Rakesh Ranjan

Test Less – Test Right

Short Bio:

- Manager of Software Testing & QA at IBM silicon Valley Lab, San Jose CA
  - http://www-01.ibm.com/software/data/db2/
- Part time Lecturer, Department of Computer Engineering, San Jose State University
  - http://www.sjsu.edu/people/rakesh.ranjan/
- 17 years of IT industry experience:
  - Software systems architecture on Linux/Unix/IBM Midrange
  - Large software product development & testing
  - Extensive Database & Business Analytics experience
- Co-authored 2 books on DB2 and Business Intelligence
- Frequent speaker at Midrange system conferences
- Strategic thinking and execution with completeness of visions

http://www.linkedin.com/in/ranjanr

http://www.ranjanr.blogspot.com/

Source: N/A

Agenda

- Introduction/Smarter Testing
- Random Sampling
- Combinatorial Testing Techniques
- Code Coverage and gap analysis
- Memory leak analysis
- Change centric Testing
- Orthogonal Defect classification
- Smarter Test Infrastructure
- Exploratory Testing Techniques

Source: N/A

Why Smarter Testing?

- Finding and fixing a software problem after delivery is extremely expensive(x100) than fixing it in early design or requirement phase.
- Current software projects spend 40-50% of their time on avoidable work.
- 20% of defects generate 80% of avoidable work
- 10% of defects cause 90% of system downtime
- Peer reviews catch 60% of the defects
- Scenario based reviews (focused reading techniques) catch 35% more defects than general reviews
- Personal checklists and disciplined practices can reduce defect introduction by 75%

Source: N/A

Smarter Testing means Strategic Defect Reduction

Source: N/A

Challenges of Complexity in a Large Product Environment

- Too many branches & simultaneous releases
  - Multiple development releases
  - Multiple maintenance releases
- Too many platforms to support
  - Debug and optimized builds on various platforms and architecture
  - Special customer builds & Tests
- Maintaining Quality while doing frequent merges
  - Code dependency forces frequent merges with parent branch
  - Need to keep sanity of the branch on all platforms
- Keeping up with changes
  - Changes in OS/kernel/patches
  - Changes in compiler / parser / runtime components

Source: N/A

Cost of Quality

- How to respond to people who complain about cost of quality?
  - *"If testing costs more than **not testing** then don't do it" - Kent Beck*
- Why can't we just test everything?
  - *"I want full testing coverage of the software. Our policy is zero tolerance. We won't have bad quality on my watch."*
  - *Lack of appreciation for complexity of testing*
- We use statistics, confidence levels and calculated risk
- Thankfully, Mathematics and Statistics are there to help

Source: N/A

Statistics – Random Sampling

- Randomly selecting combinations for testing
- Very high number of tests required for very high quality goal
- Coverage can not be used as key measurement of quality
- Express a level of confidence for a level of quality
- For extremely high level of quality, approach does not make sense

99% Confidence and 99% Quality	
Number of unique combinations of inputs	Number of samples required
100	99
1,000	943
10,000	6,247
100,000	14,627
1,000,000	16,369
10,000,000	16,613
100,000,000	16,638
unknown	16,641

99% Confidence for one millon samples	
Desired quality level	Number of samples required
90%	166
99%	16,393
99.9%	624,639
99.99%	994,027
99.999%	999,940

Source :http://www.developerdotstar.com/mag/articles/test_smarter_not_harder.html

Pairwise Testing and Its Effectiveness

- With assumption that bugs in software is caused not by individual input but by combination of two factors
- Is designed to get coverage of every possible combination of two variables, without testing every possible combination of all the variables
- Very high number of tests required for very high quality goal
- For reasonable quality goal, effectiveness is high

We measured the coverage of combinatorial design test sets for 10 Unix commands: basename, cb, comm, crypt, sleep, sort, touch, tty, uniq, and wc. […] The pairwise tests gave over 90 percent block coverage. [D. M. Cohen et al., 1996]

A set of 29 pair-wise AETG tests gave 90% block coverage for the UNIX sort command. We also compared pair-wise testing with random input testing and found that pair-wise testing gave better coverage. [D. M. Cohen et al., 1997]

Source: http://www.pairwise.org/results.asp

Orthogonal Array Testing Strategy (OATS)

