
ALTRAN
PRAXIS

TSP Symposium 2012

Delivering Agility and Discipline: Experiences with High-Assurance Software Engineering

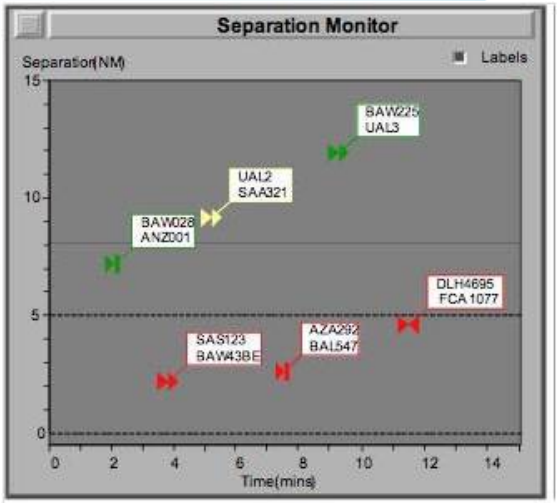
Rod Chapman

Contents

- High-Assurance Systems?
- Formal Methods – Why Bother?
- Combining TSP with Formal Approaches
- Other Sources of Inspiration
- A Future?

“High-Assurance” Systems?

- Military and Civil Aircraft – “fly by wire” systems
- High-Grade Secure Systems
- Jet Engine control and monitoring
- Air-Traffic Management



“High-Assurance” Systems?

- Compliance with industry and international standards.
- Often impose a lifecycle model, or objectives for particular forms of verification.
- Most are old – up to twenty years.
- Examples: DO-178B (aviation), CENELEC 50128 (Rail), Common Criteria (Secure Systems).

“High-Assurance” Systems?

- Regulation – need to satisfy some third-party that systems are fit for purpose.
- Significant pain is not managed well.

MINISTRY OF DEFENCE

“High-Assurance” Systems?

- Most are embedded, real-time, novel, possibly fault-tolerant…
  - About as hard as it gets!
- Significant potential for loss in case of failure, and pain in case of late defect and rework.
- Therefore, *even more incentive* for aggressive quality control during early lifecycle phases.

Correctness by Construction

- Since the early 1990s, we have developed an approach that became known as “Correctness by Construction”
- Main characteristics:
  - Strong process and “zero defect tolerance” culture.
  - Evidence-based assurance.
  - Static Verification (not just “test it lots…”)
- Use of “*Formal Methods*”

What's "Formal" Anyway? Here's a boring definition…

- The recently published DO-333 (Formal Methods Supplement of DO-178C) para FM.1.6.1 ("Formal Models") says…

  "…to be formal, a model should have an unambiguous, mathematically defined syntax and semantics."

- And FM.1.6.2 ("Formal Analysis") goes on to say

  "…an analysis method can only be regarded as formal analysis if its determination of property is sound. Sound analysis means that the method never asserts a property to be true when it is not true."

In “PSP English…”

- Maths
- Tools that yield 100% for well-defined defect classes

Every software project uses Formal Methods...

Professor Martyn Thomas CBE

Here's a formal language that you *all* use...

```
  0:   55
  1:   89 e5
  3:   56
  4:   53
  5:   8b 75 08
  8:   b9 05 b5 00 00
  d:   b8 00 00 00 00
 12:   8d 50 01
 15:   39 d1
 17:   74 1c
```

Here it is again...can you spot the bug?

```
 0: 55                    push   %ebp
 1: 89 e5                 mov    %esp,%ebp
 3: 56                    push   %esi
 4: 53                    push   %ebx
 5: 8b 75 08              mov    0x8(%ebp),%esi
 8: b9 05 b5 00 00        mov    $0xb505,%ecx
 d: b8 00 00 00 00        mov    $0x0,%eax
12: 8d 50 01              lea    0x1(%eax),%edx
15: 39 d1                 cmp    %edx,%ecx
17: 74 1c                 je     35
```

The big question is not *if* to use formal methods, but *when to start…*

So why bother with FM?

- Production of “a big pile of paper (with funny-looking math notation)” is *not the point.*

$$
\begin{array}{l}
\Delta IDStation; \Delta RealWorld \mid \\
\quad TISOpThenUpdate \\
\quad \land\ latch = locked \land latch' = unlocked \\
\vdash \\
\quad (\exists\, ValidToken \bullet goodT(\theta ValidToken) = currentUserToken \\
\qquad \land\ UserTokenOKNoCurrencyCheck \\
\qquad \land\ FingerOK) \\
\quad \lor \\
\quad (\exists\, TokenWithValidAuth \bullet goodT(\theta TokenWithValidAuth) = currentUserToken \\
\qquad \land\ UserTokenWithOKAuthCertNoCurrencyCheck) \\
\quad \lor \\
\quad (\exists\, ValidToken \bullet goodT(\theta ValidToken) = currentAdminToken \\
\qquad \land\ authCert \neq \emptyset \land (the\ authCert).role = guard)
\end{array}
$$

> See: *TISOpThenUpdate* (p. 5), *UserTokenOKNoCurrencyCheck* (p. 5), *UserTokenWithOKAuthCertNoCurrencyCheck* (p. 5)

So why bother with FM?

