Though This Be Madness, Yet There Is Method in It?

Alan Wassyng
Outline

- Background & Introduction
- Why Aren’t Developers Using Formal Requirements?
- Science & Engineering vs CS & SE
- Methods & (not so) Integrated Tools
- Conclusions
Background

- Biased by:
  - educational background - Applied Math (SE self taught)
  - experience - 15 years academia, 15 years industry, 11 years academia (with significant industrial partnerships)
  - goal – rigorous, practical SE in practice, especially in critical systems

- Joint work with:
  - Mark Lawford, Tom Maibaum, Paul Joannou

- Supported by discussions with:
  - Dave Parnas, Sushil Birla, Paul Jones, John Hatcliff, Brian Larson, John Knight, Rance Cleaveland, Michael Holloway, Jens Weber, Oleg Sokolsky, Insup Lee, Mats Heimdahl, Joe D’Ambrosio, John Fitzgerald

- Additional long-time support from:
  - Dave Parnas, Glenn Archinoff, Rick Hohendorf, Mike Viola, Greg Moum, David Lau, Jeff McDougall, David Tremaine, Dominic Chan, Alanna Wong, Peter Froebel
Introduction

- I am going to start with some perceived problems – and some thoughts on why they are problems, and why we have not done more to tackle them
- We’ll start near the beginning – requirements
  - Formal requirements should be a showpiece of our domain
  - We all know how we sell them – precise and unambiguous; make us think in a more focused way about them; foundation for mathematical verification or model driven development; can be checked for consistency, existence of properties; …
  - What is the common community wisdom about why they are not used more?
    - Perceived to be more expensive (but we say they are not)
    - Companies budget projects by ramping up – does not fit well
    - Perceived to be difficult to do (sometimes they are)
    - Lack of knowledge and training
Requirements in the ‘real’ world
This is our opportunity!

8.0  \( Points = (Track-id \times Track-id) \xrightarrow{m} (Track-id \xrightarrow{m} Point-control); \)

11.0  \( PointsInTracks : Tracks \times Points \rightarrow \mathbb{B} \)

  .1  \( PointsInTracks (tracks, points) \triangleq \)
  
  .2  \( \text{dom points} \subseteq tracks; \)

15.0  \( Control2Dir : Points \rightarrow \mathbb{B} \)

  .1  \( Control2Dir (points) \triangleq \)
  
  .2  \( \forall mk- (t1, t2) \in \text{dom points} \cdot \)
  
  .3  \( \{t1\} \triangleleft points (mk- (t1, t2)) = \{t1 \mapsto \text{LEFT}\} \Rightarrow \)
  
  .4  \( \exists mk- (t, t') \in \text{dom points} \cdot \)
  
  .5  \( t = t1 \land \)
  
  .6  \( \{t1\} \triangleleft points (mk- (t1, t')) = \{t1 \mapsto \text{RIGHT}\} \land \)
  
  .7  \( \{t1\} \triangleleft points (mk- (t1, t2)) = \{t1 \mapsto \text{RIGHT}\} \Rightarrow \)
  
  .8  \( \exists mk- (t, t') \in \text{dom points} \cdot \)
  
  .9  \( t = t1 \land \)
  
  .10  \( \{t1\} \triangleleft points (mk- (t1, t')) = \{t1 \mapsto \text{LEFT}\}); \)
This is our opportunity!

8.0 \( \text{Points} = (\text{Track-id} \times \text{Track-id}) \xrightarrow{m} (\text{Track-id} \xrightarrow{m} \text{Point-control}); \)

11.0 \( \text{PointsInTracks} : \text{Tracks} \times \text{Points} \rightarrow \mathbb{B} \)

1. \( \text{PointsInTracks} \ (\text{tracks}, \text{points}) \triangleq \)

2. \( \text{dom points} \subseteq \text{tracks}; \)

15.0 \( \text{Control2Dir} : \text{Points} \rightarrow \mathbb{B} \)

1. \( \text{Control2Dir} \ (\text{points}) \triangleq \)

