

# SW Development & QA

## CSH5 Chapter 39

### “Software Development and Quality Assurance”

Diane E. Levine, John Mason, & Jennifer Hadley

1

Copyright © 2011 M. E. Kabay. All rights reserved.

## Topics

- Introduction
- Goals of SW QA
- SW Devt Lifecycle
- Types of SW Errors
- Designing SW Test Cases
- Before Going Into Production
- Managing Change
- Sources of Bugs & Problems

2

Copyright © 2011 M. E. Kabay. All rights reserved.

## Introduction

- Development potentially affects all 6 elements of Parkerian Hexad
- Usually affects
  - ❑ Integrity
  - ❑ Availability
  - ❑ Utility
- COTS (commercial off the shelf) software often needs to be customized
- Many programs still developed from scratch
- Project managers typically underestimate
  - ❑ Effects of errors
  - ❑ Time required to get project right

3

Copyright © 2011 M. E. Kabay. All rights reserved.

## Goals of SW QA

- IEEE definition of quality: *“degree to which ...[x]... meets customer or user needs or expectations.”*
- Uncover All Program Problems
  - ❑ Assume they are there – TEST to find them
- Reduce Likelihood that Defective Programs Enter Production
  - ❑ Costs escalate (~10x) with every stage through which problems are undiscovered or ignored
- Safeguard Interests of Users
  - ❑ No point in having software that is irrelevant
  - ❑ SQA should report at same mgmt level as SW development
- Safeguard Interests of SW Producers
  - ❑ Avoid legal liability for failures, damages

4

Copyright © 2011 M. E. Kabay. All rights reserved.

## SW Devt Lifecycle (SDLC)

- Overview of SDLC
- Phases of Traditional SDLC
- Classic Waterfall Model
- RAD & JAD
- Integrating Security at Every Phase

5

Copyright © 2011 M. E. Kabay. All rights reserved.

## Overview of SDLC

- SW devt projects often large
  - ❑ Need orderly process to coordinate efforts
- Phased approach defines milestones
  - ❑ Often defined using specific documents
  - ❑ But in reality, development does not match strict step-by-step progression
  - ❑ Often have overlapping stages in different sections of project
- Different models for development co-exist
  - ❑ Useful in different contexts
  - ❑ Adapt to needs for rapid access to early productivity vs rigid demands for strict controls

6

Copyright © 2011 M. E. Kabay. All rights reserved.

## Phases of Traditional SDLC



1. Investigation
2. Analysis
3. Design
4. Coding and debugging  
*[error in text: "Decoding"]*
5. Testing
6. Implementation
7. Maintenance

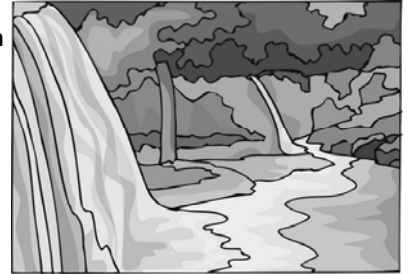
7

Copyright © 2011 M. E. Kabay. All rights reserved.

## Classic Waterfall Model



1. Requirements and analysis
2. Design
3. Implementation
4. Testing
5. Maintenance



8

Copyright © 2011 M. E. Kabay. All rights reserved.

## RAD & JAD (1)



### ➤ Overview of Iterative Methodologies

- Based on 80/20 rule of productivity
  - ✓ First 80% of functionality can be built in first 20% of project time
  - ✓ Avoid making users wait for perfection – get them improvements to their work ASAP
- Stronger, continual user involvement
- Small development teams
- Prototyping software to glean user responses and suggestions for improvement
- Software reuse
- Automated tools

9

Copyright © 2011 M. E. Kabay. All rights reserved.

## RAD & JAD (2)



### ➤ Phases

- Requirements planning
- User design
- Construction
- Cutover

### ➤ Techniques

- JRP (joint requirements planning)
- JAD (joint application design)

10

Copyright © 2011 M. E. Kabay. All rights reserved.