- The main point:
  - FM forces you to think really hard…
  - Availability of precise (and sound) analytical tools.

Thinking and tooling exposes…

Ambiguity…

Thinking and tooling exposes…

Contradiction…

Thinking and tooling exposes...

Incompleteness...

Thinking and tooling…

#include “customer conversation”;

Thinking and tooling…

- Why use tools? Why not just peer-review really really hard?
- Some problems are *just too hard* for the human brain, no matter how talented or numerous.
  - Example: concurrent software running on multi-core machines.
- Let's *complement* not replace humans.

Our story with TSP so far…

- How we got started…
- What we did (and didn't) do next…
- Where we are now…
- Some thoughts about the future…

How we got started…

How we got started…

- SEI and Praxis train each other…
- RCC uses SPARK and strong static verification in PSP training.
  - No surprise – it works!
- RCC becomes PSP Instructor…

Strong Static Verification? Eh?

- Also known as “static analysis” (of designs, code, whatever…)

- Such tools have a bad reputation. Why?
  - False Positives – “warning – your program might have a defect…”
  - False *Negatives* or *unsound* behaviour. Defects escape…
  - Slow

On Soundness…

- *Sound* tool says: “Your program definitely has *no* defects (of category X)…”
- *Unsound* tool says: “I’ve done my best, and I can’t find *any more* defects (of category X)…”
- Soundness gives you the confidence to deploy a tool *early* in your process…

On Soundness…

- BUT…soundness is very hard to achieve.

- We need *unambiguous* notations *early in the lifecycle* to analyse. These are very rare…
  - Almost all contemporary imperative high-level programming languages fail this test spectacularly..

Quiz Time!

- In C, what does this mean?

```
int i;
int a[10];
…
i = a[i++];
```

- Now imagine you're designing an automatic verification tool – what would you do?

Deploying static verification in PSP

- If you know a tool is
  - *sound* for a defect class, and
  - *so fast* that defect removal falls below your “shall I bother to log it” threshold.

then…

- Deploy that tool as early and as often as possible.
- Adjust later process steps and checklists to *ignore that defect class completely.*

Deploying static verification

- The catch….
- Deploying such languages and tools requires enormous discipline and determination – e.g. in choice of language subset and training.
- But…this ought to be a no-brainer for a mature organisation, right?!?!

What happened next…

- A *BIG* project called “iFACTS” comes along…
- Ramps up from 0 to >100 engineers in a short time.
- PSP rollout with one instructor? Ah well…

In the mean time…

- Start teaching Intro to Personal Process to *all* staff, including
  - HR
  - Finance team
  - Project managers
  - our SEPG…
  - anyone that will come!

Some unexpected results:

- We have “PP-erized” some business functions:
  - Parking management (I kid you not…)
  - HR Grade/Performance/Salary process
- Coming next…
  - Project Managers’ “month end scorecard” submission process.

- All have *painful* failure modes and cost of rework – just like software…

My knight in shining armour

- In the mean time, iFACTS has matured and re-launched, and has a new development team leader:

- James reads the PSP book
- Designs measures and begins to track effort against size of dev team work packages.
- Essentially – PROBE

Back to Formal Methods

- iFACTS uses the Z language for formal functional specification.

- Best correlation with implementation effort was found by measuring the *change of size of formal spec.*
  - Aha! Formal languages are *automatically countable*…

Back to Formal Methods

- Team data shows

$$£ = \beta_0 + \beta_1 * \square\square$$

- Note “lines of code” does not appear.

- □□is a measure of the *input* to the software development process, not the size of the resulting product.

Back to Formal Methods

- We're all used to "Regression Test", right?

- What about "Regression Proof"?

Back to Formal Methods

- The iFACTS software is subject to theorem-proving for “type safety” every night.
  - Changes are analysed in isolation before check-in.
  - Developers are *not allowed* to check in a code change that “breaks the proof”
- “Type Safety” = no exceptions, crashes, buffer overflow, etc. etc. *for any input data.*

Back to Formal Methods

- The analysis re-generates and proves about 136000 theorems from 226k logical sloc of SPARK code every day.
- Any failure is notified to all developers by email.
- The entire analysis can be reproduced in about 30 minutes on a modest workstation.