- Each pair of input occurs only once
- All pairwise combinations have been covered.
- Also provides redundant coverage of every single value for each control
- Applying orthogonal array technique in testing requires determining the size of the array
- No of combinations = no of test cases
- Tools can be used to generate array and map the actual values to the variables
- OATS is a proven efficient method of complex testing
- OATS combined with boundary value analysis can be very powerful test technique

	A	B	C
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2

Source: If applicable, describe source origin

Code Coverage & Gap Analysis – Execution != Tested

- Structural Testing(White Box)
  - Code driven
- Functional Testing(Black Box)
  - Requirements driven
- Statement coverage
- Decision coverage
- Condition coverage
- Using code coverage for better regression system

- How does it work?
  - Code is instrumented, static profiling is generated
  - Test case execution time, dynamic profiling is generated
  - Both profiling merged and report generated

Files				Functions				Blocks			
total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%
77	67	10	**87.01**	1,731	1,174	557	**67.82**	77,316	33,005	44,311	**42.69**

intel *generated by Intel® Compilers code-coverage tool*

Web-Page Owner: Nobody

Source: If applicable, describe source origin

Coverage Analysis – Improving Your Test Harness

- Which test case caused the coverage
- Quality of covered code?
- What is the correlation between block coverage and decision coverage
- What is the correlation between complexity of code and defect
- What is the cost effective way of identifying uncovered code
- Is complexity a better predictor of defects than lines of code?

Source: If applicable, describe source origin

Memory Leak Analysis – Different Approaches

Memory Leak is closely related to the notion of lifetime of heap memory(run time memory); A pool is being leaked if the program or the run time system doesn't reclaim its memory when the lifetime has ended. Memory leak is difficult to find and expensive to fix.

- Traditional Approach to find memory leak
  - A gradual reduction in free memory, eventually leading to paging
  - Application becoming slower and slower
  - Application memory footprint getting bigger
  - Dump the memory block allocation
  - Grep for pattern and comparing for Pools and blocks
  - Problem of false positives; manual work
- Contradiction approach
  - Reverse heap analysis

Source: If applicable, describe source origin

Finding Memory Leak by Contradiction Approach

- The example function takes a linked list x as an argument and returns the reversed list pointed to by y
- Statement 12 is taken as the point where memory leak might occur
- Reverse inter-procedural flow analysis is done from that point to disprove the negation
- Errors are disproved by contradicting their presence

```
 1  typedef struct list {
 2      int data;
 3      struct list *next;
 4  } List;
 5
 6  List *reverse(List *x) {
 7      List *y, *t;
 8      y = NULL;
 9      while (x != NULL) {
10          t = x->next;
11          x->next = y;
12          y = x;
13          x = t;
14      }
15      return y;
16  }
```

Source: http://www.cs.cornell.edu/~rugina/papers/sas06.pdf

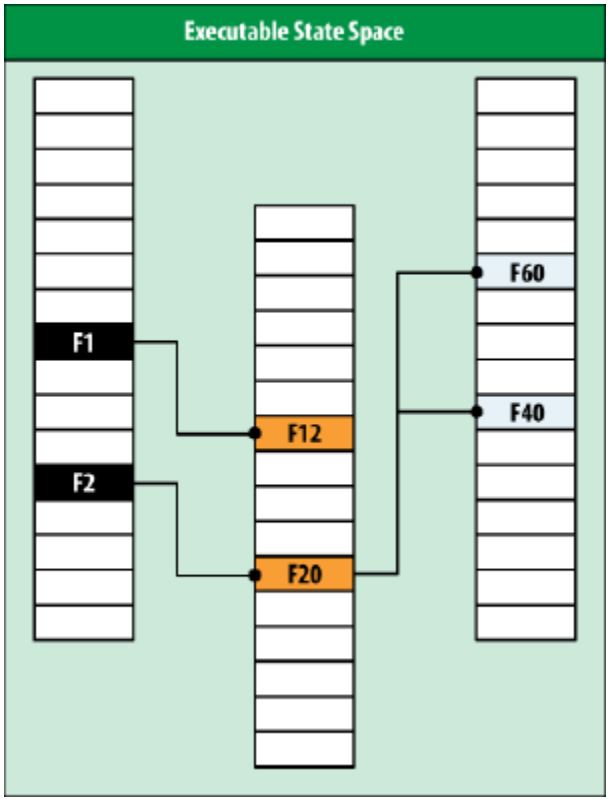
Automated Error Prevention

- Automated Error Prevention
  - You should find (or learn from someone who has found and analyzed) a bug only once. The knowledge gained in the process of finding and analyzing bugs should be used to improve the process so that you never encounter the repeat occurrences of similar bugs in the product.
  - Example: database instance shutdown due to resource leak
  - Memory was allocated but not de-allocated after being used.
  - One way to approach the problem is to get this fixed
  - Automated Error Prevention approach will be to design the practices where every allocation will be forced to use de-allocation, so that problem does not re occur.
- How to enforce automated error prevention?
  - Code reviews (inefficient way)
  - Static code analysis
  - Toll that scans such coding mistakes and reports to developers