## Integrating Security at Every Phase



- Security is not an add-on
- Must be integrated from start and included at every phase
- Applies to all development methodologies
- Reduce project cost by catching and preventing problems as early as possible
  - Every phase multiplies costs of going back to fix a problem ~10-fold



11

Copyright © 2011 M. E. Kabay. All rights reserved.

## Types of SW Errors



- Internal Design or Implementation Errors
- User Interface

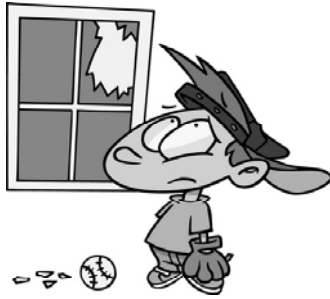


12

Copyright © 2011 M. E. Kabay. All rights reserved.

## Internal Design or Implementation Errors

- Initialization
- Logic Flow
- Calculation
- Boundary Condition Violations
- Parameter Passing
- Race Conditions
- Load Conditions
- Resource Exhaustion
- Interapplication Conflicts
- Other Technical Errors
- Regulatory Compliance Considerations



13

Copyright © 2011 M. E. Kabay. All rights reserved.

## Initialization Errors

- Difficult to find
- Programmer forgets to save initialization data to disk **BEFORE** trying to run program
  - ❑ So program fails on first run
  - ❑ But works on second try
- Or program may leave out only certain initial values, causing funny results on first run or loop



Power

14

Copyright © 2011 M. E. Kabay. All rights reserved.

## Logic Flow Errors

- Error when control passes to wrong routine or module
  - Can happen when a conditional statement either
    - ❑ Looks at wrong variable or
    - ❑ Looks at right variable but has bad value
  - Can also happen when execution falls through a missing end-of-routine marker



15

Copyright © 2011 M. E. Kabay. All rights reserved.

## Calculation Errors

- Wrong formula
- Roundoff errors when programmer forgets precision of data fields
  - ❑ E.g., multiplies a long-real by a short-real variable
  - ❑ Thus loses precision in product
- Variables may be defined with wrong precision
  - ❑ E.g., 16 bits instead of 32 bits

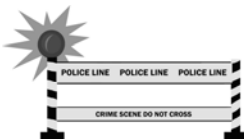


16

Copyright © 2011 M. E. Kabay. All rights reserved.

## Boundary Condition Violations

- Attempting to read or write beyond the end of an array is viewed as a sign of serious corruption by most operating systems
  - ❑ Call a hardware *halt* instruction – e.g., HP3000 *suddendeadth()* with parm for type of error
- Arrays defined with wrong number of maximum values
  - ❑ E.g., “year” array with only 365 values
  - ❑ In leap year,
    - ✓ December 30 is #365
    - ✓ Next day tries to roll to 366 > limit
    - ✓ Causes process or system crash
- But going under or over any limits can cause problems



17

Copyright © 2011 M. E. Kabay. All rights reserved.

## Parameter Passing Errors

- Modules / routines / objects must respect expected / defined variables passed among them
  - ❑ Passing wrong types or contents may cause serious errors
- Can corrupt data (e.g., writing into wrong area of memory or disk)
- May affect logical flags
- Can switch execution into aberrant paths



18

Copyright © 2011 M. E. Kabay. All rights reserved.

