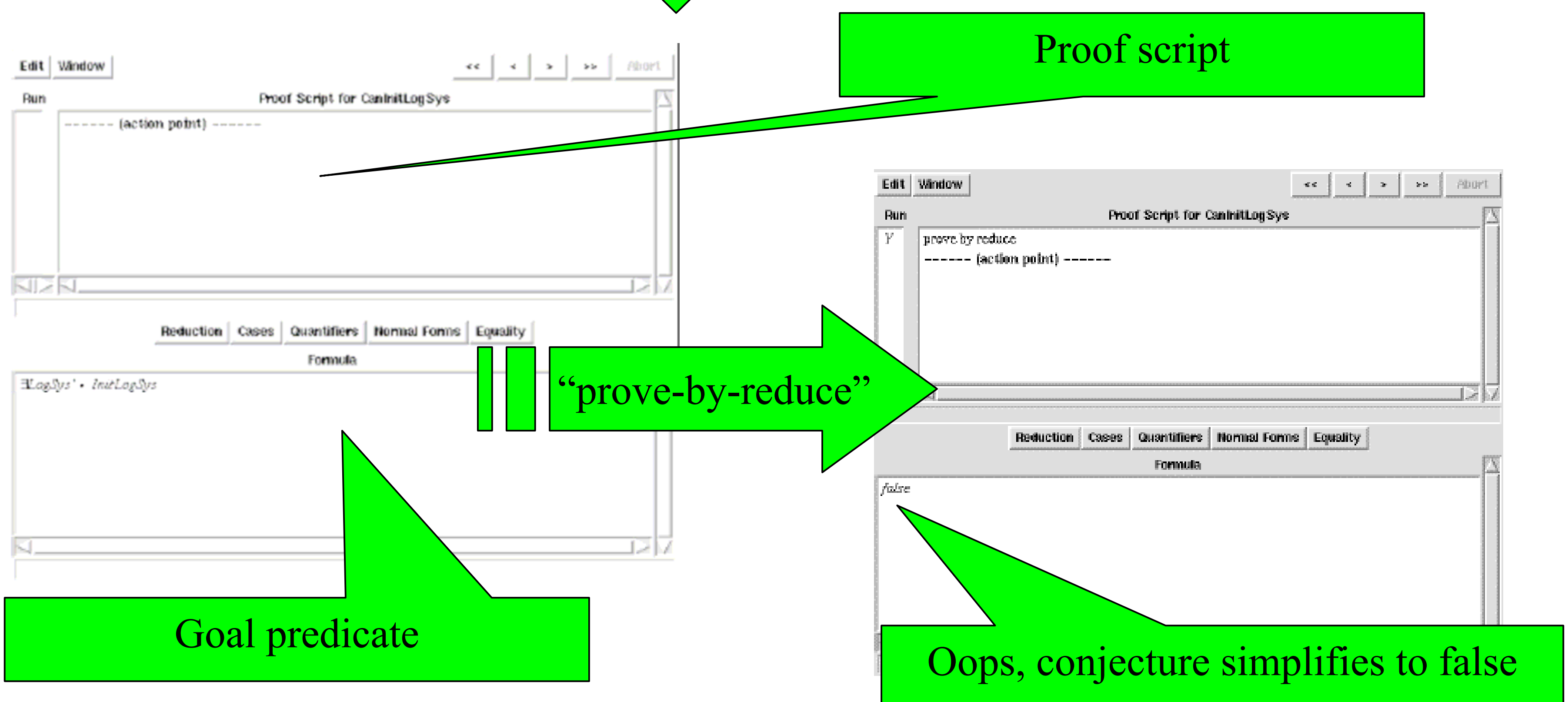
Edit Window



Z/EVES

$$\begin{array}{l}\textbf{theorem } CanInitLogSys \\ \quad \exists\, LogSys' \bullet InitLogSys\end{array}$$

