Embedded System Software Quality

Why is it so often terrible? What can we do about it?

Prof. Philip Koopman

ISSRE 2016 Keynote
**Overview: What I’ve Learned in 200 Reviews**

- **Embedded software failures are on the rise**
  - Computers go far beyond “Internet of Things”
  - Slapped together source code isn’t good enough
  - Security, safety, critical infrastructure all matter

- **The usual suspects won’t solve this problem**
  - Better process, more testing, formal methods, ...

- **The fundamental problem:**
  - Poor software development literacy
  - Code is not written by “computer” people
  - Managers don’t get why software is difficult
  - Low probability failures * large fleet exposure

- And ... machine learning makes things harder!
Bad Code Shows Up Everywhere

(1991) Patriot Missile Software
Defect / 28 Americans Dead
Workaround: frequent reboots

Therac 25: 1985-1987
6+ radiation overdoses

FDA Recall Data 2002-2012

To keep a Boeing Dreamliner flying, reboot once every 248 days

Knight Capital Says Trading Glitch Cost It $440 Million
Runaway Trades Spread Turmoil Across Wall St.

Repurposed Bit activates testing mode
Bad Code Is A Leading Security Problem

Attacks can affect the physical world

Hack attack causes 'massive damage' at steel works

https://goo.gl/CDsbV2

Attacks Against SCADA Systems Doubled in 2014: Dell

By Mike Lennon on April 13, 2015

Dell SonicWALL saw global SCADA attacks increase against its customer base from 91,676 in January 2012 to 163,228 in January 2013, and 675,186 in January 2014.

https://goo.gl/24Jp7j

Key SCADA Attack Methods

- 26% Buffer Overflow or Similar
- 6% Crypto Issues
- and other categories...

© 2016 Edge Case Research LLC
Security problems often due to bad fundamentals

Is It That Hard To Get Basics Right?

No password & no signature for externally pushed firewall update

It’s a buffer overflow bug:

```c
memcpy(bp, pl, payload);
```
Automotive Engine Control Code Quality

- $$$B Toyota Unintended Acceleration Fiasco: 2010-...?
  - “Reckless disregard” Jury verdict in one death
  - Hundreds of death and injury settlements (still ongoing)

- Code quality overview (256K SLOC in C):
  - Throttle angle routine: MCC of 146 (MCC above 50 is “untestable”)
  - 80,000 MISRA C violations
    - Uninitialized variables, condition side effects, unsafe type casting
    - 2272 global variable declaration type mismatches
  - 10,000 read/write global variables
  - Missing concurrency locks; confirmed race condition bug
  - Recursion with essentially full stack; no stack protection
  - Ineffective watchdog (kicked from HW timer)
  - No configuration management, no bug tracking, real time scheduling overload, minimal peer reviews

For more, web search: “Koopman Toyota”
Is Bad Software Eating The World?

SOFTWARE NOW TO BLAME FOR 15 PERCENT OF CAR RECALLS
YOU CAN’T JUST HOLD THE HOME AND LOCK BUTTONS TO SOLVE THIS ONE  June 2, 2016

The research firm J.D. Power, through its Safety IQ application, found that there have been 189 distinct software recalls issued over five years—covering more than 13 million vehicles. These weren't merely interface-related issues either; 141 of these presented a higher risk of crashing.

Software can make or break a company
- So why is it still so often bad?

Don’t we have tools for that?
- Automated testing, build tools
- Static analysis tools
- Formal methods
- Better development processes
- Model-based design
- CMMI
- ...
Who's Writing All This Code?

Stack Overflow survey: Nearly half of developers are self-taught

“There are many ways to learn how to code. Forty-eight percent of respondents never received a degree in computer science, 33 percent of respondents never took a computer science university course,” the report said. “System administrators are most likely to be self-taught (52 percent). Enterprise level services developers are most likely to have an industry certification (13 percent).”

- In embedded systems:
  - Most are domain specialists (Mech. Eng., Industrial Eng., etc.)
  - Older workers are self-taught
  - Younger workers took intro programming

Who pays more for computer grads: Silicon Valley or an industrial controls device manufacturer?
How Did We Get Here?

- Simple answer: one bad line of code at a time
  - Emphasis on **coding**, not **engineering software**

- Deeper answer:
  - Gaps in technical skills
  - Gaps in process skills
  - Gaps in management understanding
  - Gaps in education

- **Gerry Weinberg's Second Law:**
  - If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization
Coding Is Essentially 0% of Creating Software

2013 Embedded Market Study

What percentage of your design time is spent on each of the following stages?

- Developing overall system specs: 14% (2013), 15% (2012), 15% (2011)
- Conceptual design stage: 11% (2013), 11% (2012), 12% (2011)
- Detailed design stage: 12% (2013), 22% (2012), 22% (2011)
- Simulation stage: 7% (2013), 8% (2012), 8% (2011)
- Prototyping: 10% (2013), 12% (2012), 12% (2011)
- Sending to production: 6% (2013), 6% (2012), 6% (2011)
- Documentation/coding/mtgs: 1% (2013), 2% (2012), 2% (2011)

http://e.ubmelectronics.com/2013EmbeddedStudy/index.html

Copyright © 2013 by UBM. All rights reserved.
Staffing Requires 1:1 Designers To Testers

IEC 60730 Appliance Safety
Is A “Computer” Degree Enough?

