Rigorous Software Development
An introduction

Simão Melo de Sousa

RELEASE (UBI), LIACC (Porto), CCTC (Minho)
Computer Science Department
University of Beira Interior, Portugal

October 2011
1. Context and Motivations

2. Formal Software Engineering and Software Verification
Table of Contents

1. Context and Motivations

2. Formal Software Engineering and Software Verification
Software and cathedrals are much the same. First we build them, then we pray.
*Sam Redwine, at the 4th International Software Process Workshop - 1988*

The Formal Methods Approach:

*Aide toi, et le ciel t’aidera!*
*Jean de La Fontaine - Le chartier embourbé. Livre VI - Fable 18.*

... That is what this short introduction is about.
Main reference for this lecture

Rigorous Software Development, A Practical Introduction to Program Verification
Series: Undergraduate Topics in Computer Science.
Reliability as Fault Avoidance and not Fault Tolerance

Introduction to the methods and tools that provide means to design Information and Computer Systems (ICS) that are functionally and logically correct.
modern ICS = new challenges

- reliability
- correctness
- security
- safety
- robustness
- etc...

as a central challenge to the design of modern ICS.

- was and still is a central issue in the design of safety critical systems.
- The *reliability* quest lies at the foundations of Computer Science. Establishing its rigorous and complete foundations is a debate as old as Computer Science itself, indeed older ... (but Computer Science is a very young discipline).
Dysfunctions are expensive

- ICS maintenance = $\frac{2}{3}$ of total costs;
- Dealing with dysfunctions or bugs = 20 times more expensive after than before production.
- Apart from these software life cycle considerations, it is well known that bugs can have deep impact in the customers’ trust (e.g. Pentium bug) or financial losses (IBM vs FAA), letting aside human life losses (e.g. Therac 25) or ICS where bugs are simply unacceptable.
About 16 years ago, W. Gibbs said:

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.


Past? Provocation?

Unfortunately no. See for instance the usual software equation:

Software in the market = bug report channel
Unfortunately,

There is no definite solution!

Nevertheless, there are satisfactory solutions:

**Reshape and Adapt** the software life cycle in order to include **reliability** as a central requirement.

⇓

Common Criteria (EAL 5-7), Cenelec / IEC 61508 (Safety Integrity Level 3 and 4), DO-178B (Design Assurance Level A and B).
Unfortunately, there is no definite solution!

Nevertheless, there are satisfactory solutions:

**integration and use** of:
- Tests and Simulation
- Formal Methods
Main Principle

For a given **model** of the target ICS, **provide** a set of input data that are **representative** and **compare** the output answer with the expected result.

But, good tests are rare:

**exhaustiveness:** in the general case, impossible (the set of possible values is possibly infinite)

**representativeness:** in the general case, problems occur when unexpected values are given as input.
An illustrative example (From J. Harrison Lectures):

Littlewood proved, in 1914 a surprising result: the function

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

where

$$li(n) = \int_0^n \frac{du}{\ln(u)}$$

and

$$\pi(n) = \text{number of primes } \leq n$$

changes its sign infinitely. Surprising? yes, because despite intensive (manual) tests (up to $10^{10}$, before the computer era), no sign changes were detected until this proof.
A general definition

The term Formal Methods (FM) refers to the use of mathematical modelling, 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 modelling of software is to highlight the character of the mathematics involved.


The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to the reliability and robustness of a design.

from wikipedia.

In a broad sense, FM is intended to be for ICS engineering what physics is already for aerospace engineering.
Some properties

exhaustiveness: Formal reasoning over the possibly infinite set of values.

rigour: well established mathematical foundations

adequacy: provided tools and techniques are evaluated as adequate means for providing evidences of reliability (see for instance the ISO standard Common Criteria for the security of ICS)
Some considerations

Tests vs. FM

- (Edsger W. Dijkstra):
  
  *Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.*

- But, Formal Methods only prove what can be (indeed, what is) carefully stated (D. Knuth):
  
  *Beware of bugs in the above code; I have only proved it correct, not tried it.*

- Thus, (D. Syme & A. Gordon):
  
  *if you specify you must test.*
Table of Contents

1. Context and Motivations

2. Formal Software Engineering and Software Verification
Formal Methods intend to provide means to:

ensure that a given ICS has a given (adequate) behaviour
Formal Software Engineering and Software Verification