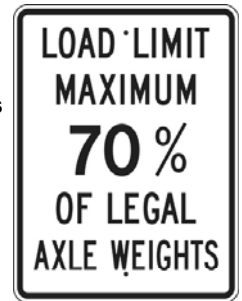
## Race Conditions

- Race condition exists when processes depend on precise timing for correct operation – but have no control over timing
- Typical examples
  - ❑ Lost update problem
  - ❑ Deadly embrace in locking
- May not notice errors because overlap may be very short and so occur very rarely
  - ❑ E.g., if update takes 10 ms but there are only 100 updates per minute, unlikely that two users will try to update same record simultaneously



## Load Conditions

- Resources are always limited
  - ❑ Storage
  - ❑ Numbers of users
  - ❑ Total number of transactions
  - ❑ Throughput
- Consider
  - ❑ High volume (total work)
  - ❑ High stress (maximum work in specified period)



## Resource Exhaustion

- System resources can be used up
  - ❑ CPU
  - ❑ RAM
  - ❑ Disk storage
  - ❑ OS tables
  - ❑ Semaphores
- E.g., inadequate RAM
  - ❑ May cause thrashing
  - ❑ Main memory constantly written to virtual memory on disk and back
  - ❑ Slows throughput to rate of disk I/O instead of RAM and bus speeds
    - ✓ Typically at least 1000 times slower



## Interapplication Conflicts

- Application program interfaces (APIs) can change as programs and operating systems change
- Inconsistencies can develop between versions of operating system, utilities and application programs
- Must keep up to date on changes and adapt to evolving programming environment



## Other Technical Errors

- Interactions with devices
  - ❑ Ignoring error codes
  - ❑ Using busy or missing devices
- Must recover gracefully from abnormal conditions
- Incorrect program compilation (builds)
  - ❑ Using old library components
  - ❑ Using wrong library (e.g., test version with debugging code active)



## Regulatory Compliance Considerations

- Some legislation such as Sarbanes-Oxley (SOX) may require careful records of all errors and their remediation
  - ❑ Must demonstrate *due care and diligence* in preventing harm
  - ❑ Document existence and proper operation of internal controls
- Failure to keep records may put organization in serious legal difficulties



## User Interface Errors

- Overview of User Interface
- Functionality
- Control (Command) Structure
- Performance
- Output Format



Copyright © 2011 M. E. Kabay. All rights reserved.

25

## Overview of User Interface

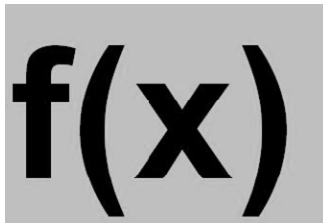
- UI: all aspects of system relevant to user
  - ❑ UVM: *user virtual machine*
  - ❑ Screens, mouse movements, keyboard functions, print outputs, sounds, colors....
- Problems arise when developers fail to consider users' perspective
  - ❑ "Who could possibly have thought of doing *that???*"
  - ❑ "Why can't the \*\*\*\* users figure that out themselves???"
- Documentation is essential to support users
  - ❑ But also for management and audit purposes

Copyright © 2011 M. E. Kabay. All rights reserved.

26

## Functionality Problems (1)

- Confusing, awkward, difficult, impossible
- Function is missing
- Undocumented features
- Required information is missing
- Program fails to confirm / respond to valid input
- Errors in output
- Conflicting names for features
- Too much information

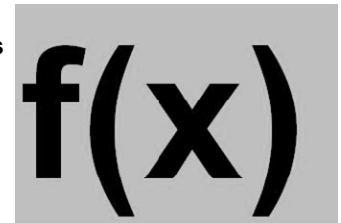


Copyright © 2011 M. E. Kabay. All rights reserved.

27

## Functionality Problems (2)

- Cursor disappears or appears in wrong place
- Screen displays are wrong
- Instructions difficult to find or read
- Identical functions require different operations
- Screens don't match expected format
- Passwords or other confidential data unprotected
- Impossible to trace data entry or changes (bad audit trails)
- Segregation of duties not enforced



Copyright © 2011 M. E. Kabay. All rights reserved.

28

## Control (Command) Structure Errors

- Sequence of operations determined by control structure
- Errors can confuse users or cause data loss; e.g.,
  - ❑ Impossible to move between menus
  - ❑ Confusing, repetitive, contradictory menus
  - ❑ Inadequate command-line entries
  - ❑ Requiring non-intuitive command-line entries
  - ❑ Deviating from operating system conventions
  - ❑ Failure to show correct error messages
  - ❑ Contradicting standard keyboard function keys
  - ❑ Omitting required commands
  - ❑ Violating privacy and other security constraints

Copyright © 2011 M. E. Kabay. All rights reserved.