- Mechanical engineers (etc.) have intro software
  - But typically no deep training in engineering software-based systems
- How many CompEng/CS students have deep software engineering skills?
  - Well written requirements, change tracking, traceability
  - Rigorous approaches to testing & peer reviews
  - Really understand all the boxes for V, Scrum, or whatever?
Example Technical Skills

- **Basic skills**
  - Programming, A/D, D/A, math, power

- **Software Construction**
  - Modularity, complexity, style

- **Real Time Software**
  - Interrupts, scheduling, concurrency, time

- **Robust System Design**
  - Watchdog, stack overflow, exceptions

- **Validation**
  - Test strategies, coverage, traceability
  - Peer reviews, lightweight formal methods

---

How a dumb software glitch kept thousands from reaching 911

By Brian Fung  October 20, 2014  https://goo.gl/0d5ANZ

But on April 9, the software responsible for assigning the codes maxed out at a pre-set limit; the counter literally stopped counting at 40 million calls. As a result, the routing system stopped accepting new calls, leading to a bottleneck and a series of cascading failures elsewhere in the 911 infrastructure.

© 2016 Edge Case Research LLC
Example Process Skills

**Development process**
- Process models, artifacts, metrics
- Requirements & estimation
- Architecture, software design, test design

**Deployment process**
- Configuration management
- Version control
- Defect tracking and resolution (software & process defects)
- Building, deployment, field defects
Example Specialty Skills

**Safety**
- Critical system design
- Safety plan
- Safety standards
- Redundancy

**Security**
- Secure system design
- Security plan & best practices
- Cryptography, security protocols, key management
- Patch management and secure boot
Example Management Skills

Software Quality Challenges
- Process & technical quality
- Technical debt management
- Validation: testing and beyond
- Role of SQA/PPQA

Software Project Management
- Software process models
- Requirements management
- Estimation, Mythical Man Month
- External safety & security certification

MyFord Touch problems: Ford to issue upgrade
Glitches in MyFord Touch software that replaces knobs and buttons with a touchscreen have led to plummeting user approval ratings for Ford cars

© 2016 Edge Case Research LLC
Why Managing Software Costs Is Difficult

Product costs are managed by

BOM = Bill Of Materials

Mechanical System
90% of BOM cost
Mostly commodity

Electronic Controller
10% of BOM cost
Mostly commodity

Software
0% of BOM cost
90% of product differentiation

© 2016 Edge Case Research LLC
FALSE: A “mostly working,” undisciplined prototype can be deployed
- It’s more expensive to fix a mess than to take a step back and do it right

FALSE: Getting compiled code quickly indicates progress
- Skimping on up-front design and review creates bug farms

FALSE: Testing improves software quality
- It mostly removes the “easy” and frequent bugs
- Product testing won’t find subtle timing bugs and edge cases

FALSE: Peer review is expensive
- It’s perhaps 10% of project budget to find 50%-70% of bugs
- “Sorry, no time to peer review” – they’re too busy writing bugs

FALSE: Software development is quick & cheap; anyone can write code
- It takes 1-2 SLOC/person-hr if you want good embedded code
- Fixed end date + inflexible requirements = project failure
Deploying SW Competence Upgrades

- Embedded systems is about breadth...
  - computer skills + domain skills
  - Computer experts need domain literacy
    - Thermodynamics, materials, statics & dynamics, ...
  - Domain experts need software literacy
    - Teaching “how to code” will (mostly) just get you bad software

- Mid-career skill upgrade delivery paths:
  - Self-instruction, on-line training, etc.
  - Project reviews with hands-on mentoring of fixing problems
  - How about some books that cover all this important stuff?

Coriolis Flow Meter (actual size)
And Now ... Self-Driving Cars

- **6/2016: Tesla “AutoPilot” Crash**
  - Claimed first crash after 130M miles of operation

![Diagram of crash scene]

1. **Trailer turns left** in front of the Tesla.
2. **Tesla doesn’t stop**, hitting the trailer and traveling under it.
3. **Tesla veers off road** and strikes two fences and a power pole.
Need to test for at least ~3x crash rate to validate safety

- Hypothetical deployment: NYC Medallion Taxi Fleet
- 13,437 vehicles @ 70,000 miles/yr = 941M miles/year
- 7 critical crashes in 2015
  - 134M miles/critical crash (death or serious injury)

How much testing to validate critical crash rate?

- Answer: 3x – ~10x the mean crash rate
  - Probably you’ll get a crash ➔ even more testing
- These are optimistic test lengths...
- Assumes random independent arrivals
- Is simulated driving accurate enough?

