Some Independent Software Verification Techniques Used within the FDA Regulatory Context

Brian Fitzgerald
I’m not a policy person at FDA. I’m part of a group of software engineers who review premarket submissions, participate in quality system inspections, and analyze/investigate medical device problems involving software design and performance. I have a technical focus, not a regulatory focus.
What we will cover in 45 minutes

♦ Some background on the problem
♦ The prior state of our capability
♦ The current state of our capability
♦ The near future state of our capability

♦ Remember! we regulate medical devices. So your mileage might vary!
♦ FDA rarely sets mandatory standards
Background

- FDA – CDRH regulates medical devices, their component software and software which is a medical device under FDCA.
- Two principle avenues (among many others.)
  - Premarket reviews
    - Reasonably safe (class 3)
    - Substantially equivalent (class 1 & 2)
  - Postmarket surveillance
    - Adverse event reports from users
    - Adverse event reports from manufacturer
    - Adverse event reports from Medsun
    - Snitch!!

This is a gross over simplification of the law but is intended to provide a high level overview to the non medical device industry of the US regulatory scheme.

If you want more details please contact DSMICA website or go to http://www.fda.gov/cdrh/devadvice/ for much more comprehensive details.
1) The complexity of such an analysis increases approximately in proportion to the number of states in the software state machine. Think $2^{20}$. Or more.

2) Environment is used here in the traditional Systems Engineering sense where there is the ‘System’, the ‘Environment’ and the ‘Universe’. The ability of a product (software) designer to predict and thereby control risks in the product is reduced as sources of hazards migrate from the System outward to the Environment and finally out to the Universe. The context of system use by real people is covered by the discipline of Human factors engineering where elements of psychology, situation and essential performance are coordinated into the failure analysis. Unfortunately the software industry has a poor track record of using Human Factors along side its own traditional process controls.

We frequently speculate as to where the answer to the “poor quality of software” problem could be found hiding. Is it under the complexity or under the environment?

If we can reduce as far as possible the latent errors due to complexity we will see!
1) By functional I refer to case based tests, where a quality professional evaluates a reasonably comprehensive finite verification regime designed to trap the most obvious implementation errors.

2) Often called “unit testing”.

3) There are often just too many potential permutations and combinations of system states unless rigorous coding standards are set. Efforts have been made in this regard to define a useful yet restricted C code guide such as MISRA.

4) The traditional compile-and-link-then-test engineering approaches often suffer from this effect.
FDA’s definitions of verification and validation can be found in 21CRF820 and are broadly consistent with the accepted Quality system definitions originating from ISO9000 but differ in subtle ways to that often found in the traditional software industry’s jargon.

Background [more]

♦ Maybe a little easier to validate than verify!
   ♦ By defining the means, methods and records needed to validate the software it has been possible to introduce the concept of an acceptable level of due diligence in the engineering practices typically found in the software sector.

♦ But it hasn’t helped enough! There are still too many device failures related to software design flaws. There is no guarantee that a good process will produce a good product!

7
Quality management systems have been based on ISO 9000 for 40 years and in the medical industry ISO 13485 represents its specific instantiation.

All devices containing software are to be designed under the QSR design controls which involve verification and validation activities.

Recent advances in software verification tools and techniques may complement and leverage the traditional “mainly validation” approach by allowing close scrutiny of this “due diligence”.

1) Quality management systems have been based on ISO 9000 for 40 years and in the medical industry ISO 13485 represents its specific instantiation.

2) See 21 CFR 820 et seq
Reliance on/Acceptance of standards has been building

- A solid base of consensus standards has emerged across the global medical device marketplace in recent years which broadly harmonize the details of the due diligence approach.

<table>
<thead>
<tr>
<th>Process focus</th>
<th>Product focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEC 60601-1-4</td>
<td>3rd edition of IEC 60601-1</td>
</tr>
<tr>
<td>AAMI SW88</td>
<td></td>
</tr>
<tr>
<td>IEC 62304</td>
<td></td>
</tr>
<tr>
<td>AAMI TIR 36, 32</td>
<td></td>
</tr>
<tr>
<td>IEC 80001 and IEC 80002</td>
<td></td>
</tr>
</tbody>
</table>
The then Office of Science and Technology (OST) conducted several manual code reviews in collaboration with ORA.

- Previous attempts at verification
  - 15 years ago manual code reviews were conducted by FDA and led to precedent setting compliance actions
  - They were reserved for extraordinary situations because the resources needed were very demanding
1) FDA has obtained a judicial order for seizure and a consent decree of a ventilator device in 1992. Code review was one of the tools in that case.
Prior State of Capability

♦ Just awful!
  ♦ Manual code review from line printer sent by certified mail
  ♦ Little documentation or cooperation
  ♦ Long table and blackboard
  ♦ Guesswork, Coffee, Luck
  ♦ Not very different from private sector acquisition scenario except no collaboration
Current Capability

- Recent advances in Static Analysis and program slicing have brought the tracing of error propagation a long way from the ‘Lint’ of old.

- Now it is routinely possible to traverse the codebase from a modeling perspective.

- Semantic errors in related routines can be modeled to see if they could result in one of a defined set of recognizable unsafe states of program execution.
Current Capability

- Static Analysis
  - Not your grandfather’s static analysis!
  - Creates a large abstract database of the state-machine as defined by the source code and shows the set of feasible logical contingencies for how the code can be traversed during symbolic execution and what values the variables, pointers, etc. will have in those contingent pathways.
  - Unsafe memory states are identified which could emerge during runtime and their root cause is traced back in the code.
Semantic errors in implementation can be modeled together to see if they could result in one of a defined set of recognizable unsafe states of program execution.

