



Dependable Software Systems

A Taxonomy of Bugs

Material drawn from [Beizer]

Courtesy Spiros Mancoridis

Dependable Software Systems (Taxonomy)



The Importance of a Bug

- A reasonable metric for bug importance is:
- **importance** = *frequency* * (*correction_cost* + *installation_cost* + *consequential_cost*)



The Importance of a Bug (Cont'd)

- **Frequency**: How often does the bug occur?
- **Correction Cost**: What does it cost to discover and correct a bug?
- **Installation Cost**: Depends on the number of installations. Cost is small for single user systems, large for widely used systems (*e.g.*, Windows XP)
- **Consequences**: Depends on what kind of awards are made to the victims of the bug.



How Bugs affect Us?

- Bug consequences range from mild to catastrophic.
- Bug consequences should be measured in human rather than machine terms.



How Bugs affect Us? (Cont'd)

- **Mild**: *e.g.*, a misspelled output.
- **Moderate**: *e.g.*, bug impacts performance.
- **Annoying**: *e.g.*, bills for \$0.00 are sent.
- **Disturbing**: *e.g.*, ATM won't give you money because of bug.
- **Serious**: *e.g.*, Your paycheck transaction is not recorded.
- **Very Serious**: *e.g.*, Your paycheck is deposited to the wrong account.
- **Catastrophic**: *e.g.*, Bug starts a war.



Relative Frequency of Bugs

- Based on data from many sources, about 7 million lines of code.
 - Requirements: 24.3%
 - Structural Bugs: 25.2%
 - Data: 22.4%
 - Implementation and Coding: 9.9%
 - Interface and Integration: 9.0%
 - System, Software Architecture: 1.7%
 - Test Definition and Execution: 2.8%
 - Other (unspecified): 4.7%



Requirements Specification Bugs

- Requirement specifications can be:
 - incomplete
 - ambiguous
 - self-contradictory
 - misunderstood
 - a moving target
- Bugs in requirements are the earliest to invade the system and the latest to leave.



Requirements Validation

- *Do the stated requirements define the system that the customer really wants?*
- The most expensive mistakes to correct are the ones that occur earliest in the development process.
- Need to spend some time validating the requirements.
- Prototyping and other forms of automated modeling or “animation” often used.



Requirements Validation: What to Look for

- Do the requirements match what the customer asked for?
- Are there any ideas other than what the customer asked for?
- Are the requirements consistent?
- Are the requirements complete? Have important parts been left unspecified?



Requirements Validation: What to Look for (Cont'd)

- Is everything do-able? Is it practical?
- Will resource usage be reasonable?
- Are there unnecessary features that might best be left out?



Requirements Reviews

- *Must hold regular reviews while requirements definition is being formulated.*
 - should involve developers, customer, and (if appropriate) users
 - may be formal or informal
 - complete vs incomplete documents
 - full vs partial examination
 - good communication skills are very important!
 - iterate, iterate, iterate!

Dependable Software Systems (Taxonomy)



Requirements Reviews (Cont'd)

- Can a requirement be tested once implemented:
 - for presence?
 - for correct implementation?
- Is it understood? Is suitable detail provided?



Questions About Requirements

- Where did it come from?
- Who wants it?
- Why is it there?
- Is it still required?
- Is it amenable to change?
- How likely is it to change?
- What impact would changing it have?
- Ought it be allowed to change?



Structural Bugs

- Control and Sequence bugs
- Logic bugs
- Processing bugs
- Initialization bugs
- Data-Flow bugs



Examples of Control and Sequence Bugs

- paths left out
- unreachable code
- improper nesting of loops
- incorrect loop termination criteria
- missing processing steps
- duplicated processing
- unnecessary processing



Control and Sequence Bugs

- are amenable to theoretical treatment
- are less common when modern programming languages are used
- are more common among novice programmers
- dominate old code written in languages like COBOL and assembly
- are caught by structural testing (path testing in particular)
- result from attempts to optimize code



Examples of Logic Bugs

- non-existent cases
- “impossible” cases that are not impossible
- “don’t care” cases that matter
- improper negation of Boolean expressions
 - *e.g.*, using $>$ as the negation of $<$
- improper simplification and combination of cases
- overlap of exclusive cases



Examples of Logic Bugs (Cont'd)

- confusing OR with XOR
- misunderstanding the semantics of the order in which Boolean expressions are evaluated for specific compilers (*e.g.*, short circuits).



Logic Bugs

- Logic bugs are like arithmetic bugs.
- However logic bugs are more common because people tend to have a better training in arithmetic than they do in logic.
- Logic-based testing is the best defense against logic bugs.



Examples of Processing Bugs

- arithmetic bugs
- mathematical function evaluation
- algorithm selection
- conversion from one data representation to another
- ignoring overflow
- floating point number comparisons
- These bugs are frequent (12%), but have localized effects and tend to be caught by unit testing.

