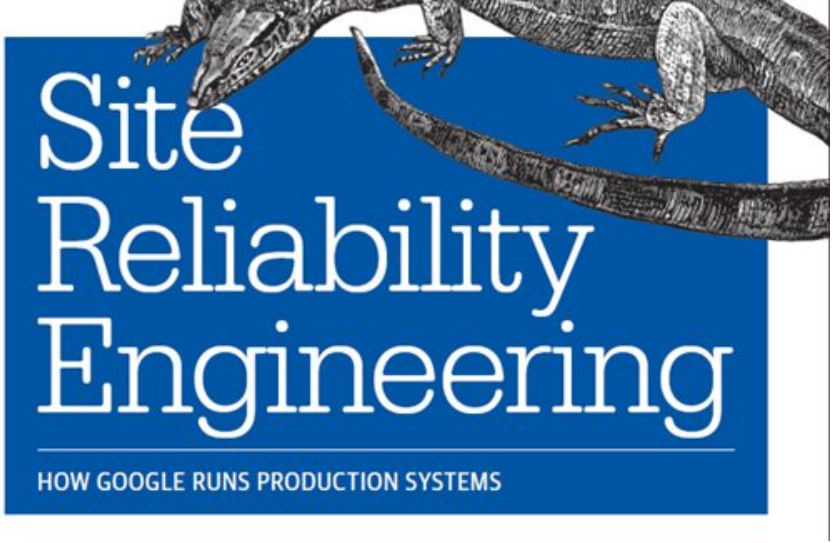


Software Testing for Continuous Delivery

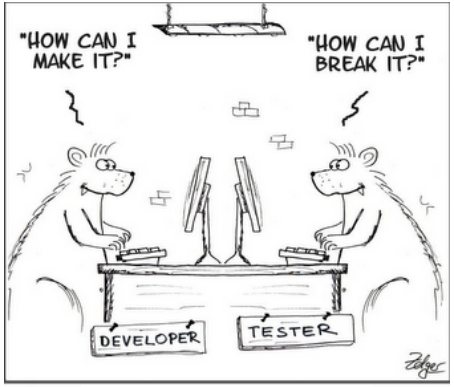
Seminar 12: Reliability (con't) & Types of Testing

Dr. Byron J. Williams
October 09, 2019

UF Herbert Wertheim College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY *of* FLORIDA

Content sourced from: "Software Quality Engineering" by Jeff Tian
Images courtesy Google Image Search

FICS Research



They are not so much different,
but they have different path for the same goal,
to improve quality!!

Testing Perspectives

"developer" **-** person whose primary responsibility is to write source code - the output of the developers should be working software, not just something that compiles

Developer Testing (developer mind-set)

taking ownership of the quality of the produced code, instead of expecting that someone else will test it

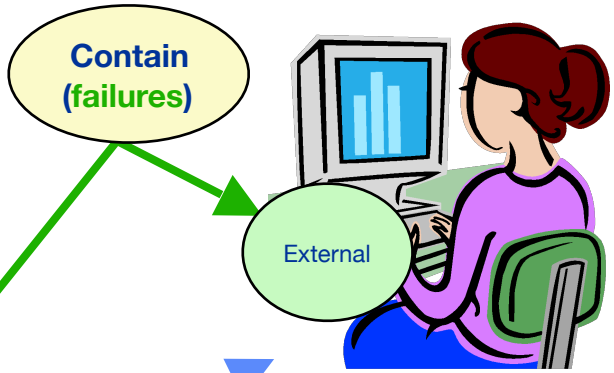
Developer testing is an umbrella term for all test-related activities a developer engages in

To build - **Testing to Support**

Software Testing (tester mind-set)

investigating how the product might fail

To break - **Testing to Critique**

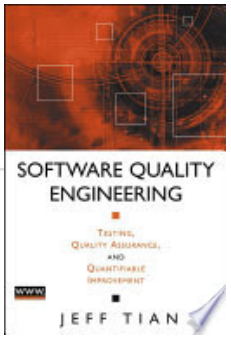


Correctness: few problems with limited damage to customers

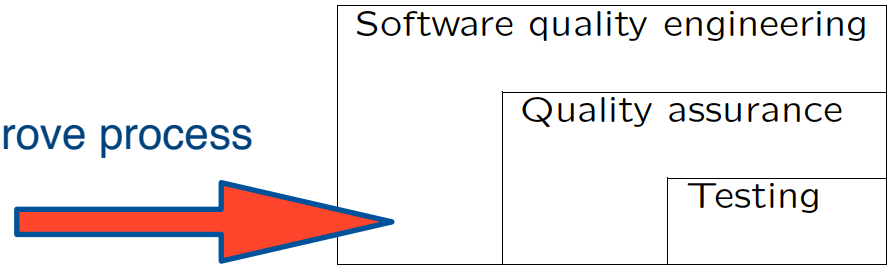
Quality Assurance

To ensure that few, if any, defects remain in the software when it is delivered to its customers or released to the market