Proof Window



Cryptol

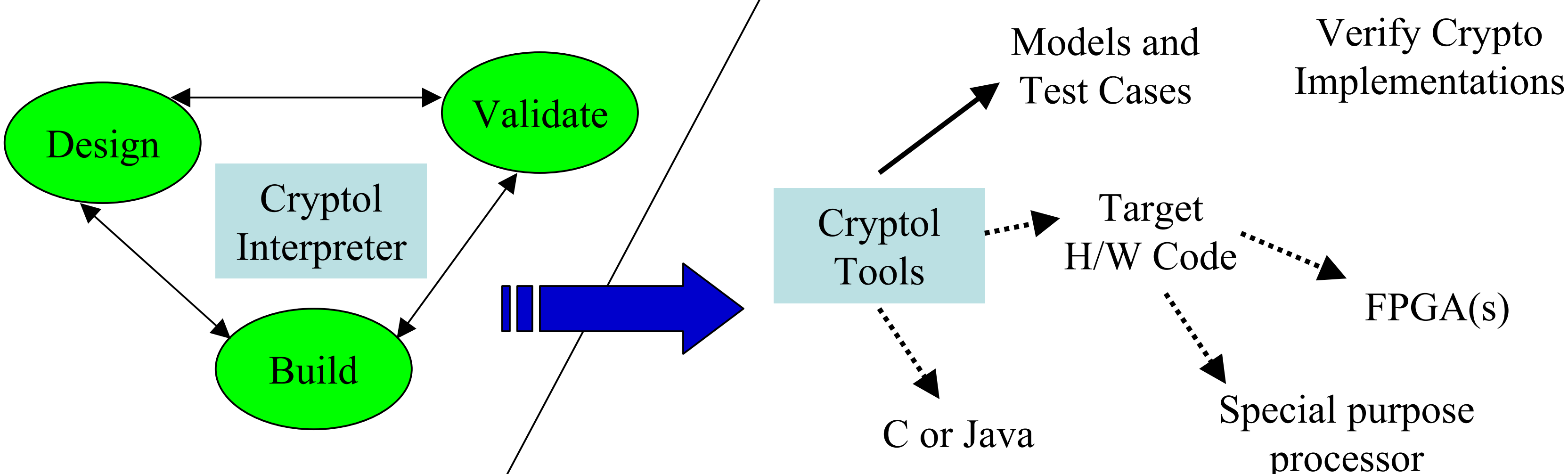
```
des: {a, b} (a>=7)=>
     ([2**(a-1)],[b][48]) -> [64]
des (pt, keys) =
     permute (FP, swap (split last))
 where {pt' = permute (IP, pt);
   iv = [| round (k, split lr)
        || k <- keys
        || lr <- [pt'] #iv |]
   last = iv @ (width keys -1); };

round (k, [l r]) = r # (l ^ f (r, k));
f (r, k) = permute
    (PP,
     SBox (k ^ permute (EP, r)));
swap [a b] = b # a;
permute: {a b} (b >= 1) =>
    ([a][b], [2**(b-1)]) -> [a];
permute (p, m) =
    [| m @ (i-1) || I <- p |];
```

- Galois Communications/NSA, et al.
- Domain specific language for modeling cryptographic algorithms
- Unambiguous (precise, implementation independent)
- Executable (debug, generate test cases)
- Declarative (multiple use)
- Structure and guide implementation
- Reference library for cryptographic algorithms

Cryptol

Domain-specific Design Capture

Assured implementation



Multiple uses from one specification

Cryptol developed by Galois

Derived from Brad Martin Slide

Java Modeling Language

```
package org.jmlspecs.samples.jmltutorial;
import org.jmlspecs.models.JMLDouble;

public class SqrtExample {
 public final static double eps = 0.0001;
 /*@ requires x>=0;
 /*@ JMLDouble.approximatelyEqualTo
     @          (x,  \result*\result, eps); @*/
/*@ signals_only IllegalArgumentException;
   @ signals (IllegalArgumentException e)
   @      e.getMessage() != null && !(x>0.0); @*/

public static double sqrt(double x) {
  if (x>=0 {return internalSqrt(x); }
  else {throw new
   IllegalArgumentException("x is negative:" + x);}}
```

- Annotating & reasoning about Java code
- In-line annotations/pragmas
- Design by contract, documentation, blame assignment, efficiency, modularity of reasoning
- Interface specifications

Gary Leavens, et al.