Source: If applicable, describe source origin

Change-Centric Testing

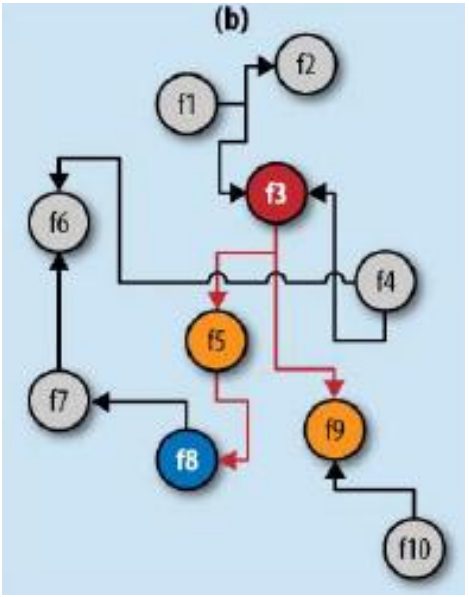
- Methodology to capture the impact of incremental code changes on the state space of the executable
- Black shades represent actual change in the code(methods)
- Dark gray represent methods that depend on changed code
- Light gray represent third order change impact (functionalities that might be impacted by black and dark gray)
- Overall sanity of the codebase needs to be ensured by running other regression test



Source: If applicable, describe source origin

Change-Centric Testing – How To

- Understanding the dependency between caller/callee in an executable; tools can be used to generate call graph; store data for tools to query/retrieve
  - Callgrind (call graph generator)
- Understanding mapping of source files to test cases and code coverage
  - Define mapping and store the data
  - Identify gaps in coverage
  - Develop test plan to address coverage gaps



Source: If applicable, describe source origin

Smarter Infrastructure

Easy & efficient Development environment

On-demand Build & Test

Continuous code Integration

Product build & validation On multiple platforms

Auto notifications & Reporting

Integration with Defect tracking

Code Integration With main branch

Product delivery & packaging

Source: If applicable, describe source origin

Code Quality Management – solution & challenges

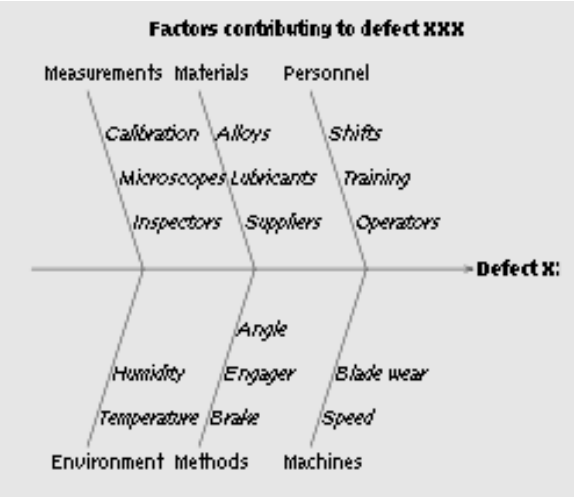
- Tools like Static analysis and other enforcement built into check-in system
- Regression system in place
- Automated build break reporting system that points to individual check-in
- Continuous monitoring of testing and standards compliance
- Self auditing and correction
- Project management dashboard to report key development performance indicators