2. \( \forall \text{mk-} \ (t_1, t_2) \in \text{dom points} \cdot \)

3. \( \{t_1\} \triangleright \text{points} \ (\text{mk-} \ (t_1, t_2)) = \{t_1 \leftrightarrow \text{LEFT}\} \Rightarrow \)

4. \( \exists \text{mk-} \ (t, t') \in \text{dom points} \cdot \)

5. \( t = t_1 \land \)

6. \( \{t_1\} \triangleright \text{points} \ (\text{mk-} \ (t_1, t')) = \{t_1 \leftrightarrow \text{RIGHT}\} \land \)

7. \( \{t_1\} \triangleright \text{points} \ (\text{mk-} \ (t_1, t_2)) = \{t_1 \leftrightarrow \text{RIGHT}\} \Rightarrow \)

8. \( \exists \text{mk-} \ (t, t') \in \text{dom points} \cdot \)

9. \( t = t_1 \land \)

10. \( \{t_1\} \triangleright \text{points} \ (\text{mk-} \ (t_1, t')) = \{t_1 \leftrightarrow \text{LEFT}\}; \)
This is our opportunity!

8.0  \( Points = (\text{Track-id} \times \text{Track-id}) \rightarrow^m (\text{Track-id} \rightarrow^m \text{Point-control}); \)

11.0  \( PointsInTracks : \text{Tracks} \times \text{Points} \rightarrow \mathbb{B} \)

1.  \( PointsInTracks (\text{tracks}, \text{points}) \triangleq \)

2.  \( \text{dom points} \subseteq \text{tracks}; \)

15.0  \( Control2Dir : \text{Points} \rightarrow \mathbb{B} \)

1.  \( Control2Dir (\text{points}) \triangleq \)

2.  \( \forall \text{mk-} (t1, t2) \in \text{dom points} \cdot \)

3.  \( (\{t1\} \triangleleft \text{points} (\text{mk-} (t1, t2))) = \{t1 \mapsto \text{LEFT}\} \Rightarrow \)

4.  \( \exists \text{mk-} (t, t') \in \text{dom points} \cdot \)

5.  \( t = t1 \land \)

6.  \( \{t1\} \triangleleft \text{points} (\text{mk-} (t1, t')) = \{t1 \mapsto \text{RIGHT}\} \land \)

7.  \( (\{t1\} \triangleleft \text{points} (\text{mk-} (t1, t2))) = \{t1 \mapsto \text{RIGHT}\} \Rightarrow \)

8.  \( \exists \text{mk-} (t, t') \in \text{dom points} \cdot \)

9.  \( t = t1 \land \)

10. \( \{t1\} \triangleleft \text{points} (\text{mk-} (t1, t')) = \{t1 \mapsto \text{LEFT}\}; \)
Let’s continue with examples …
and I cannot resist the Hamlet aside
And – are these any better?

Simple tabular expressions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Triggering Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>(%Power% = $on$) ∧ (%ShutDown% = $operate$) ∧ (%Watchdog% = $operate$)</td>
<td>@T(%Reset% = $released$)</td>
</tr>
<tr>
<td>(%Power% = $off$) ∨ (%ShutDown% = $shutdown$) ∨ (%Watchdog% = $shutdown$)</td>
<td>@T(%Reset% = $pressed$)</td>
</tr>
<tr>
<td>%PumpSwitch% =</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td></td>
<td>$closed$</td>
</tr>
<tr>
<td></td>
<td>$open$</td>
</tr>
</tbody>
</table>

Delimiters to classify variables

GTS_infoGetIds(t, data) → list:

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = Ø</td>
<td>undefined</td>
</tr>
<tr>
<td>t ≠ Ø</td>
<td>∀ j, ((0 ≤ j &lt;</td>
</tr>
</tbody>
</table>
Madness

- Insanity (the legal term)
  - mental illness of such a severe nature that a person cannot distinguish fantasy from reality

Fantasy

Reality
Back to Madness

- What about methods & tools?
- We usually have lots to say about the use of UML in industry …
Software Tools in Industry

- Predominantly UML based
  - 116 on design alone - $90 - $12,000
  - A few comprehensive tool chains – some (very few) are based on formal approaches
  - Requirements phase is an excellent demonstration of what can go wrong when we have the wrong focus – next topic ...

