Introduction to Safe & Secure Coding Methods
About me

- Frenchy, geek, runner, Pittsburgh-enthusiast
- PhD in France, author of the POK OS
- Previously at the European Space Agency

The views expressed here are solely those of the author in his private capacity and do not in any way represent the views of anybody else (either a person or a company/institution). The content published here has not been approved, endorsed, embraced, friended, liked, tweeted or authorized by anybody else than the author.
Story of the French Trains

• Old System, Operational since several years
  – Designed using empirical methods

• No standard for train and track size
  – Area-specific size

• Nationwide Order of new Trains
  – Several suppliers, trains must go around the country
  – One fits fits all?
Solution

Problem

Cost: 10M+ euros

La machine pour raboter les quais:
Architecture Issues

• We build trains since decades ...
  – … but we still do not build them right!

• So, what about other technologies (i.e. software?)

• Inadequate methods/process, tools and validation
  – Choose the best weapons (do not kill a mosquito with a rocket launcher!)
  – “No kiddo, you're not gonna make a plane with PHP”
Why Software Matters?

• Prevalence of Software everywhere
  - Software size increase … and it matters!
  - Difficult to manage, analyze and keep safe

• Potential catastrophic errors
  - Ariane 5 (rocket exploded)
  - Therac 25 (deaths)

• Nothing specific to security/safety …
  - This is rather the impact of defect that matters
Get the facts

Evolution of Software Size
(x50 over 32 years)

SLOC in thousands at first operational flight

Evolution of Security Incidents
(x8.82 over 6 years)

Incidents Reported by Federal Agencies in Fiscal Years 2006-2012

Number of incidents

Sources

- “National Strategy, Roles, and Responsibilities Need to Be Better Defined and More Effectively Implemented”
Usual suspects

- In-deterministic syntax or semantics
- Incomplete specifications and/or design
- Lack of tests and verification
Usual suspects

• In-deterministic syntax or semantics

• Incomplete specifications and/or design

• Lack of tests and verification
Syntax/Semantics Issues

Erroneous Code

```c
if (a = b)
{
  /* stmts */
}
```

```c
if (a == b)
{
  /* stmts */
}
```

```c
if (a == b)
  /* stmt1 */
  /* stmt2 */
```

Correct Code

```c
if (a == b)
{
  /* stmts */
}
```

```c
if (a == b)
{
  /* stmt1 */
  /* stmt2 */
}
```

Related known bug: Apple SSL/TLS goto fail

https://gotofail.com/
Syntax/Semantics Solutions

• Use compiler flags (-Wall, -pedantic)

• Deterministic semantics (e.g. Ada)

• Coding standards (e.g. MISRA-C)

• Static Analysis (e.g. UnderstandforC)
Ada

• Pros
  – Remove indeterminism at the language level
  – Built-in profiles for safety (e.g. Ravenscar)
  – Better verification using pre-/post-conditions

• Cons
  – Need a brain (expensive today)
Compiler Flags

• Pros
  – Easy to use and apply
  – Find usual language defects

• Cons
  – Need appropriate language
  – Limited to input semantics
  – Tool limitation
Coding Standards

● Pros
  - Restrict the language
  - Spot several language limitations

● Cons
  - Mostly tool-dependent
  - Proprietary tools
  - Limited Analysis
Static Analysis

- **Pros**
  - Easy to use/deploy
  - Spot issues not found by compiler or coding standards

- **Cons**
  - State space explosion
  - Proprietary tools
### Applying Solutions

```c
void toto (int a, int b)
{
    int c;
    if (a = b)
    {
        c = 2;
    }
}
```

<table>
<thead>
<tr>
<th>Tool</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>warning: suggest parentheses around assignment used as truth value [–Wparentheses]</td>
</tr>
<tr>
<td>ADA</td>
<td>if (var1 = var2) then - statements end if;</td>
</tr>
</tbody>
</table>
| MISRA-C | • a = b is a violation  
        • Need brackets after if |
| Static Analysis | Same as compiler (GCC) |
Usual suspects

- In-deterministic syntax or semantics
- Incomplete specifications and/or design
- Lack of tests and verification
Incomplete design/specifications

- Requirement are not well-defined
- Developers make their own assumptions
- Communication errors between contractors
Specifications error scenario

Temperature Regulator Specifications

- Acquire temperature using special sensor
- Regulate temperature using a controller
  - Activate the heater if temp < thresholdCold
  - Activate the cooler if temp > thresholdWarm

OEM

- Contract sub-system development with supplier
- Integrate, test and deliver

Sensor Supplier

Controller Supplier
Specifications error scenario

Sensor Supplier

Sensor subsystem

Sensor (celsius)

Controller Supplier

Controller subsystem

Controller (fahrenheit)

OEM integration

Integrated system

Inconsistent interfaces!

E.g. Mars Climate Orbiter
http://en.wikipedia.org/wiki/Mars_Climate_Orbiter#Cause_of_failure
Why it matters?

● Introduced early: in the requirements ...

● ... discovered lately: during integration testing

● Increase development costs, postpone delivery

Where ~80% of errors are introduced

Where they are fixed

Source: NIST - “The Economics Impact of Inadequate Infrastructure for Software Testing”
Incomplete specifications

- No concurrency mechanism
  - What about concurrent tasks?
  - e.g. iTunes on multi-core systems

- No interface specifications
  - Use of general types (int) for everything
  - e.g. system state
typedef enum {
    ok = 1,
    ko = 2
} mystate_t;

mystate_t state;

void change_state (mystate_t newstate) {
    state = newstate;
}

int state;

void change_state (int newstate) {
    state = newstate;
}

Impact when adding a new state? When specifying a bad value?
Solutions

- Check requirements completeness
- Check and Validate your Architecture
- Plan and Validate your Specifications and Design
Usual suspects

- In-deterministic syntax or semantics
- Incomplete specifications and/or design
- Lack of tests and verification
Testing Areas

- Unit testing
  - check your code is ok

- Integration testing
  - check your code is ok with the code of other folks

- Code Coverage
  - everything has been executed once
Unit Testing

- Check independent code parts
- Early discovery of error

Limitations
- Hard to test multi-threaded code (multi-core?)
- Often written by the developer himself …
- Cannot discover most errors from integration
Integration testing

- Check components assembly

- Write code to stress test system functions

- Discovery of most issues
  - Inconsistent components and specifications
  - e.g. performance (such as tasks not synchronized)
void foo ()
{
    if (bar == 1)
    {
        /* stmts1 */
    }
    else
    {
        /* stmts2 */
    }
}

- Coverage level, depends on software criticality
  - Statement Coverage
  - Decision Coverage
  - Modified Condition/Decision Coverage
- See safety standard (e.g. DO178C, ECSS, etc.)
Conclusion

- Software is still an emerging domain
  - We still make error when we build trains!

- Need to choose the right method/tools/process
  - Adopt & Keep Good Software Engineering Practice
  - Use Proven & Established Method & Technology

- Test, test and test … as early as possible!
Questions?

- Julien Delange, PhD
  - julien@gunnm.org
  - http://julien.gunnm.org

- Stay in Touch, twitter @juli1pb

Note: The views expressed here are solely those of the author in his private capacity and do not in any way represent the views of anybody else (either a person or a company/institution). The content published here has not been approved, endorsed, embraced, friended, liked, tweeted or authorized by anybody else than the author.