The main issue

Formal Methods intend to provide means to:

ensure that a given ICS has a given (adequate) behaviour

Central notion of \( \{ \begin{align*} & \text{model} \\ & \text{specification} \end{align*} \) to bring the object under study to mathematics (logic, algebra, etc...).
The main issue

Formal Methods intend to provide means to:
ensure that a given ICS has a given (adequate) behaviour

In practice: this central issue is divided into two sub-problems

evidences of behaviour: How to ensure/verify the expected behaviour at the specification level.

model against code:

1. How to obtain, from a well behaved model, a program with the same behaviour?
2. or, provided code and specification, how to ensure that both share the same behaviour/properties?
Software Life Cycle: The classics...

- Waterfall (Boehm, 1977)
- V, Spiral, etc...

Semi-formal Methods:

- SA, SADT, SSADM, MERISE, UML, etc...
- Deep System Analysis, but do not provide strong evidence w.r.t. reliability requirements
- Nevertheless, positive contributions.
Development Process and Requirements

- **Design:**
  Each step: a more finely grained look at the system than the previous step. Requirements of the previous step must be refined. The new requirements should reflect the upper stage requirements.
  Each stage: should produce reports/outputs that form the basis of the next stage and of corresponding validation step.

- **Validation:**
  Should give an answer (hopefully positive) to the challenges stated by the related design stage reports.
  Essentially based on tests and simulation in a controlled environment
Several (safety/reliability/security based) software standards highly recommend the use of mathematically based tools and methodologies to ensure the absence of determined errors: DO-178B, EN 50128 (see figure), GalileoSW, IEC 61508.

The ISO/IEC-15408 Common Criteria (for the security of IT) indeed requires the use of such tools for the highest certifications.
Formal Software Development in a picture

Balzer Software Life cycle

Diagram showing the Balzer Software Life cycle, including stages such as Formal Specification, Analysis, Formal Analysis and Verification, Prototyping, Simulation, and Customer feedback.
At this point we should stress that one can roughly divide the formal methods tools into two main families:

1. **System Validation/Verification Tools**: Horizontal application of the Balzer Life Cycle
2. **Correct by Construction approach**: Downward application of the Balzer Life Cycle

The former clearly provide means to address only the first sub-problem (with very specialized techniques and tools), while the second intend cover the whole issue.
Formal Software Development in a picture

Applied Balzer Software Life cycle

System and Requirement Analysis
System / High Level Design
Subsystem / Architectural Design
Module Design
Code / Implementation

System Test
Integration Test
Functional Test
Module / Unit Test

Feedback

Vertical Application

Horizontal Application
Are the FM tools ready to be used by the ICS industry?

- Tools maturity and usability: from my own experience, there is a real (and constant) improvement.

- Does a software company need to have FM experts in order to use FM? Yes, at least one.

- Does the FM expert need to have a PhD in order to successfully apply FM? No (personal experience).

- FM tools still have problems to cope with modularity and scalability (size, etc...). . .

- But, Formal Verification "tours de force" (e.g. works from Leroy, Barthe, Klein, Cousot, Nipkow, . . .) are increasingly frequent...

---

1000000 of Verified lines of code: You can't say any more it can't be done! Here, we've done it! (Jim Woodcok - Formal Verification of Software)
Is the ICS industry ready to use FM?

First. ICS industry should realize that FM are not a product (as anti-viruses).

(This is in fact happening (or already the case) in several ICS key area)

But ICS industry already has ICS development process and does not want to lose it.

Thus, adopting FM implies reshape and adapt the in-house ICS design "savoir-faire".

⇒ FM tools should take the Balzer life-cycle into consideration

ICS standards about security and reliability recommend nowadays the use of FM for the highest levels of certification.

Shift paradigm: next versions of these standard (that really rule Critical system industry ) are willing to require the use of FM.

notable exception : Common Criteria, that already requires.

lack of specialists, IVV is a emerging market.
Formal Software Engineering and Software Verification

Pragmatical considerations on the application of FM

What is to be formally specified and verified?

- Formal Specification and Verification are far from easy (and are not cheap, but the real cost has to be considered in the long term)
- General practice: determine the critical parts of the ICS to be verified. Apply FM on these parts