- Majority of “rigorous” tools are model checkers, theorem provers, SAT/SMT solvers and static analysis tools – cost varies from free to $$$
  - Major restrictions in many of them, scalability is a huge problem, and most require advanced expertise to use them

- Most of the non-rigorous tools help with documentation but do not help or prescribe much on the technical side
  - “Prescription” seems to be a dirty word in software
Software Tools in Industry

- Predominantly UML based
  - 116 on design alone - $90 - $12,000
  - A few comprehensive tool chains – some (very few) are based on formal approaches
  - Requirements phase is an excellent demonstration of what can go wrong when we have the wrong focus – next topic …

- Majority of “rigorous” tools are model checkers, theorem provers, SAT/SMT solvers and static analysis tools – cost varies from free to $$$
  - Major restrictions in many of them, scalability is a huge problem, and most require advanced expertise to use them

- Most of the non-rigorous tools help with documentation but do not help or prescribe much on the technical side
  - “Prescription” seems to be a dirty word in software
Look at almost all system requirements methods/tools – including pretty good comprehensive tool suites

- Notice that they do what computer scientists have been advocating for years – all variables are *typed*
- Obviously all those hard earned lessons have paid off!

**BUT**
Look at almost all system requirements methods/tools – including pretty good comprehensive tool suites
  • Notice that they do what computer scientists have been advocating for years – all variables are typed
  • Obviously all those hard earned lessons have paid off!

BUT

At the initial requirements level what we need are units – not data types!
  • x: integer or y: float is not useful at this stage
  • x m/s or y kg is what we need
An Aside on Academia & Industry

Academics keep giving us tools and methods that don’t work in practice! They don’t scale, have stupid assumptions – and I hate using that notation.

Software developers in industry are not aware of advances in software engineering and don’t use the tools and methods that already exist – and would help them.

I have to have this finished by tomorrow – I hope it works

Can’t make this method scale – have to write another paper
An Aside on Engineering

- Application of science and economics to design and build objects and/or processes – AND protect the public from harm
- Builds on basic science
- Technicians typically build and maintain the artifacts
Example Engineering Fields

- **Civil Engineering**
  - Science of materials, mechanics, elasticity, (physics & chemistry)
  - Mathematics (continuous mainly)
  - Economics
  - Technicians are: contractors in various domains

- **Software Engineering**
  - Mathematics
  - *Computer Science*
  - Economics
  - Physics & chemistry & …
  - Technicians are: Anyone, *computer scientists*, software engineers, … we typically call them ‘software developers’
Software Engineers

- Engineers produce methods and tools for use by engineers and technicians – engineers have a duty to society to build effective artifacts that do not jeopardize public safety
- Software Engineers are engineers in the domain of software intensive systems
Computer Scientists

- Each engineering field is built on basic sciences applicable to that domain
- Computer science is one of the basic sciences that provides a basis for software engineering
- (Computer science is also one of the basic sciences for electrical engineering)
Software Developers

- **Technicians in the software domain**
  - Self-taught / trained “programmers” .. and computer scientists! Software Engineers, and other engineers

- **Computer scientists** fill both the role of providing the basic science for software engineering, as well as being technicians who use the methods and tools produced by software engineers

- **Nikkei 2001**: 58% of embedded systems workforce in Japan has no training in software!

- Mostly, a computer scientist who works on basic science does not also work as a software developer

- Aside: our experience is that there are many mechanical engineers developing software for medical devices!
Formal Methods

- The application of rigorous mathematical approaches and techniques to the creation of development (includes analysis) methods as well as the actual development of software applications
- Observation: primarily computer scientists and mathematicians
Our hypothesis is that we see this trend in which we forget to specify units for requirements variables because we are losing sight of the big picture.

We are too focused on formal methods for the sake of formality. The formal methods movement started with the intent of using mathematics to help us cope with very real, practical problems.