Dependable Software Systems (Taxonomy)



Examples of Initialization Bugs

- forgetting to initialize variables before they are used
- accepting an initial value without a validation check
- initializing to the wrong format, data representation, or type
- a bug in the first value of a loop-control parameter



Remedies for Initialization Bugs

- good programming languages
- tools
- data-flow testing



Examples of Data-Flow Bugs

- using an uninitialized variable
- attempting to use a variable before it exists
- modifying data and then not storing or using the result
- initializing a variable twice without intermediate use



Remedies for Data-Flow Bugs

- good programming languages
- tools
- data-flow testing



Data Bug

- Data bugs include all bugs that arise from the specification of data entities and their:
 - format
 - number
 - initial values
- Bugs in data are at least as common as bugs in code.
- Data bugs account for roughly 25% of all bugs in a typical software system.



Bugs in Dynamic Data

- Dynamic data are transitory and hence tend to produce bugs that are difficult to catch.
- Data is often gone before the bug's symptoms are discovered.
- Also problems arise when “garbage” data is left in shared resources such as invalid data in arrays and pointers to “freed” blocks of memory.



Implementation and Coding Bugs

- We do not consider syntax errors to be coding bugs. Syntax errors are caught by the compiler.
- Coding bugs are often typographical (*e.g.*, use number 1 instead of letter I).
- Common coding mistakes are due to misunderstanding the operation and side-effects of an instruction or statement.



Implementation and Coding Bugs

(Cont'd)

- How about erroneous comments?
- Testing techniques won't catch documentation bugs.
- Documentation bugs might mislead programmers and cause new bugs ...



Interface and Integration Bugs

- Interface Bugs
 - External Interface Bugs (UI)
 - Internal Interface Bugs (API)
- Integration Bugs



External Interfaces

- The “world” communicates with a system through an external interface.
- The “world” includes:
 - people
 - devices
 - sensors
 - communication lines, ...
- The primary design criterion for an external interface should be **robustness**.
- Each external interface employs a protocol.



External Interfaces (Cont'd)

- Protocol errors and misunderstandings:
 - Protocols may be complicated or hard to understand.
 - Protocols, especially new ones, may be wrong or implemented incorrectly.
- Invalid timing assumptions.
- Misunderstood input and output formats.
- Insufficient tolerance to bad input data.



Internal Interfaces

- In principle, internal interfaces are not different from external interfaces.
- Internal interfaces have the same problems with external interfaces, as well as problems resulting from:
 - wrong subroutine call sequences
 - misunderstood call-parameters
 - misunderstood entry or exit parameter values



Internal Interfaces (Cont'd)

- The remedy is good design and well-defined standards.
- There is a trade-off between:
 - the number of interfaces
 - the complexity of the interfaces.



Integration Bugs

- **Integration bugs** are bugs having to do with the integration of presumably working and tested components.
- Most of these bugs result from inconsistencies or incompatibilities between components.
- All communication methods can host integration bugs:
 - transfer data (in)directly between components
 - components share data



Integration Bugs (Cont'd)

- Communication methods include:
 - data structures
 - call sequences
 - semaphores ...
- While integration bugs do not constitute a large bug category (9%) they are an expensive category because:
 - they are caught late in the game
 - they force changes in several components and/or data structures.



Remedies for Integration Bugs

- Domain testing
- Data-flow testing



System and Architecture Bugs

- Many architecture bugs depend on the system load, and their symptoms emerge only when the system is stressed.
- Architecture bugs are difficult to find and repair.
- Architecture bugs are usually based on incorrect assumptions:
 - there will be no interrupts
 - that code is re-entrant (non self-modifying during execution)
 - that memory or registers are initialized ...



Remedies for Architecture Bugs

- Careful integration of modules.
- Subjecting the final system to a brutal stress test.



Test Definition/Execution Bugs

- Testers do not have immunity to bugs.
- Tests often require code.
- Testing code can be incorrect.



Remedies for Testing Bugs

- **Test Debugging:** Easier than program debugging because test programs tend to be smaller and less constrained than ordinary programs.
- **Test Execution Automation:** Manual testing is fault-prone and inefficient.
- **Test Design Automation:** Difficult, but becoming more of a reality with research results in software testing.



Other Types of Testing ...

- **AGRESSION TESTING:** If this doesn't work, I'm gonna kill somebody.
- **CONFESSION TESTING:** Okay, Okay, I *did* program that bug.
- **CONGRSSIONAL TESTING:** Are you now, or have you ever been a bug?
- **DEPRESSION TESTING:** If this doesn't work, I'm gonna kill myself.



Other Types of Testing ...

- **REGRESSION TESTING:** Uh-oh, a bug... I'm outta here.
- **DIGRESSION TESTING:** Well, it works, but can I tell you about my truck...
- **EXPRESSION TESTING:** #@%^&*!!!
bug.
- **OBSESSION TESTING:** I'll find this bug if it's the last thing I do.



Other Types of Testing ...

- **OPRESSION TESTING:** Test this now!
- **POISSION TESTING:** Alors! Regardez le poisson!
- **REPRESSSION TESTING:** It's not a bug, it's a feature.
- **SECCESSION TESTING:** The bug is dead! Long lives the bug!
- **SUGGESTION TESTING:** Well, it works but wouldn't it be better if...