Software Quality Engineering



- Goal - Ensure software quality
- Premise - Quality cannot be achieved by assessing an already completed product. The aim, therefore, is to **prevent quality defects** or deficiencies in the first place, and to make the products assessable by quality assurance **measures**
  - Defined: ***Quality assurance*** is the systematic activities providing evidence of the fitness for use of the total software product
- QE Outcome
  - Learn from and package experiences to improve process



Reliability

- **Reliability**: Probability of failure-free operation for a specific period or a given set of input under a specific environment

- **Accomplished through**
  - availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning

- Key Measures - System Quality
  - Mean-time-to-**Failure** (**MTTF**) - how often does the thing stop working
  - Mean-time-to-**Repair** (**MTTR**) - once it stops working, how long does it take until you fix it

What is Site (system) Reliability Engineering?

- Assigned Reading:
  "What is Site Reliability Engineering?"
  - https://landing.google.com/sre/interview/ben-treynor.html

"Fundamentally, it's what happens when you ask a software engineer to design an operations function."

— *Ben Treynor, VP Engineering, Google*

Defect Reduction: Testing

- **Execution of software and checking results**
  - Locates failures
  - Isolate and fix the fault(s) that led to the failure
- When to test
  - **Need some executable**
  - Unit tests --> components —> integration, system --> acceptance test of entire system
  - Alpha & Beta testing
  - Can also use prototypes
- Questions:
  - **What to test? When to stop? Lots more…**
  - checklists, coverage information, usage scenarios, reliability…others *(next topics)*

Defect Resolution

- Actions agreed upon and impact / priority determined
- Defect logging & tracking
- Consistent defect interpretation & tracking
- Timely defect reporting (used to monitor and control projects)
- Learn from past problems (locations in the code, defect types, developer issues)
- Developer Issues - conceptual mistakes, unfamiliarity with domain, inexperience with methods,

Ad-Hoc Testing

- Ad-hoc testing
  - “run-and-observe”
  - Implicit checklists may be involved
- Drawbacks
  - Lack of structure
  - Likely to miss
  - Likely to repeat oneself
  - In general, the whole process is hard to repeat
- One way to structure is to build a checklist

Systematic Testing: Checklists

- "systematic" ➔ process is explicitly defined
  - Recall: how to achieve software quality (SE)
- Testing with checklists
  - List of items that must be tested —> Each item is "checked off" —> When list is complete, testing is done
- Examples
  - Functional (black-box)
  - System elements (white-box)
  - Structures (implementation/white-box)
  - Properties (black-box or white-box)

Functional Checklists: Exercise

- Function/feature (external) checklists
  - Black-box in nature
  - List of major functions that are expected
- An example high-level functional checklist for an ATM

- Card insertion & rejection
- Password management
- Envelope and printing
- Abnormal termination
- Backup and restore
- Commit and rollback
- Locking
- Logging and recovery
- Migration
- Stress
- …

Implementation Checklists: Example

- What are the different forms of implementation (internal) checklists
  - White-box in nature but at varying levels of abstraction
    - E.g., lists of modules/components/etc. - Module interaction patterns
- Example: coding standard
  - Naming conventions: to improve software maintainability
    - Functional (black-box)
  - E.g., standard items (in concurrency control)
    - ACID (atomicity, consistency, isolation, durability)
    - Locking (e.g., read-lock, write-lock, two-phase)
    - Optimistic vs. pessimistic
    - Serialization (timestamp ordering, commit ordering, etc.)

Testing for Partition Coverage

Sensitize test cases

- i.e., defining specific input variables and associated values to exercise certain parts of the program in the white-box view or to perform certain functions in the black-box view
- e.g., function `add(int a, int b)`
- considering valid/invalid input values of *a* and *b*
- How many cases are in exhaustive test?

Test Case	Condition		Input	
	int a	***int b***	***a***	***b***
1	False	False	3.2	-0.4
2	False	True	"UF"	2
3	True	False	7	3/4
4	True	True	-9	-2

Partitions: Formal Definitions

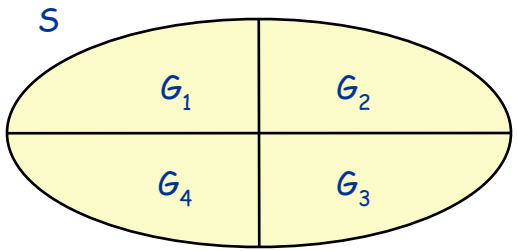
- A set S contains a list of unique elements
- A partition of S creates subsets G_1, G_2, … G_n such that
  - Sets are mutually exclusive $\forall i, j,\ i \neq j \Rightarrow G_i \cap G_j = \emptyset$
  - Sets are collectively exhaustive $\bigcup_{i=1}^{n} G_i = S$

Better efficiency (eliminate "duplicates")

Better coverage (eliminate "holes")

- Each $G_1..G_n$ in a partition is called an **equivalence class**, where the specific relation that is used to define the subsets is:
  - Reflexive - holds on every member by itself
  - Symmetric - holds if order is change
  - Transitive - holds in a relation chain

Partitions



$$\forall i, j,\ i \neq j \Rightarrow G_i \cap G_j = \emptyset$$

$$\bigcup_{i=1}^{n} G_i = S$$

x is a natural number AND x<=100//x represents "grade"

If we're dealing with "A" students most of the time…

Are the following partitions?