It reminds me of the “new math” movement in teaching math in the US in the late 1960s – “Why Johnny Can’t Add” – Morris Kline.
How successful are we?

- How do we measure this?
  - Scientifically designed surveys (some people are doing it)
  - Look at what developers are using in practice

- What evidence do we have right now?
  - Anecdotal stories of success (/ failure)
  - A (very) few good surveys
  - Formal methods light
  - Papers on the use of formal methods
  - Number of papers published in the field
    - Highlights the problem!
  - Rationalization
    - Industry funding model
  - Have we abstracted away details that are essential if we want our work to be relevant?
That’s the end of the introduction

u still mad?
Requirements – what do we want?

- Precise & Unambiguous
- Complete
- Rationale
- High enough level of abstraction
- Minimal design
- Support for verification & validation
Requirements – what do we want?

- Precise* & Unambiguous*
- Complete*
- Rationale
- High enough level of abstraction
- Minimal design
- Support for verification* & validation*

* and these are the reasons we turn to a formal syntax & semantics
Requirements – what do we want?

- Precise & Unambiguous
- Complete
- Rationale
- High enough level of abstraction
- Minimal design
- Support for verification & validation

understandable
by
domain experts
Understandable?

- Actually, “understandable” should be a short term goal - and “understandable” does not mean understandable with minimal effort!
- To be successful, we should want domain experts to be able to (help) develop and document the system level requirements – and those requirements should be in a form that can be used directly to document the software requirements.
But this is not the most important issue

- If we create a requirements method (includes notation and model) we must not do it in isolation!
- Each life-cycle phase in the software development process should consider:
  - If the phase is preceded by another phase, how do we make use of the outputs of that phase?
  - If the phase is succeeded by another phase, what do we need to produce in this phase that will be most useful in the following phase?
  - In other words – don’t locally optimize each phase. We need to plan how to integrate them to minimize rework, avoid the introduction of errors, and give support to both developers and certifiers
  - This includes both the process aspects and the documentation that results in each phase
Life-cycle Integration

Legend:
- Documents produced in the forward going development process
- Documents produced for verifications, reviews and testing
- Tools connected to documents/activities
- Activities and data flow

Formal Requirements Documents

Design Review and Verification Reports

Software Design Document

Validation Test and Reliability Qualification Reports

Software Integration Test Report

Unit Test Report

Requirements Review Report

Designers document design decisions using SFTs to use reqs in class interface specs & design verification

"Get, Process, Set" used as a design convention to aid design verification

Safe state convention aids fault tolerant code

Design is used to comment the code - also aids code verification

List of secrets to aid design decomposition

32
Principle 1

- Integration
  - Integration of the life-cycle phases
    - For years we have discussed the integration of tools into comprehensive tool chains
    - The best way of integrating tools is to first produce integrated methods – tools that aid us in implementing those methods will form comprehensive, integrated tool chains
  - Integration of practical & effective formal methods into the entire life-cycle
    - It is all too common for formal methods practitioners to show how their method works on just a single phase or even a relatively small component of a life-cycle phase
    - Have to include concepts not related to formality, but that still help to produce dependable software (rationale for instance)
    - Mundane tools that implement glue in truly integrated methods, may also be incredibly useful
Principle 2

- Ground formal methods in software engineering – it needs that context to direct the research.
### Example: tabular expressions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
</table>
| \( f_{\text{NOPsenttrip}_i} \leq \text{Calibrated } i^{\text{th}} \text{ NOP signal} \)  
\( \{\text{Calibrated NOP signal, is now in the trip region}\} \)          | \( e_{\text{Trip}} \)        |
| \( f_{\text{NOPsp}} - k_{\text{NOPphys}} < \text{Calibrated } i^{\text{th}} \text{ NOP signal} < f_{\text{NOPsp}} \)  
\( \{\text{Calibrated NOP signal, is now in the deadband region}\} \)          | No Change |
| \( \text{Calibrated } i^{\text{th}} \text{ NOP signal} \leq f_{\text{NOPsp}} - k_{\text{NOPphys}} \)  
\( \{\text{Calibrated NOP signal, is now in the non-trip region}\} \)          | \( e_{\text{NotTrip}} \)        |

