Toward Evidence-Based Low-Defect Software Production

James Kirby Jr.*
james.kirby@nrl.navy.mil
US Naval Research Laboratory
Center for High Assurance Computer Systems
Tuesday 12 July 2016

*With participation by David Weiss.
Our Big Bet on Computer Software
Critical building material of early 21st Century

- Defense, Government, economy
- 90% recent military aircraft functionality provided by software
- Autos, phones, cameras, etc. are computers driven by complex software embedded with sensors & actuators
- Much National Critical Infrastructure is software-intensive
- Numerous software-intensive Government initiatives
  - Health IT, Connected Cities, Machine Learning & AI
  - 3rd Strategic theme of National Strategic Computing Initiative (NSCI): improving software productivity
- HPC software is critical national infrastructure
  - Operational life measured in decades
  - Longer-lived than hardware on which it runs
  - More valuable than hardware on which it runs

Sectors employing more than 50,000 software developers

<table>
<thead>
<tr>
<th>Industry Sectors</th>
<th>Developers (thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturing</td>
<td>147.9</td>
</tr>
<tr>
<td>Wholesale Trade</td>
<td>59.5</td>
</tr>
<tr>
<td>Information</td>
<td>175.2</td>
</tr>
<tr>
<td>Finance &amp; Insurance</td>
<td>99.2</td>
</tr>
<tr>
<td>Professional, Scientific, Technical Services</td>
<td>530.3</td>
</tr>
<tr>
<td>Management of Companies &amp; Enterprises</td>
<td>54.9</td>
</tr>
</tbody>
</table>

Software Production

• Software development
  – Starting fresh or reusing existing software

• Software sustainment
  – Evolving software throughout development and operational life as needs, understanding, technology, and infrastructure *inevitably evolve*

• Software assurance
  – Developing confidence that *evolving software* continues to exhibit critical properties

[Weiss, Kirby, Lutz 2013]
Unsatisfactory Software Production Technology

• Developers and users **unable** to develop and sustain in timely and affordable manner software exhibiting **low defect rates**
  – Significant obstacle to **Cybersecurity** success
  – Avoiding, mitigating errors **significant drag** on economy
  – Industry **unlikely to adopt** slow expensive technology

• Cybersecurity requires invalidating this common wisdom:

"Perfect" (bug-free) software is impractically expensive and slow to produce, and so the vast bulk of consumer and enterprise software products are shipped when they are "good enough" but far from bug-free. As a consequence, there has been a constant struggle to keep attackers from exploiting these **chronically inevitable** bugs.

Production of Software-defined Systems

- Software-defined Radio, Software-defined Networks, etc.
  - Software intensive systems
- Domain Experts (engineers) develop models, specifications
  - MATLAB, Python
- Software developers hand code in C/C++ to the models, specifications
  - Have minimal domain knowledge
  - Receive little oversight by domain experts

Consequences

- Description of complex software behavior *twice developed by hand*
  - Opportunity to *insert delay, effort, and error* into software production
- Engineers don’t have direct oversight on their delivered products
  - Engineers unexpert in implementation technology
  - Programmers unexpert in domain science/engineering
- Retrofit in response to competitive innovation takes years to decades
  - Competitors innovating more quickly than we can respond
- Some legacy systems cannot be upgraded
  - Unable to run ever-evolving development tools

1 Contributed by Dr. Ramesh Bharadwaj, NRL
Strawman Software Production

- **Software development**
  - Requirements may be developed
    - May record *software behavior* to some level of detail
  - Design may be developed
    - May record *software behavior*
    - May record *software architecture* required to accomplish software behavior
  - Code is developed by hand
    - Records *how to accomplish* complex software behavior precisely and completely
      - What and why left as an exercise for the reader
    - *Evolves* as understanding, needs, infrastructure, technology change
      - *May invalidate requirements, design* which are often not evolved

- **Software sustainment**
  - *Code evolves* as understanding, needs, infrastructure, technology change
  - Requirements and design *ignored*
    - Code itself and other developers are only reliable sources of information

- **Software assurance**
  - Develop confidence that *evolving software* continues to exhibit critical properties *without reliance* on out-of-date requirements and design
Alternative Production Technologies

• We need improved technology enabling software developers and users to develop and sustain low-defect software in a timely and affordable manner
  – Technology includes tools, techniques, know how
  – Users include domain scientists, engineers, other subject matter experts

English Wheel and Moal Coachbuilders factory and products (moal.com)

Mechanical Press and Hyundai Assembly Line
Alternative Production Technologies  
Replace labor-intensive hand coding

• Software Product Line Engineering [Campbell 2008] [Weiss and Lai 1999]
  – Software family constructed such that desired members can be quickly produced
  – Decisions distinguish members of family of related software systems
  – Developers and users resolve decisions to produce desired member

• Model-Driven Development [Kirby 2006, 2013]
  – Developers and users create models from which computer programs are generated
  – Model specifies software behavior for requirements, design, implementation

• Synthesis Formal Methods [Alur et al 2015]
  – Developers and users develop domain-specific high-level description of desired behavior
  – Synthesis generates correct-by-construction implementation

• Program Transformations [Baxter and Mehlich 1997]
  – Developers and users guide selection of transformations of formal design to produce correct-by-construction code
    • Comprise design decisions implicitly used by current software developers
    • Produce complete design and its rationale
Evidence-based Approach to Improving Software Production Technology

• Goal/Question/Metric (GQM) [Basili 1993]
  – Define software production measurement in top-down fashion based on goals
• Identify **goals** of software production
• Refine goals into set of **quantifiable questions**
• Questions imply **metrics** that guide data collection
• Collected data provides **evidence-based view**
Goals of Software Production

• **Reduce defect rate** of developed and sustained software

• **Reduce time** to develop, sustain, and assure software

• **Reduce effort** to develop, sustain, and assure software

• **Widespread insertion** of improved software production technology
Evaluating Software Production 1
Goals and Questions

• **Reduce defect rate** of developed and sustained software
  – What is software defect rate? Where inserted? Removed?
  – What production factors contribute to software defect rate?
  – What knowledge is crucial to sustaining low-defect software?

• **Reduce time** to develop, sustain, and assure software
  – How much time is required to develop, sustain, and assure software?
  – How is this time spent? Can we detect wasted time?

• **Reduce effort** to develop, sustain, and assure software
  – How much effort is required to develop, sustain, assure software?
  – How is this effort spent? Can we detect wasted, duplicate effort?
  – How does a software production effort make and remember its decisions and assumptions? [Hutchins 1995][Aranda and Easterbrook 2006]
Evaluating Software Production 2
Goals and Questions

• *Widespread insertion* of improved software production technology
  – What technology are developers and users using?
  – What national investment is required to insert new technology?
    • Is technology usable by existing software developers and users?
    • What software tools, education, training, expertise required?
    • How well-suited is technology for insertion?
      – Defense, Government, many software-dependent sectors of US economy
  – What software tools are available to support new technology?
    • What computing resources do they require?
    • Do software tools support inevitably evolving software?
    • Do software tools scale?
    • How are software tools sustained?
    • Is there a healthy market for software tools?
Software is Critical Building Material

• Software defects
  – Important source of software vulnerabilities
  – Significant drag on US economy
  – Avoiding and mitigating them waste perhaps 1% GDP

• Software production technology that enables timely, affordable production of low-defect software more likely to be adopted
  – Reducing software defects
  – Making government, Defense, & economy more agile, innovative, competitive
Bibliography

• Aranda and Easterbrook. "Distributed cognition in software engineering research: Can it be made to work?." *Supporting the Social Side of Large Scale Software Development* (2006): 35.
Questions?