<table>
<thead>
<tr>
<th>Testing Miles</th>
<th>Confidence if NO critical crash seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>122.8M</td>
<td>60%</td>
</tr>
<tr>
<td>308.5M</td>
<td>90%</td>
</tr>
<tr>
<td>401.4M</td>
<td>95%</td>
</tr>
<tr>
<td>617.1M</td>
<td>99%</td>
</tr>
</tbody>
</table>
How Do You Validate Machine Learning?

- **Self-driving cars are so cool!**
  - But also kind of scary

- **These need all those skills**
  - Good technical skills
  - Good process skills
  - Safety & security
  - Management who “gets it”
  - **AND:** Validating Machine Learning
NREC: 30 Years Of Cool Robots

Carnegie Mellon University Faculty, staff, students
Off-campus Robotics Institute facility
Validating Robots Is Difficult

- Even the “easy” (well known) cases are challenging

- Extreme contrast
- No markings
- Poor visibility
- Unusual obstacles
- Construction
- Water (appears flat!)

[Wagner 2014]
The Tough Cases Are Legion

http://piximus.net/fun/funny-and-odd-things-spotted-on-the-road
http://edtech2.boisestate.edu/robertsona/506/images/buffalo.jpg
Deep Learning Can Be Brittle & Inscrutable

QuocNet:

Car  Not a Car  Magnified Difference

AlexNet:

Bus  Magnified Difference  Not a Bus

Where are Machine Learning Requirements?
- V model traces reqts to V&V
- ML system is just a framework
- The training data forms de facto requirements

Is training data “complete” and correct?
- Training data is safety critical
- What if a moderately rare case isn’t trained?
  - It might not behave as you expect
  - People’s perception of “almost the same” does not necessarily predict ML responses!
Automated Stress-Testing for Autonomy Architectures

Test Specification and Execution Overview

- **Distribution:** A – NREC case number STAA-2013-10-02

**Existing Documentation for SUT:**
- Message Dictionary (ICD, IDD)
- System Requirements (SRS)
- Safety Requirements

**ASTAA Test Spec (XML):**
- Interface Definition
  - Ports & Protocols
- Message Dictionary
  - Message Frame 1
    - Parameter 1 (int)
    - Parameter 2 (bounded float)
  - Message Frame 2
    - Parameter 1 (float)
    - Parameter 2 (speed)

**Constructors & Destructors:**
- Invariants Definition
  - Mode State Machines
    - Invariant List
      - Inv1: Param1 <= PLimit1
      - Inv2: <Condition>
  - Invariant Failure Destructors

**Test Generator:**
- Test Cases (XML)
  - Test Command Sequences
    - Type Exceptions
    - Bounded int, bounded float
    - Speed, Curvature

**Exceptions Database:**
- Invariants Definition
- Constructors
- Test Command Sequence
- Destructors
- Interface Definition

**Test Case (XML):**
- Invariants Definition
- Constructors
- Test Command Sequence
- Destructors
- Interface Definition

**ASTAA Test Runner:**
- Invariant Monitors
- Test Injectors
- Orchestrator
- Logger

**Module Manager:**
- Protocol Module

**System Under Test:**
- Safety Monitor (optional)

**Define ASTAA Test Specification** (guided manual process)

**Execute Test Generator** (automated process)

**Execute Test Cases with Test Runner** (automated process)
Some Autonomy Problems We’ve Found

- **Improper handling of floating-point numbers:**
  - Inf, NaN, limited precision

- **Array indexing and allocation:**
  - Images, point clouds, etc...
  - Segmentation faults due to arrays that are too small
  - Many forms of buffer overflow with complex data types
  - Large arrays and memory exhaustion

- **Time:**
  - Time flowing backwards, jumps
  - Not rejecting stale data

- **Problems handling dynamic state:**
  - For example, lists of perceived objects or command trajectories
  - Race conditions permit improper insertion or removal of items
  - Garbage collection causes crashes or hangs
Good job of advocating for progress
- E.g., recommends ISO 26262 safety standard

But, some areas need work, such as:
- Machine Learning validation is immature
- Need independent safety assessment
- Re-certification for all updates to critical components
- Which long-tail exceptional situations are “reasonable” to mitigate
- Crash data recorders must have forensic validity
Conclusions

- **Challenge: Software Development Literacy**
  - Critical skill shortage in engineering software
    - Learning to code doesn’t fix this alone
  - Long-term need to upgrade workforce skills
    - Teaching this in college won’t fix this alone
  - Tools and such only help if basic skills in place

- **Machine Learning ups the ante**
  - Validating ML training is challenging
  - Workforce knows even less about ML than about engineering software!

- **And yet, our society revolves around the quality of all this software**
  - A small step: http://www.edge-case-research.com/videos/
Specialists in Making Robust Software:

- Embedded software quality assessment and improvement
- Embedded software skill evaluation and training
- Lightweight software processes for mission critical systems
- Lightweight automation for process & technical area improvement

info@edge-case-research.com
http://www.edge-case-research.com/