Software Verification and Reliability - I

Automated Testing and Formal Methods

"Testing is the process of comparing the invisible to the ambiguous, so as to avoid the unthinkable happening to the anonymous."

--James Bach

Verification and Validation

- Validation
  - *Did you build the right thing?*
- Verification
  - *Did you build it right?*
- Today, we'll focus on *Verification*

Part - I

Automated Functional Testing

“Programmer's Drinking Song (sung to the tune of "100 Bottles of Beer")

99 little bugs in the code,

99 bugs in the code,

fix one bug, compile it again,

101 little bugs in the code.

101 little bugs in the code....

(Repeat until BUGS = 0) ”

-- *Anonymous*

Let’s put our thinking caps on!

- How do you test a ***Table***?
  - No, not a database table
  - No, not a data structure either
  - We are talking about a *physical object with four legs, often used to put stuff on* ☺ … yes, like a computer table, dining table, study table etc.

The process of Automated Testing: An Overview

- A *very expensive* process (in terms of man hours)
- Consists of:
  - *Test Analysis*
  - *Test Specification*
  - *Cost Estimation*
  - Developing the *Test Suite*
  - And finally running the suite etc.
- Often, the *Software Design Engineer in Test* deals with much more challenging development tasks, than the development team
- It must never be taken lightly!

A Tale of two Parameters: *Cost* and *Reliability*

- For each Test Case (TC) that is developed, there is an associated cost
- As always, the management would like to keep it within budget
- This results in the reduction of *Test Effort* i.e. try to do away with as few TCs as possible
  - Lower number of TCs => fewer man hours => lower cost
- But that's a good thing, right? … we don't have to pay as much?
- Not really. This is where the *"No Free Lunch Theorem"* might help ☺
  - There is always a trade off. Reduction in cost often leads to reduction in Software Quality
  - Reduction in *Quality* => possibly lower *Reliability*

Reliability in terms of Functional Requirements

- Background
  - An existing code base of *millions of lines of code* (e.g. a large-scale telecommunication system)
  - Code base split over *modules* and even independent *nodes*
  - Multiple external interfaces
  - New *features* (product customizations) are being added all the time (continuous activity)
- So how does that affect system reliability?
  - Each new feature implements new functional requirements
  - Therefore, every new feature *changes system behavior*
  - For the customer, a system is not *reliable* unless it delivers the *expected behavior*
  - Therefore, *reliability* (in terms of expected behavior) has to be maintained, with each new customization/upgrade

Of *Quality* and *Reliability* – A Tale from the Industry

- Domain
  - Telecommunications: Specifically, real-time call charging for pre-paid subscribers
- System
  - Intelligent Network (IN)
- Interfaces
  - Real-time Call Processing
  - Subscriber Provisioning
  - System Administration
  - (each interface uses a different protocol)
- Current system state
  - A code base of 10 million
  - Multiple customization are being added in parallel

Of *Quality* and *Reliability* – A Tale from the Industry … contd (1).

- A new feature request comes in
- Test Analysis shows that we need 40 TCs
- Cost estimation:
  - Cost per TC = 8 man hours => Total cost of Test Effort = 8 × 40 = ~320 man hours
  - Management goes, “what??? … are you insane??” … “We must reduce this number!”
- If we reduce the number of TCs, do you think we can still ensure the same quality? … (remember, this is pure black box testing. No cheating ☺!)
  - No. Not in general.
- It is kind of a moral quandary, isn’t it? … so what must an ambitious engineer do???

Of *Quality* and *Reliability* – A Tale from the Industry … contd (2). The Solution

- Reduce the number of man hours required per TC
- We actually managed to do this
  - New Average Cost per TC = ~2.6 man hours
  - => Total cost for Test Effort = 2.6 × 40 = 104 man hours
  - Reduction = 320 – 104 = 216 man hours (this was radical!) … moah ha ha ☺
- This raises two questions
  - How did we do it?
  - Were there any side effects?
- Let's talk about the side effects first!

Of *Quality* and *Reliability* – A Tale from the Industry … contd (3). The side effect

- The *Test Suite* developed for each feature only tests that specific feature
- To make sure that the existing functionality is in tact, the *Regression Suite* is used. Here's a simple way of doing this:
  - For each new TC, add it to the regression suite
  - Verify the existing functionality by running the regression suite
- This all sounds good. So what's the problem?
  - The size of the regression suite grows at a fast pace
  - This means more computation time is required to run the suite
  - => Increased time to market … not good!
- Open issue ☹

Of *Quality* and *Reliability* – A Tale from the Industry … contd (4). The Framework approach

- Coming back to, *"How did we do it?"*
- Introduction of *automation* to the TC development process
  - In simple terms, a framework was introduced that speeds up the process
  - But it was hard for other engineers to use it (requires a thorough knowledge of the *protocols*)
- Why is such automation hard to achieve?
  - *System specifications*, *Requirement Specifications*, as well as *Test Specifications* are all done in *natural language* e.g. English
  - It would require *Natural Language Processing* in order to extract the required information
- Lets say we can do *Natural Language Processing*. Would that make automation feasible???

Part - II

Formal Verification

"Beware of bugs in the above code; I have only proved it correct, not tried it."

-- Donald Knuth

"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.“

-- Edsger Dijkstra

The need for Formal Specification

- Natural language is ambiguous
  - Trust me. I know! ☹
- If the objective is to attain higher *reliability*, *ambiguity* does not help much
- There are two reasons for using Formal Methods
  - You can prove the *correctness* of your software system
  - It facilitates *automation*
- *Protocol Specification* is one example
  - Helped us automate the protocol testing framework
- We'll talk more about *Formal Specification* and *Formal Verification* tomorrow!

Summary

- *Software Reliability* corresponds to *functional requirements* as well
  - Behavior of the system is the key
- *Cost* is a major barrier in terms of achieving high *Software Quality* and *Reliability*
- *Automation* is a long term investment, with potentially high returns
- *Formal methods* (or pseudo-formal methods) are not only good for *correctness*, they may also aid in *automation*

"I really hate this damned machine;
I wish that they would sell it.
It never does quite what I want
but only what I tell it.“

A Programmer’s Lament