Back to Formal Methods

- Discipline eventually becomes habitual.

- For example: in PSP training assignment 4, there's a rather obvious division by zero…
  - I knew the darn tool would catch me out on this, so I spotted it, designed for it, wrote a test case… etc. etc.

- Side effect: my programming style in every other language is equally pedantic… ☺

Back to the plot…

- At long last, I train 6 engineers through PSP Fundamentals and Advanced.
- We also train up one more PSP Instructor…
- Then…corporate (parent company) requirement to gain CMMI ML3.

Back to the plot…

- This offers the opportunity to “inject” PSP thinking into many company process assets
  - Review process and forms
  - Measurement standards and planning
  - Project reporting standards
  - Estimating process, and so on…

- “PSP by Stealth…”

Pilot project…

- We are now running a pilot project with PSP.
  - Note – no full-blown TSP roll-out yet, since we don't have a TSP Coach.
  - We have to prove PSP works first…

Pilot project…

- Project is in its second iteration of software development.
  - Iteration 1 successful and yielded useful data for both individuals and the team.
  - Iteration 2 is on-going with a bigger team. More focus on team data and measurement goals this time.
  - Process combines PSP with test-driven design ideas. Test case design really is part of *design.*

Other sources of inspiration…

- “Extreme Programming” by Kent Beck.
- Read this in about 2002.
- Biggest surprise: how much XP stuff we were *already* doing…

Other sources of inspiration…

- “Lean Software Strategies” by Sutton and Middleton.
  - Significant and early work on application of Lean in Software Engineering.
  - Excellent data and story of how Lean and Formal Methods were applied to the production of the C130J Mission Systems.

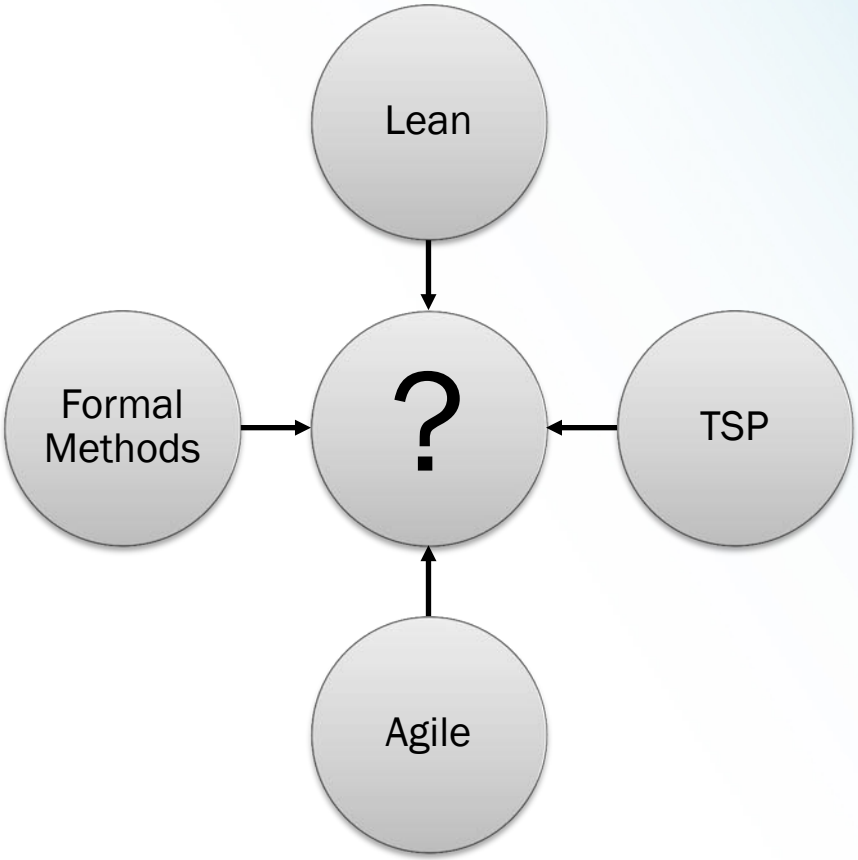
Other sources of inspiration…

- “Kanban” book by David J Anderson.
  - Full of great ideas about work-flow management for software and services teams.

Other sources of inspiration…

- "The Checklist Manifesto" by Atul Gawande
- *Absolutely wonderful* stories of effort to bring "checklist culture" into hospital surgical practice.
- Is this "TSP-for-medicine"???

A future? What happens if…



Homework…

- Try out a Formal Method
  - I suggest ALLOY from alloy.mit.edu
  - Good tutorials and example material

- Read the “Checklist Manifesto” book. Then give it to your boss. Repeat!