29

## Performance

- Quality of service (QoS) or service-level agreements (SLA) may define required throughput
  - ❑ Maximum response times
  - ❑ Minimum data throughputs per unit of time
  - ❑ Minimum transaction rates
- Rigid designs may impede rapid response to changing requirements
  - ❑ E.g., programming business rules in code instead of in database metadata

Copyright © 2011 M. E. Kabay. All rights reserved.

30

## Output Format

- Printed or screen outputs must be controllable by user
- Fonts
- Emphasis (bolding, underlining, italics...)
- Spacing
- Tables, graphs, figures....
- Precision of numeric data
- Output device

31

Copyright © 2011 M. E. Kabay. All rights reserved.

## Designing SW Test Cases

- Good Tests
- Emphasize Boundary Conditions
- Check All State Transitions
- Use Test-Coverage Monitors
- Seeding
- Building Test Data Sets

32

Copyright © 2011 M. E. Kabay. All rights reserved.

## Good Tests

- Impossible to test program to perfection
  - ❑ Too costly and lengthy
- But good testing finds *most* problems
  - ❑ Emphasize eliminating *serious* errors
- Full-disclosure debate
  - ❑ Should researchers announce bugs and vulnerabilities to the world immediately?
  - ❑ Or should they tell developers first and give time to fix problems?
- Equivalence classes important
  - ❑ Define tests that can be considered equivalent
  - ❑ Examine design
  - ❑ E.g., a program may treat all data < boundary same → an equivalence class “less than lower limit”

33

Copyright © 2011 M. E. Kabay. All rights reserved.

## Emphasize Boundary Conditions

- Especially important to check boundary data
  - ❑ Below, at and above boundaries
- Also test using different user categories
  - ❑ Admin
  - ❑ Root / super-user
  - ❑ Data entry
  - ❑ Read-only
- Be especially careful to test for buffer overflows
  - ❑ Widely used by criminal hackers
  - ❑ Can insert code into stack of interpreter in long data input strings
  - ❑ Execute unauthorized code

34

Copyright © 2011 M. E. Kabay. All rights reserved.

## Check All State Transitions

- Every change in data constitutes a state transition
- Map probable state transitions
  - ❑ Transition probability matrix: from A to B
  - ❑ Menu maps show exactly where user can go in program from each menu
- May not be able to test all possible transitions
  - ❑ But can test most likely transitions first
- Test every limit
  - ❑ Tools available for certain applications such as Web code
- Test for race conditions
  - ❑ Use multiple clients executing automated scripts

35

Copyright © 2011 M. E. Kabay. All rights reserved.

## Use Test-Coverage Monitors

- Difficult to track execution of program during test to identify which modules being run
  - ❑ Primitive hand-coding writes log records
  - ❑ Call print function with parameter for each routine
  - ❑ Conditional compilation can allow debugging statements to remain or not
- Better: test-coverage monitors track every line of source code
  - ❑ Show reports of how often every line used
  - ❑ Thus can spot holes in testing
  - ❑ Or holes in logic – code impossible to reach
  - ❑ Or unauthorized code (Trojan horses, logic bombs)

36

Copyright © 2011 M. E. Kabay. All rights reserved.

## Seeding

- Add known bugs to program
- Run test
- See if you catch the bugs
- Can also estimate approximate *rate* of capture by looking at proportion of known bugs spotted

37

Copyright © 2011 M. E. Kabay. All rights reserved.

## Building Test Data Sets

- DO NOT USE ACTUAL PRODUCTION DATA IN TESTING!
- You may extract sample data sets from production data for testing
  - ❑ But keep confidentiality considerations in mind
  - ❑ May have to anonymize or randomize some fields (e.g., for Health Insurance Portability and Accountability Act – HIPAA)
- Ideally, use completely separate test system
  - ❑ Can use historical data with due attention to security / privacy
  - ❑ Also supports compliance issues (HIPAA, SOX...)