[0,50) and [50,100]

[0,60), [60,70), [70,80), [80,90), [90,100]

Subset-even={0,2,4,…,100} and Subset-odd={1,3,5,…99}

Partitions-Based Testing

- **Basic ideas:**
  - Members in equivalence class are treated “equivalent” —> Defining meaningful partitions
  - Sampling from partitioned subsets for different types of partitions
  - Coverage of partitions: uniform
- **Different types of partitions**
  - Pure membership based partitions:
    - E.g., components in a subsystem
    - direct sampling, e.g., one component from each subsystem for coverage
  - Properties/relations used in definitions:
    - direct predicates on logical variables, e.g., P && Q
    - vs. operations on numerical variables, e.g., x<=100
  - Combinations
    - E.g., non-negative integers less than 21

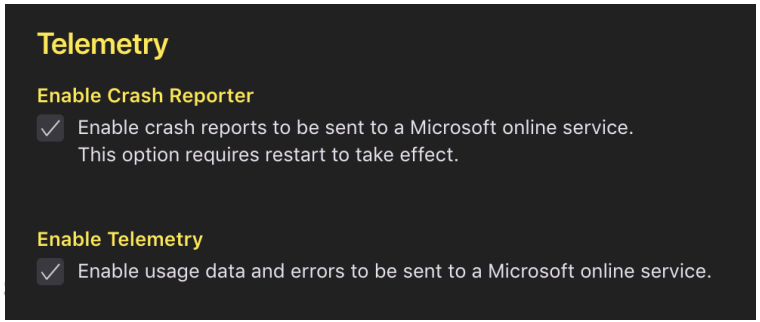
UBST (Usage-Based Statistical Testing)

- UBST ensures reliability
- Reliability: Customer view of quality
  - Probability: statistical modeling
  - Time/input/environment: OP
- **OP: Operational Profile**
  - Quantitative characterization of the way a system will be used
  - Generate/execute test cases for UBST
  - Realistic reliability assessment - development decisions/priorities

OP (Operational Profile)

- ***Definition***: a list of disjoint set of operations and their associated probabilities of occurrence
- A quantitative way characterization of the way a software system is or will be used
- Obtained via measurement, survey, & expert opinion
- Operations: multiple possible test cases or multiple runs
  - Each operation corresponds to an individual sub-domain in domain partitions, thus representing a whole equivalence class.

John D. Musa (RE)
Pioneer



Source: IEEE Software, http://www.computer.org/portal/web/csdl/doi/10.1109/MS.2009.132 & Google Images

Usage-Based (Statistical) Testing

- **Usage-based statistical testing (UBST)**
  - Actual usage and scenario/information
  - Captured in operational profiles (OPs)
  - Simulated in testing environment
    - (too numerous ➔ random sampling)
  - Example: (myCourses / beta-testing: add a course; delete a course; produce report…
- **Applicability**
  - Final stages of testing
  - Particularly system/acceptance testing
  - Use with software reliability engineering
- Termination criteria: reliability goals

Coverage-Based Testing

- **Coverage-based testing (CBT)**
  - Systematic testing based on formal models and techniques
  - Testing models based on internal details or external expectations
  - Coverage measures defined for models - Testing measured by coverage goals
  - Example: (myCourses / unit-testing: post announcement & set receiver role(s); set up submission deadline; …)
- **Applicability**
  - All stages of testing - Particularly unit and component testing
  - Later phases at high abstraction levels
- Termination criteria: coverage goals

Comparing BBT with WBT

	BBT	WBT
Perspective	external behavior (functional)	internal implementation (structural)
Defect Focus	failures	faults
Scale	large software (as a whole)	small objects (looking inside)
Timeline	later (e.g., acceptance testing)	earlier (e.g., unit testing)
Tester	IV&V	developers themselves

Comparing UBST with CBT

	UBST / BBT	CBT / WBT
Perspective	external behavior (functional)	internal implementation (structural)
Stopping Criteria	reliability goals	coverage goals
Scale	large software (as a whole)	small objects (looking inside)
Timeline	later (e.g., acceptance testing)	earlier (e.g., unit testing)
Tester	IV&V	developers themselves