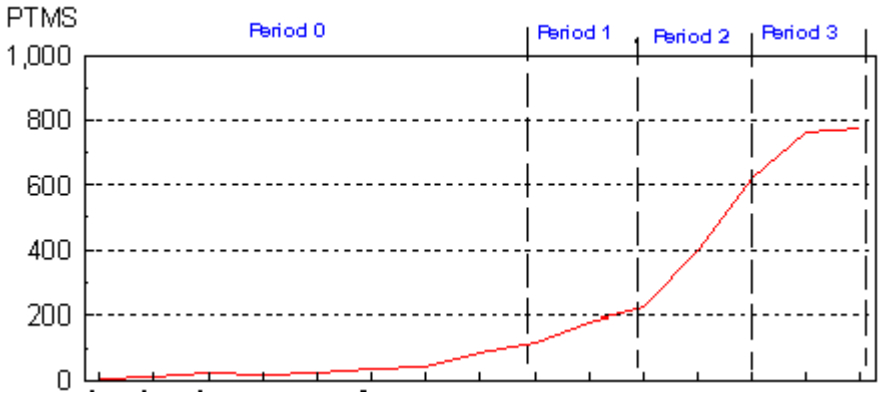
- Should be able to track progress of distributed (geographically dispersed) teams
- Integration with existing tools such as SCM
- Individual compliance tracking and reporting
- No impact on existing processes
- Real time feedback

Source: If applicable, describe source origin

Defect Analysis

- Root Cause Analysis
  - Captures extensive data on defects
  - Time consuming
  - Expensive
  - Points to too many actions as a result
- S-curve
  - Easy to monitor trends
  - Inadequate capture of semantics
  - Not capable of suggesting corrective actions

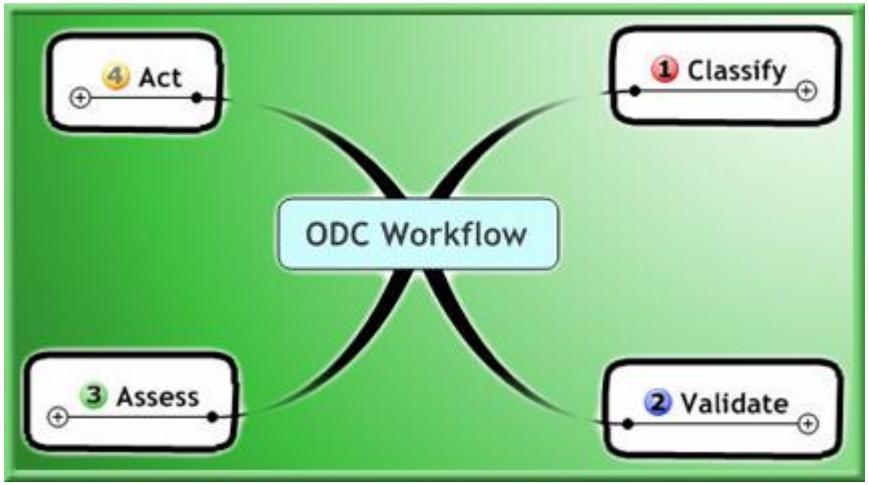




Source: If applicable, describe source origin

Orthogonal Defect Classification (ODC)

- Is a scheme to capture the semantics of the defect quickly
- Makes mathematical analysis possible on defects
- Analysis of ODC data provides diagnostics method for various phases of SDLC



- Classification captures how the defect was discovered, the effect it would have on customers or did have on customers, and the scope and scale of what had to be fixed.
- Validation, which is performed by a subset of the classification team, helps ensure that the classification step was done correctly.
- Assessment analyzes the data to understand what it means. It is normally done by a very small team on your product or in your area.
- The first three steps only identify what needs to be done. Identifying and implementing those actions requires skill, determination and management support.

Source: http://www.research.ibm.com/softeng/ODC/ODCEG.HTM#defects

Exploratory, Ad-hoc and Scripted Testing

The plainest definition of exploratory testing is test design and test execution at the same time.

- James Bach

- Which functionality is most important to the project's intended purpose ?
- Which functionality is most visible to the user ?
- Which functionality has the largest financial impact on users ?
- Which aspects of the application are most important to the customer ?
- Which aspects of the application can be tested early in the development cycle ?
- Which parts of the code are most complex, and thus most subject to errors ?
- Which parts of the application were developed in rush or panic mode ?
- Which aspects of similar/related previous projects caused problems ?
- Which aspects of similar/related previous projects had large maintenance expenses ?
- Which parts of the requirements and design are unclear or poorly thought out ?
- What do the developers think are the highest-risk aspects of the application ?
- What kinds of problems would cause the worst publicity ?
- What kinds of problems would cause the most customer service complaints ?
- What kinds of tests could easily cover multiple functionality's ?
- Which tests will have the best high-risk-coverage to time required ratio ?

Source: If applicable, describe source origin

Thank you

Source: If applicable, describe source origin