Iowa State

Extended Static Checker for Java (ESC/Java2)

- Finds run-time errors in JML-annotated Java programs by static analysis
- JML annotations specify degree of checking

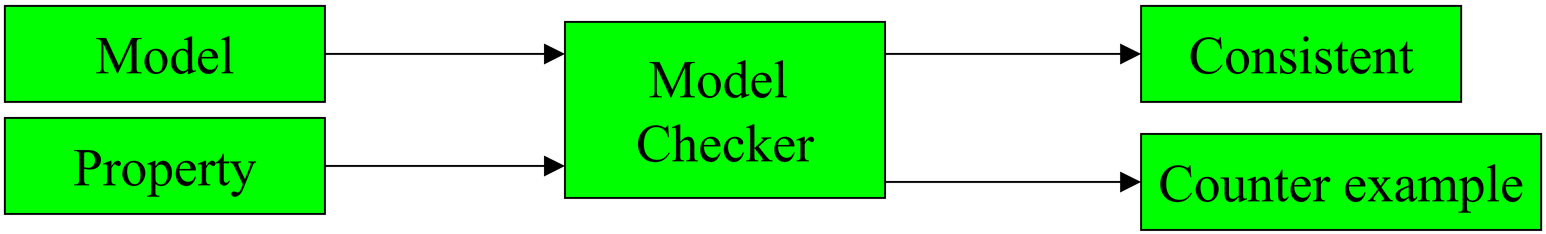
- Feels like a type checker
- Finds subtle errors that testing may miss; catches obscure combinations of conditions
- Potentially a practical, lightweight formal methods tool

- Array index out of bounds
- Division by zero
- Dereferencing a null object
- Unmet entry condition
- Unmet exit condition
- Deadlock
- Race condition, etc.

Kind Software - Ireland

Other Java R&D ongoing

Model Checking



- Model checking is an automatic technique for verifying properties of a finite model of a system
- Exhaustively tests ***all*** states of the model.
- SMV, SPIN, FDR & Murφ principal examples

Advantages

- Exhaustive
- Automatic
- Counter-examples

Disadvantages

- State space explosion
- Model must be finite and not too big – experience needed!

SPIN

Analyzing models of concurrent systems for logical consistency

- Data Communication Protocols.
- Promela (Process Meta Language)
- Synchronous and asynchronous communications
- Creation/destruction of processes

Mutual Exclusion: H. Hyman, CACM 1966

```
bool want[2]; bool turn; byte cnt;
proctype P(bool i) {want[i] = 1;
 do :: (turn != i) -> (!want[1-i]); turn = I
      :: (turn == i) -> break od
 cnt = cnt + 1;
 skip; /*critical section*/
 assert (cnt == 1); cnt = cnt -1;
 want[i] = 0 }
 init (run P(0); run P(1) )
```

Two forms of analysis:

- Random simulations
- Generate C program to perform efficient verification

```
$spin –a hyman1
$gcc –o pan pan.c …
assertion violated (cnt == 1)
```

Critical section violation

Could perform trace!

Gerard Holzmann

ACL²

Theorem proving

AAMP7 Intrinsic Partitioning Separation Theorem – Rockwell Collins – ACL2

The AAMP7G's design was proved mathematically to achieve MILS using Formal Methods techniques as specified by EAL-7 of the Common Criteria.

```
(implies
 (and (secure-configuration spex) (spex-hyp :any :trusted :raw spex fun::st1)
        (spex-hyp :any :trusted :raw spex fun::st2))
 (implies
  (let ((abs::st1 (lift-raw spex fun::st1))
        (abs::st2 (lift-raw spex fun::st2)))
  …
   (equal
     (raw-select seg (lift-raw spex (fun::next spex fun::st1)))
     (raw-select seg (lift-raw spex (fun::next spec fun::st2))))))
```

www.rockwellcollins.com/news/page6237.html

Yes, it is ugly; but Rockwell Collins has received NSA certification for its Advanced Architecture Micro Processor 7 Government Version (AAMP7G), a Multiple Independent Levels of Security (MILS) device for use in cryptographic applications.

