Formal Methods in Software Engineering

(Sensible Rigor)

Joe Kiniry
ITU & DemTech

R&D Methodology

1. find a problem that seems unsolvable and which most are frightened to tackle
2. develop new mathematics (mainly logic) to reason about the problem
3. develop a tool that demonstrates the new mathematics on Real World programs
4. eat our own dog food: use our own tools, and those of our colleagues around the world, to develop our own tools
5. PROFIT! (release as Open Source, goto 1)

Ethical Hacking

- our other main agenda is ethical hacking
- as a public employee and a scientist-activist, it is my duty to educate the public, politicians, industry, policy-makers, and the media about the risks and opportunities of digital technologies
- targets over the past several years include smart cards used for commerce and transportation, electronic voting systems, digital authentication systems, etc.

Election Software Examples

- election framework/platforms
  - KOA for kiosk-based and remote voting in the Dutch (list-based) voting system
- tallying systems
  - KOA tally system for the Netherlands
  - Votáil for Ireland
  - DIVS for Denmark
- voter registration, processing & ballot generation

Typical Process

- formal domain analysis using concept analysis and the BON specification language
- formal architecture specification using one or more specification languages like BON, JML, Z, Event-B, VDM, Alloy, TLA, etc.
- concurrent, parallel, and distributed system design using the CSP process calculus and UPPAAL
- formal reasoning about analysis and design using tools which reason about the above languages (incl. lightweight and semantic static analyses, protocol analysis, model finding and model checking, etc.)

Typical Process (II)

- automatic generation via refinement of formal architecture specification into annotated source code
  - annotations are written in JML, Code Contracts, or ACSL
- implementation is entirely done via Design by Contract
- static and dynamic analysis of code using around a dozen different technologies (lightweight and heavyweight static analysis, runtime assertion checking, model checking, etc.)

Typical Process (III)

- unit tests for all code are nearly all automatically generated from the architecture and the annotated code (typical statement coverage >>95%)
- only high-level subsystem tests are hand-written and are rigorously derived from high-level formal models of the system (e.g., ASMs, protocol descriptions, etc.)
- coverage and performance analysis and integration and deployment testing performed using FOSS and commercial tools (e.g., Emma, JProbe, JetBrains' many tools, Hudson, etc.)

Scope and Impact

- first year computer science students through experienced professors with decades of experience work together on these projects
- method and tools are taught at bachelor though postgraduate levels
- consultancy in industry for over a decade
- hundreds of archived example projects
- dozens of research papers published
- dozens of research software systems shipped
- several books being written