Static Verification and Extreme Programming

Rod Chapman

Praxis Critical Systems Limited

Contents

- XP and High-Integrity Systems?
- Pairwise Programming
- Refactoring
- Problems with Refactoring
- Other XP Practices
- Conclusions

XP and High-Integrity Systems?

- Surely a contradiction in terms? Perhaps not!
- Praxis have built (or helped to build) many safety- and security-critical systems over the years.
- 1999: Design and development of the MULTOS CA - an ultra-secure, distributed certification authority for smartcards.

XP and High-Integrity Systems?

- After building the CA: found Kent Beck's "Extreme Programming Explained" book
- Shock news: we were *already* doing many of the core XP practices.
- Many of the practices advocated by XP are actually the norm in the safety-critical industry.
  - e.g. use of coding standards and regular regression testing.

XP Practices

- Planning Game
- Small Releases
- Metaphor and "Stories"
- Simple Design
- Testing
- Refactoring
- Pairwise programming
- Collective ownership
- Continuous integration
- 40-hour week
- On-site customer
- Coding standards

XP Practices where we go further

- For 2 of the core practices, we go *further* than XP advocates. In particular, we use *static verification* to support these activities:

- Pairwise programming

- Refactoring

Pairwise programming

- XP idea: have 2 people sat at one workstation.
- One reviews and cross-checks the other's work.

Pairwise programming (2)

- My ideal pairwise partner:
  - Is *pedantic*. (never fails to spot a mistake)
  - Is *complementary.* (good at problems that I'm not good at.)
  - Is *efficient*. (spots mistakes right away…)
- Sounds like a tool to me!

Pairwise programming (3)

- Thesis: Me + SPARK Examiner = a very good pairwise programming team.
- Catch: This only works effectively where languages are *designed* for static verification.
  - *Very few* languages have this property
- SPARK allows us to "turn the dials up to 11."

Refactoring

- Fowler defines this as "...the process of changing a software system in such a way that it *does not alter the external behaviour* of the code yet improves its internal structure."

- XP community advocates regular and pedantic *regression test* to verify that external behaviour is not changed.

Refactoring (2)

- Hang on! Can we use static verification to verify a refactoring? Yes!

- If "property X" is true of a program before a refactoring, then X should also hold afterwards, where X might be
  - Language subset and semantics
  - No dataflow errors
  - Proof of exception freedom
  - …

Refactoring (3)

- The SPARK Examiner is written in SPARK.
- Over the years, we have refactored many large subsystems.
- We use static analysis *before* regression testing to check that properties are preserved.

Problems with refactoring

- We often find code that is in need of refactoring, but isn't...why not?
- Typically:
  - Projects don't plan to refactor, so there's never enough time...
  - Coverage analysis...
  - "Because it will break the tests..."
- We need more support for auto-generating and refactoring *tests* as well as code!

Refactoring guideline

- How about this:
- Refactoring is allowed only if:
  - Identical (or better) static verification results,
  - Identical test results,
  - Identical (or better) coverage analysis results.

Other XP Practices

- Most of the other core XP practices are familiar and common:
  - Coding standards - somewhat obviously!
  - Small releases
  - Continuous integration
  - Collective ownership

Other XP Practices

- One XP practice causes some concern: Simple Design
- XP Idea: design the simplest thing that will work now. Refactor later as (inevitably) requirements change. Avoid "BUFD".
- Problem: You can’t "refactor in" safety and security. (At least, it's very expensive if you try!)
- Idea: Do enough design up front to get safety and security architecture right from the start.

Conclusions

- XP practices are *already* used in high-integrity projects.
- Static Verification can complement and strengthen some XP practices, especially pairwise programming and refactoring.
  - Catch: effectiveness of SV critically depends on language.

Conclusions (2)

- Very few languages are *designed* for static verification: Ada, SPARK, Cyclone, Eiffel (to some extent), ?
- "Popular" languages defy deep, efficient SV, so the XP community haven't "got it" (yet...)
- Watch out for Java Modelling Language (JML) and the Extended Static Checker for Java 2, (JML ESC/Java2)
  - Serious danger of SV becoming fashionable!

Resources

- Me: rod.chapman@praxis-cs.co.uk
- SPARK: www.sparkada.com
- JML: www.jmlspecs.org