Value Propositions

- Product-focused measure of correctness: objective rather than process quality measures
- Early detection of defects
- Guarantees of correctness: e.g., model checkers consider all possible execution paths through a system
- Analytical approach to complexity: e.g., "what-if" analyses, FM better suited than testing

Honeywell on Formal Methods: Analysis of complex systems to ensure correctness and reduce cost

"Firstly, such methods can provide a cost reduction in complex system procurement, through an improved understanding of system design, interfaces and requirements validation and management. Secondly, formal methods can provide increased assurance that critical requirements are met."

Cant, Mahony, McCarthy, Vu
DSTO, Australia

Value Propositions

Is proof more cost-effective than testing?

SHOLIS:

- Ship Helicopter Operating Limits Information System
- Safety-critical, aids the safe operation of helicopters on naval vessels
- Z and SPARK

Value Propositions

Faults Found and Effort Spent During SHOLIS Phases			
Project Phase	**Faults found**	**Effort**	**Faults/Effort**
Specification	3.25%	5.0%	0.65
Z Proof	16.00%	2.5%	6.40
High-level design	1.50%	2.0%	0.75
Detailed design code and informal test	26.25%	17.0%	1.54
Unit test	15.75%	25.0%	0.63
Integration test	1.25%	1.0%	1.25
Code proof	5.25%	4.5%	1.17
System validation test	21.50%	9.5%	2.26
Acceptance test	1.25%	1.5%	0.83
Other	8.00%		

Value Propositions

Why do software projects fail so often? *Charette*

- Unrealistic/unarticulated project goals
- Inaccurate resource estimates
- *Ill-defined system requirements*
- Poor project status reporting
- Unmanaged risks
- *Poor stakeholder communication*

- Use of immature technology
- *Inability to handle complexity*
- *Poor development practices*
- Poor Project Management
- Stakeholder politics
- Commercial pressures

Successes - Microsoft

"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, ***driver verification*** we're building tools that can do actual proof about the software and how it works in order to guarantee reliability."

Bill Gates

WinHec 2002, April 18, 2002

Successes – Intel

Intel's motivation:

- 1994 FDIV error in Intel Pentium processor cost US$500M
- Similar error today would likely cost more
- Intel really interested in technologies to reduce errors

Market pressures leading to increasingly complex designs

- 4-fold increase in errors in Intel processor designs/generation
- 8,000 (approx) errors introduced during the design of the Pentium 4
- Fortunately, pre-silicon detection rates close to 100% … "just enough to tread water."

Intel's success with formal methods
John Harrison
Software, Science and Society, December 5, 2003

Successes – Intel

Extensive testing and pre-silicon simulation

- Slow
- Too many possibilities
  - 2^{160} possible pairs of floating point numbers
  - Vastly higher number of possible states of a complex micro-architecture

FV standard practice in hardware:

- Hardware is designed in a more modular way than most software
- There is more scope for complete automation
- The potential consequences of a hardware error are greater

Harrison

Successes – Intel

- Verification of Intel Pentium 4 floating-point unit using a mixture of symbolic trajectory evaluation and theorem proving
- Verification of bus protocols using pure temporal logic model checking
- Verification of microcode and software for many Intel Itanium floating-point operations, using pure theorem proving

Results:

- FV found many high quality errors in P4 and verified 20% of the design
- FV now standard practice in the floating-point domain

Harrison

Successes – Intel

Proof versus experiment

In mathematics, it is normal to prove results rigorously, and experimental "inductive" testing is exotic and controversial

Testing can miss things that would be revealed by formal proof

In computing, it is normal to establish results by empirical testing and proving them formally is exotic and controversial

Successes – Praxis

- Correctness-by-construction
  - ❑ Do not introduce errors in the first place.
  - ❑ Remove any errors as close as possible to the point that they are introduced.
- Process incorporates formal notations used to specify system and design components with review and analyses for consistency and correctness