- complete & disjoint
- natural language expression
- prefixes rather than delimiters
Example: Making it usable

### Original style condition table

<table>
<thead>
<tr>
<th>{For each i = 1,..,18}</th>
<th>f_NOPsp ≤ Calibrated $i^{th}$ NOP signal</th>
<th>f_NOPsp &gt; Calibrated $i^{th}$ NOP signal</th>
<th>Calibrated $i^{th}$ NOP signal ≤ f_NOPsp - k_NOPPhys</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_NOPsentrip$_i$</td>
<td>e_Trip</td>
<td>No Change</td>
<td>e_NotTrip</td>
</tr>
</tbody>
</table>

### Revised style condition table

<table>
<thead>
<tr>
<th>{For each i = 1,..,18}</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>f_NOPsp ≤ Calibrated $i^{th}$ NOP signal</td>
<td>e_Trip</td>
</tr>
<tr>
<td></td>
<td>f_NOPsp - k_NOPPhys &lt; Calibrated $i^{th}$ NOP signal &lt; f_NOPsp</td>
<td>No Change</td>
</tr>
<tr>
<td></td>
<td>Calibrated $i^{th}$ NOP signal ≤ f_NOPsp - k_NOPPhys</td>
<td>e_NotTrip</td>
</tr>
</tbody>
</table>

- no rules
- reads left to right
- inequalities always same direction
Notation

2.1.3.9.1 Neutron Overpower Parameter Trip

2.1.3.9.1.1 Inputs/Natural Language Expressions

<table>
<thead>
<tr>
<th>Input</th>
<th>NL Expression</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{NOPsentrip}}_i, i=1,..,18$</td>
<td>-</td>
<td>2.1.3.9.2.4</td>
</tr>
</tbody>
</table>

2.1.3.9.1.2 $c_{\text{NOPparmtrip}}$

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any ($i \in 1,..,18$) ($f_{\text{NOPsentrip}}<em>i = e</em>{\text{Trip}}$)</td>
<td>$e_{\text{Trip}}$</td>
</tr>
<tr>
<td>{Any NOP sensor is tripped}</td>
<td></td>
</tr>
<tr>
<td>All ($i = 1,..,18$) ($f_{\text{NOPsentrip}}<em>i = e</em>{\text{NotTrip}}$)</td>
<td>$e_{\text{NotTrip}}$</td>
</tr>
<tr>
<td>{All NOP sensors are not tripped}</td>
<td></td>
</tr>
</tbody>
</table>

We did not use $\forall$, $\exists$ and other dense math notation.

Comments

Hyperlinks
Table Tools

- Tabular expressions proved to be readable (maybe even writable) by nuclear domain experts
- In general, practitioners from all fields seem to like them
- No sequence of operation imposed – they are simply a notation for describing functions
- Original tools were not very good – definitely hampered getting others to adopt tabular expressions
- Mark Lawford has produced a *Tabular Expression Toolbox* that works mostly the way we wanted – and uses a Simulink frontend to hide SMT solver & theorem prover
Principle 3

- Make methods and tools that are useful to and usable by average educated practitioners
  - Example: tabular expressions - with natural language expressions
  - It took about 2 years to get tables to the point where we had the right sort of usability!
  - Still need much more work to have semantics and tools for tabular expressions to allow us to:
    - Compose tables (functional composition)
    - Break apart tables to form new tables
Control complexity

- Complexity has become the number one enemy of being able to build dependable systems
- We should do everything we can to control it – and that often means resisting adding unnecessary features
- In the nuclear domain in some countries, it is mandatory to separate control and safety systems
- This is an incredibly powerful way of controlling complexity
- You can see similar ideas in other domains – safety or security kernels, etc.