Examples include:
Buffer Overrun, Buffer Underrun, Type Overrun, Type Underrun, Null Pointer Dereference, Divide By Zero, Double Free Use After Free, Free Non-Heap Variable, Uninitialized Variable, Leak, Dangerous Function Cast, Delete[ ] Object Created by malloc, Delete[ ] Object Created by new

**Buffer Overrun**
A read or write to data after the end of a buffer.
http://en.wikipedia.org/wiki/Buffer_overflow
Writing a piece of data to a specific location designated as only being reserved for a shorter length of that data. A variant of run-time data corruption.

**Buffer Underrun**
A read or write to data before the beginning of a buffer.
http://en.wikipedia.org/wiki/Buffer_underrun
Reading a piece of data from just before a specific location designated as only being reserved. A variant of run-time data corruption.

**Type Overrun**
An overrun of a boundary within an aggregate type.
Writing a piece of data of a specific (usually computed) byte length to a location designated as only being reserved for a shorter length. A variant of run-time data corruption.

**Type Underrun**
An underrun of a boundary within an aggregate type.
Reading a piece of data of a specific (usually computed) byte length to a location designated as only being reserved for a shorter length. A variant of run-time data corruption.

**Null Pointer Dereference**
An attempt to dereference a pointer to the address 0.
http://www.owasp.org/index.php/Null-pointer_dereference

**Divide By Zero**
An attempt to perform integer division where the denominator is 0.

**Double Free**
Two calls to free on the same object. Freeing up the same piece of memory twice (e.g. in widely different parts of the code base) can cause runtime memory corruption.
http://www.owasp.org/index.php/Double_Free

**Use After Free**
A dereference of a pointer to a freed object. It may have been written over since you last thought you know what was there.
http://cwe.mitre.org/data/definitions/416.html

**Free Non-Heap Variable**
Many of these flaws in design cause embedded system failures in the field but they are also, in many cases, responsible for security vulnerabilities.

NIST SAMATE program creates a controlled peer reviewed set of deliberate vulnerabilities in code and vendors see if their tools will detect them.

Naturally, some tools are better than others!

Samate program at NIST  http://samate.nist.gov  Excellent resource.

Current Capability

- FDA has acquired some of these tools.
  - No one tool meets our needs but a suite can, when used together, combine the high points of each to provide us a workable approach to investigation.
  - Tools are in continual evolution and their pros and cons can be seen on SAMATE site
- These tools were not intended to be used by third parties!
- They were intended to complement the development process itself.
Current Capability

- FDA has acquired some other tools.
  - Code Comparison tools,
  - Code documentation tools
  - Code metrics suites
  - Hash tools
  - All tools are in continual evolution and keep getting better

- But it is the people who make the difference
  - Several staff with 30+ years of industry experience
Current Capability

♦ FDA has written some of its own tools.
  ♦ E.g. Checking if all the code needed to build the abstract representation has been received.

♦ A surprisingly intractable problem;
  ♦ How to quickly detect whether there are missing #includes and extraneous code and #includes
Current Capability

- FDA has developed a companion checklist of related dependency information to supplement any request for the source code.
  - Endianess
  - Memory alignment details
  - Protected areas of memory
  - I/O maps
  - Hardware specifications, registers info
  - Etc.
Current Capability

- When do we use this investigational technique?
- Rarely!
- But usually only within certain narrow policy guidelines when we wish to analyze and verify the code in devices where there is a compelling risk to the public health.
Current Capability

♦ When do we use this investigational technique?
♦ Consider these scenarios
  ♦ Many adverse events reported over a short interval
  ♦ Very large changes in code from version to version for a small ‘tweak’!
  ♦ Dissimilation about root cause
  ♦ Several adverse events reported after a ‘minor’ feature enhancement!
  ♦ Adverse events show a pattern of seemingly random failure modes
Current Capability

- The relationship of the QMS data to the analysis
  - FDA expects traceability from the design documentation into the software because the link to the device layer is essential for determining whether a discovered flaw could be the smoking gun of documented device failure.
  - Remember the adverse events only describe (badly) the apparent symptoms of the failure and the link from ‘flaw’ to ‘failure’ must be established.
Current Capability

- Longitudinal studies
  - Frequent adverse event reports concerning a device might be evidence of ‘out of control’ engineering.
  - We might ask for the last 10 versions of the software and compare what the changes are between them.
  - We establish the timelines of the versions against the adverse event reports, corporate developments and any submissions made to FDA.
The Future state

- Automated commercial software tools which perform data mining and semantic text mining between the architecture model and the source code.
  - Really do highlight many unmet requirements and unrequired features
  - Links/Calibrates the design control output to the product

- FDA has collaborative relationships with several academic institutions in this area.

- Very promising, very soon
The Future state

- Electronic submissions in the future might include
  - The software and its technical information
  - The hardware detailed specifications
  - The design documentation related to the specific product design.

E-Submissions
- Could facilitate background pre-processing of reviews
- Could facilitate lateral discovery of common mode failures between manufacturers very quickly
The Future state [more]

- Further advancing the consensus standards playing field
  - Reduced instruction set code practices
    - Auto industry, - MISRA
  - Integrated Advanced Static Checkers embedded in IDE’s
  - Widespread acceptance by industry of advanced Static Analysis as routine part of the Software Quality Assurance.
Your questions?

♦ Thank you for your patience.