38

Copyright © 2011 M. E. Kabay. All rights reserved.

## Before Going Into Production

- Regression Testing
  - ❑ Testing everything you have done before
  - ❑ Done after every change
- Automated Testing
  - ❑ Repetitive testing difficult for people to monitor effectively (eyes glaze over)
  - ❑ Automated testing far more efficient
  - ❑ Can keep track of all errors
  - ❑ Provide detailed report
- Tracking Bugs from Discovery to Removal
  - ❑ Fix the bug
  - ❑ Find out *why* there was a bug
  - ❑ Fix the underlying *cause(s)* of the bug

39

Copyright © 2011 M. E. Kabay. All rights reserved.

## Managing Change

- Change Request
  - ❑ Organized documentation of who wants what why when
  - ❑ Allows prioritization
- Tracking System
  - ❑ Make sure that bugs are not forgotten
  - ❑ Also gather statistics about where bugs are being found and of what type – helps diagnosis
  - ❑ Essential for continuous process improvement
- Regression Testing – as mentioned, always required after every bug fix
- Documentation
  - ❑ Essential for continued smooth operations
  - ❑ Required for compliance with legal, regulatory standards

40

Copyright © 2011 M. E. Kabay. All rights reserved.

## Sources of Bugs & Problems

- Design Flaws
- Implementation Flaws
- Unauthorized Changes to Production Code
- Insufficient or Substandard Programming Quality
- Data Corruption
- Criminal Hacking

41

Copyright © 2011 M. E. Kabay. All rights reserved.

## Design Flaws

- Poor communication between users and designers
- Keep good documentation throughout development process
  - ❑ Helps identify breakdown in communication
- Trying to meet unrealistic delivery schedules
  - ❑ Managers must resist pressure to rush
  - ❑ All of the development stages are important
  - ❑ Rushing may lead to serious errors later

42

Copyright © 2011 M. E. Kabay. All rights reserved.

## Implementation Flaws



- Time pressure a major problem
- Developers, tester skimp on documentation and testing
- End up with unrecognized flaws
- Allocate sufficient time to avoid blunders
- Rule of thumb: 60% of software project should be devoted to TESTING

43

Copyright © 2011 M. E. Kabay. All rights reserved.

## Unauthorized Changes to Production Code



- Any unauthorized change to production code is a serious violation of standard policies
- Should be viewed as sabotage
- Critically important to determine who did it and why
- Find out how programmer got access to production code

44

Copyright © 2011 M. E. Kabay. All rights reserved.

## Insufficient or Substandard Programming Quality



- Testing and support records may identify who causes most of the problems in code
- Or possible that entire team needs coaching – or replacement
- Screen programmers carefully before placing on critical projects
- Use independent team for QA

45

Copyright © 2011 M. E. Kabay. All rights reserved.

## Data Corruption



- Logical corruption
  - ❑ Poor programming
  - ❑ Invalid data entry
  - ❑ Bad locking during concurrent access
  - ❑ Illegal access to other process' data stack
- Physical corruption
  - ❑ Hardware failure
- Analyze all data corruption cases thoroughly
  - ❑ Document
  - ❑ Fix underlying problems

46

Copyright © 2011 M. E. Kabay. All rights reserved.

## Criminal Hacking



- Audit trails (logs) essential tool for analysis of problems
  - ❑ Conforming to regulatory & legal requirements
  - ❑ Incident response
  - ❑ Research
- Policies
  - ❑ Archiving – how long to keep?
  - ❑ Security – e.g., chained checksums, encryption
  - ❑ Intrusion detection systems

47

Copyright © 2011 M. E. Kabay. All rights reserved.

# DISCUSSION



48

Copyright © 2011 M. E. Kabay. All rights reserved.