Example:
- Darlington Nuclear Power Plant control system: > 500K LOC
- Darlington Shutdown Systems: each < 40K LOC
Principle 5

- Experimental validation of methods and tools
  - Expensive
  - Difficult
  - Essential – if we don’t do this we remain a faith-based community instead of a science-based community
Principle 6

• Prescription in standards and certification
  • Learn enough about our product to become much more prescriptive about what attributes we have to see in software systems (being able to measure them would be good)
  • If we have evidence about the efficacy of methods and tools we can become much more prescriptive about them as well
  • We know that many colleagues have written about why we should not be prescriptive – we just don’t agree 😊
  • Other engineering domains are essentially more prescriptive than we are! Mainly tied up with the fact that ours is a relatively new domain, extremely complex, and software is … soft – and we don’t know how to be prescriptive
  • Example: FAA and enforcement of DO-178x
Principle 7

- Develop software intensive systems with certification as a goal
  - Dependability (reliability, safety, security) cannot be bolted on afterwards
  - Even if your domain is not regulated – develop the software as though it were, and demonstrate to yourselves that it is of adequate quality
  - Certification is not verification – it is much more
  - Why do we focus so much on verification?
    - Because we have to
    - We don’t know yet how to build software intensive systems so that they are correct by construction – that should be our goal
Principle 8

- Formal methods have to cope with practical software development
  - We cannot afford to ignore practical issues on the grounds that it makes the math more difficult
  - Example: timing requirements – we have to be able to cope with tolerances on time durations
    - $t \mod k_{\text{delay}} = 0$
    - If $k_{\text{delay}} = 500$ ms then all is good
      - $500, 1000, 1500, 2000, 2500, 3000, 3500, \ldots$
    - If $k_{\text{delay}} = 500$ ms $\pm 100$ ms then
      - $[400, 600], [800, 1200], [1200, 1800], [1600, 2400], [2000, 3000]\,$
      - So, any value in the range $1200 – \text{max time}$ is valid
  - Example: MDE must be able to cope with hardware interface issues
Principle 9

- Academia must encourage us to solve the tough problems
  - Understanding our product is difficult
  - Think of how difficult it is sometimes to teach software design principles when we do not have time to get the students to solve complex enough problems that illustrate why we need those principles
  - Well, it is not much different for our research! Developing methods and tools that can handle industrial size and complexity cannot be achieved if we have time only to try the methods on smaller scale, somewhat idealized examples
  - We need to be able to harness the intellectual power of young faculty, not just faculty who have tenure and no worries about promotion
Principle 10

**EDUCATION**

- Our undergraduate software engineering courses need some re-focusing
  - Software Engineering should be like any other engineering discipline – there needs to be some focus on protecting the public from harm
  - Implication: More rigour (we do not have to call it formal methods); safety-critical and/or mission critical examples
- Professional Master’s in safety-critical systems (or medical devices or automotive or avionics or nuclear power or ..)
- A serious attempt at technology transfer – but we need convincing case studies and we need practical methods and tools. However, there are some successes already in this area – SMT solvers; theorem provers; static code analysis; model-checking – but most of these still have significant limitations
A Plea for Help

- This community has the ability/potential to make dramatic improvements in the dependability of software!
- The goals are simple – please keep them in mind.
- For any system, we need to demonstrate:
  - If built according to spec, the application will be successful (validation)
  - The application is built to spec (verification / correct by construction)
  - The application will “likely” deliver “safe” behaviour in the face of hardware malfunction or adverse effects in the environment (fail safe)
  - The application will continue to be correct and safe over its lifetime (maintainable!)
Conclusion

What about 5 years from now

- Comprehensive, integrated, practical methods and tools – where the methods and tools support a rigorous approach to critical software development .. and certification

- Clear progress in:
  - how to deal with complexity
  - how to determine, specify & validate required behaviour in a form readily accessible to system/software practitioners
  - how to generate refinements all the way to an implementation
  - how to evaluate dependability (safety, security, reliability)
  - how to gauge contributions of “coverage” and effectiveness of different verification techniques

- Better understanding of
  - computer science, software engineering, software development
  - … and the role of formal methods
References

THANK'S FOR ATTENTION