The Future is Ahead of Us:
Software Capability and Software Assurance

Bill Scherlis
SEI Principal Scientist
Professor of CS
Director, Inst Sw Research
STC 2013

function quicksort(a) =
  if (#a < 2) then a
  else
    let pivot = a[#a/2];
    lesser = {e in a | e < pivot};
    equal = {e in a | e == pivot};
    greater = {e in a | e > pivot};
    result = {quicksort(v): v in [lesser, greater]};
    in result[0] ++ equal ++ result[1];

We treat our software as a phenomenon of nature
— Sir Tony Hoare
Complexity
Predictability

A market experience story: Quality vs. Features?

Is this the correct model?

Outline

- Technical background
  - Early is better
  - Data intensive
  - Complexity
  - Value proposition
  - No plateau
  - Attributes and incrementality
  - Policy
- A useful analogy
- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability

The work of coding, I discovered, was an endless round of failure, failure, failure before eventual success. Computer-science students are used to failing. They do it all the time. It’s built into the process, and they take it in stride.

My computer-science colleague guest lectured in my class on composition, decomposition, and software. My computer-science students do the same process in code.

Outline

- Technical background
  - Early is better
  - Data intensive
  - Complexity
  - Value proposition
  - No plateau
  - Attributes and incrementality
  - Policy
- A useful analogy
- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability

Automated infrastructure for builds, tests, and analytics
A simple example: Automated performance tests

Traceability in current practice: Accountability for every line of code, accomplished automatically by advanced tools.

Tools can automatically provide accountability for every increment of change

Tools manage and track issue-focused data and communication
Outline

- Technical background
  - Early is better
  - Data intensive
  - Complexity
  - Value proposition
  - No plateau
  - Attributes and incrementality
  - Policy
- A useful analogy
- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability

Disruption: Clock speed tops out

Overclocking (according to Tom’s Hardware)

- Pentium 4 at 5 GHz (2.5x baseline) – 96 amps
  - Cooling for 180 watts (i.e., 1 megawatt / m² chip area)
    - Liquid nitrogen
    - Compressor
- How much increase in computing performance?

Rich supply chains for IT

- Rapid growth in scale, complexity, geography in the sourcing of systems
Sourcing and ecosystems

Consequence: Data centers and distributed computing

- Architectural convergence
  - Large-scale data centers
    - Buy "cheap flaky machines in preference to expensive (also flaky) machines." (Google)
    - Major data center cost is hardware depreciation
  - Scientific supercomputing
    - Ultra-fast processor with low MTBF → distributed redundant with incremental self-
  - Desktops, mobile devices, etc.
    - Multiple processors

- Key to these models
  - Responsibility for reliability: from hardware to software
    - "Write better software" (John Wilkes, Google cluster mgmt)

Outline

- Technical background
  - Early is better
  - Data intensive
  - Complexity
  - Value proposition
  - No plateau
  - Attributes and incrementality
  - Policy

- A useful analogy

- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability

Software growth

From: dirkriehle.com

Linux
Human capacity and software code

- Software capability is driven by technology development
  - Practice, process, measurement
  - Tools
  - Languages
  - Models and analysis
  - Socio-technical ecosystems and software frameworks / libraries
  - Hardware and systems infrastructure

\[ L + M + A \rightarrow L' \]
Capability, interlinking, agility

Critical engineering building material

Outline

- Technical background
  - Early is better
  - Data intensive
  - Complexity
  - Value proposition
  - No plateau
  - Attributes and incrementality
  - Policy

- A useful analogy

- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability

Automatic programming

Software may be the single largest asset type in the US economy. NRC STEP 2006
The software routinization cycle

Patterns and metaphors

Ad hoc solution sets

Increased leverage

Routines

Higher level problems

Automation

Adapted from Mary Shaw

"Successful IT investments initiate a virtuous cycle of investment and gain"
Difficult-to-assure dependability attributes

- Safe concurrency
  - Races and memory model
    - Lock based
    - Thread confined
    - Data races
  - Deadlocks
  - Real-time thread/memory
- Information flows
  - Aliases, references, effects
  - Security attributes
  - Encapsulation, overlay abstractions
- Code safety
  - Appropriate typing
- Policy compliance
  - API policy compliance
  - Framework patterns
  - Protocol compliance
  - Architectural compliance
  - Object references and aliasing
  - Modularity for composition

Performing errors

- A published (Mathematica) version of binary search:
  
  ```mathematica
  BinarySearch[l_List, k_Integer, low_Integer, high_Integer, f_] :=
  Module[{mid = Floor[(low + high)/2]},
  If[(low > high, Return[low - 1/2]);
  If[(f[l[[mid]]]) == k, Return[mid];
  If[(f[l[[mid]]]) > k,
  BinarySearch[l, k, 1, mid - 1, f],
  BinarySearch[l, k, mid + 1, high, f]]]
  ```

- This library code contains a flaw that was not detected for five years.

- Why does this matter?

Automation and tools

- Precise and focused modeling and direct analysis of code and executions can support verification
Outline

• Technical background
  – Early is better
  – Data intensive
  – Complexity
  – Value proposition
  – No plateau
  – Attributes and incrementality
  – Policy

• A useful analogy

• Three emerging technical enablers
  – Composition
  – Deep metrics
  – Traceability

Business environment

• Multiple existing standards for evaluation, with common issues
  – Snapshot
  – Composability
  – Recertification
  – Cost and delay

• Mandate
  – Congressional language
  – DSB’07
    – Failure to maintain a lead in the ability to prevent and evaluate confers advantage to adversaries
  – NITRD DIS
    – Designed-In Security

Building Code: Idealized

Five salient features
(1) Engineering constraints
(2) Predicted quality outcomes
(3) Visible evidence of quality
(4) Explicit support for response
(5) Continuous evolution

Consensus and compromise
(1) Enable innovation
(2) Protect IP
(3) Limit impacts on cost, performance, schedule, quality
(4) Fairly allocate risk and responsibility
(5) Afford measurement and visibility of risk and cost

They exist.
They work.

Adapted from NSF SaTc, Nov 2012 (inspired by Carl Landwehr and WGS)
Building Code: Accommodations and Possibilities

(1) Fast pace of technology and ecosystem advancement
   - More goals (what); less mechanism (how)
   - Require a positive case with concrete evidence

(2) Scale, interconnection, customization unlike physical systems
   - Composition is key

(3) Diversity and inter-relatedness of quality attributes
   - Build models, analyses, metrics, composition for each
   - Combine quality and security attributes – breakage and threats

(4) Economics and measurement as fundamental drivers
   - Address incentives in building code – from EVM to IDE
   - Fairly allocate risk mitigation benefit

Outline

- We are entering a period of great opportunity and possibility
  - The policy environment is in a greater state of readiness
  - The technical raw materials are at hand.
  - We are able to gather and manage the wealth of necessary data
  - We can provide the traceability and ongoing granular CM

- The building code analogy remains apt
  - Emphasize evidence and artifact-focused evaluation
  - Divide and conquer: attributes, increments, composition
  - Base on economics, incentives, and measurement

- Three emerging technical enablers
  - Composition
  - Deep metrics
  - Traceability