- Incremental builds
  - ❑ Removes need for expensive integration phase
- Specification based testing
- Automated test tools to measure code coverage and supplement tests to achieve 100% statement and branch coverage

Successes – Praxis

Project	Year	Size (KSLOC)	Productivity (SLOC/day)	Defects (per SLOC)
CDIS	1992	197	12.7	0.75
SHOLIS	1997	27	7.0	0.22
MULTOS CA	1999	100	28.0	0.04
…A…	2001	39	11.0	0.05
TIS Core	2003	10	38.0	0.00

FAA Presentation: [post-delivery figures]

- Reliable systems: 0.5-1 defects/KSLOC
- Reasonable commercial system: 3-6 defects/KSLOC [post-delivery]
- Poor system: >15 defects/KSLOC
- [But SCC says 30 defects/KSLOC]
- Root cause of most software errors: Lack of complete understanding of the correct design space

Successes – Praxis

Size and Productivity				
SLOC			**Productivity (LOC/day)**	
	Ada	**Spark**	**During coding**	**Overall**
TIS Core	9939	16564	203	38
Support software	3697	2240	182	88

Industry average for Ada is approximately 20LOC/day
www.dacs.dtic.mil/techs/baselines/productivity.html

Successes – Z/EVES

Everett Rogers (Sloan)

- ❑ Relative advantage
- ❑ Compatibility
- ❑ Complexity
- ❑ Trialability
- ❑ Observability
- ❑ Transferability
    - Prior technology drag, irreversible investments, sponsorship, expectations

- EVES to Z/EVES
- 3 countries to 63
- But few commercial opportunities

Technology transfer:

- **Geoffrey Moore**
- **Clayton Christensen**

"Formal Methods Technology Transfer: Impediments and Innovation," September 1995. Craigen, Gerhart, Ralston

Standards

FM required (recommended) in numerous standards:

- Common Criteria (EAL 5-7) [International]
- FIPS 140-2 (Level 4) [US]
- Defence Standard 00-55 (00-56) [UK]
- Defence Standard 5679 [Australia]
- DO-178B (Level A) [US/International]
- Etc…

Formal Methods Myths

- Can guarantee that software is perfect.
- Are all about theorem proving.
- Are only useful for safety-critical systems.
- Require highly trained mathematicians.
- Are unacceptable to users.
- Are not used in real, large-scale software.
- Increase the cost of development.

Anthony Hall, 1990

Conclusions

Equivalence checking: 1M gate ASICS

Model checking:1000 latches at a time – claims of 10^{20} states

Software verification (design to code): ~80KLOC

Verified compilers for special purpose languages

Static analysis: >150KLOC

Specification and modelling: >30KLOC of specification

Bloomfield & Craigen, 2000

FM99 (Toulouse, FR) estimate of FM activities: $1-2B

Craigen, 1999

Halloween

Cell phone ring tones

Conclusions

From a mathematics perspective:

Soundness is good!

From a tech transfer/engineering perspective:

Unsound and incomplete may be better!

This is a hard lesson!

Value propositions vary with communities

Conclusions

Many potential applications

- Software
- Hardware
- Algorithms
- Protocols
- Reverse Engineering
- Standards

Increasing body of successful projects and adoption

But impediments remain: social, process, technical

Conclusions

Is Software Assurance an Oxymoron?

Perhaps, not. However, there is substantial room for improvement.

Is Mathematics a Resolution?

Extremely helpful, but software assurance is multi-faceted and various impediments remain, including lack of industry maturity.

References

1. "Validation, Verification and Certification of Embedded Systems," NATO RTO-TR-IST-027, October 2005, www.rta.nato.int/Main.asp?topic=ist.htm#recent

2. "Formal Methods Diffusion: Past Lessons and Future Prospects," Bundesamt fur Sicherheit in der Informationstechnik (BSI), Robin Bloomfield and Dan Craigen, www.bsi.bund.de/fachthem/fmethods/sonstige/fms_v1.0.pdf

3. "Formal Methods Adoption: What's Working, What's Not!" Keynote presentation for SPIN 1999, Dan Craigen, www.fee.uwaterloo.ca/~sleue/6thSPIN99.html