Empirical Studies on Bugs

Overview

- General introduction of empirical studies in SE
- Paper presentation and discussion
  - How Do Fixes Become Bugs?[2]
  - Programmers' Build Errors: Case Study (at Google) [3]
- Threats to Validity

2

Empirical Research [1]

- Research using empirical evidence. It is a way of gaining knowledge by means of direct or indirect observation or experience.
- Empirical evidence can be analyzed quantitatively or qualitatively
- Researcher answers empirical questions, which should be clearly defined and answerable with the evidence collected

3

Empirical Studies in SE

- To understand how developers build or maintain software by observing various software artifacts or monitoring software runtime behaviors
- Can be conducted with manual inspection or automatic tools
- May achieve various research goals:
  - identify software change patterns
  - reveal relations between symptoms and root causes
  - shed light on new technique design and impl.

4

Characteristics of Empirical Studies

- Cool algorithm design or intensive programming is NOT always required
  - Sometimes only manual inspection and eyeball checking is done
- "Interesting Research Questions" is the key contribution
  - Questions haven't been asked or answered nicely
  - Questions whose answers can provide actionable advice to tool builders or users

5

How Do Fixes Become Bugs? [2]

A Comprehensive Characteristic Study on Incorrect Fixes in Commercial and Open Source Operating System

Problem Statement

- Bug fixes can be wrong, and thus fail to fix the bugs or introduce new bugs
- Incorrect bugs fixes can cause serious problems
  - Trend Micro lose $8 million for a buggy patch
- Various reasons can cause buggy fixes
  - Time pressure, lack of knowledge
- What is the characteristic of incorrect bug fixes?

7

Research Questions

- How significant is the problem of incorrect fixes?
  - % of incorrect fixes, severity of caused problems
- What types of bugs are difficult to fix correctly?
- What are the common mistakes made in bug fixes?

8

Research Questions

- What aspects in the development process are correlated to the correctness of bug fixing?
  - Does fixers and reviewers' knowledge is highly correlated to incorrect fixes?

9

Methodology

- Four software projects under study
  - One commercial OS, three open source OS (FreeBSD, Linux, OpenSolaris)
- Randomly select a target number, e.g., 2000, of bug fixes, and only examine the 970 post-release bugs
- Find incorrect fixes
  - identify bug fixing changes
  - identify incorrect ones
- Measure code knowledge

10

Finding Bug Fixing Changes

- The commercial OS and OpenSolaris well maintain the link between a fix and a bug report
- For the other two OS, if the commit message or bug report contains words like "the bug is fixed by change 0a134fad", the bug is associated with the change 0a134fad

11

Finding Incorrect Bug Fixes

- Step 1: Automatic filtering
  - If two bug fixes have source code overlap, the latter one is to correct the former one
  - If the bug report of Bug A contains any reference to Bug B, the fix for Bug B is incorrect
- Step 2: manual examination and some discussion with developers

12

Measuring Code Knowledge

- Heuristic: calculate knowledge of developers for code based on their contribution for the code base

$$K\_File_{d,F,v} = \frac{\text{The LoC written by } d \text{ for } F \text{ at } v}{\text{The total LoC in } F \text{ at } v}$$

$$K\_Func_{d,f,F,v} = \frac{\text{The LoC written by } d \text{ for } f \text{ in } F \text{ at } v}{\text{The total LoC in the } f \text{ at } v}$$

13

RQ1: How significant is the problem of incorrect fixes?

App	# of post-release bug fixes	# of incorrect fixes	Ratio
A	189	39	20.6%±3.0%
B	309	46	14.8%±2.9%
C	267	41	15.3%±2.6%
D	205	50	24.4%±3.7%

- 14.8-24.4% bug fixes are incorrect. Since the fixes should be applied by a lot of customers and users, this ratio can have significant impact.

14

Impact of the Incorrect Bug Fixes

- 14% introduce crash
- 8.4% cause system to hang
- 15.4% lead to data corruption or data loss
- 5.6% cause security problem
- 7% degrade the performance
- 45.1% introduce incorrect functionality

15

RQ2: What types of bugs are difficult to fix correctly?

- 970 bug fixes are classified into three categories: memory bugs, concurrency bugs, and semantic bugs

App	Concurrency	Memory	Semantic
A	4/13 (31%)	3/17 (18%)	32/159 (20%)
B	9/21 (43%)	5/44 (13%)	32/244 (13%)
C	7/19 (37%)	6/43 (14%)	28/205 (14%)
D	10/23 (44%)	5/30 (17%)	35/152 (23%)
Overall	30/76 (39%)	19/134 (14%)	127/760 (17%)

- Developers and testers should be more cautious when fixing concurrency bugs

16

The Most Observed Detailed Bug Types

Bug types and their percentages			
data race	33%	deadlock	29%
buffer overflow	8%	memory leak	6%
uninitialized read	4%	null pointer deref	4%

- Q: Are these bugs generally difficult to fix? Or is this observation unique to incorrect bug fixes?

17

RQ3: What are the common mistakes made in bug fixes?

- data races
  - When adding locks, some methods should not be put into the critical section
- deadlock
  - When reversing the order of locks or dropping some locks, all execution paths should be tested

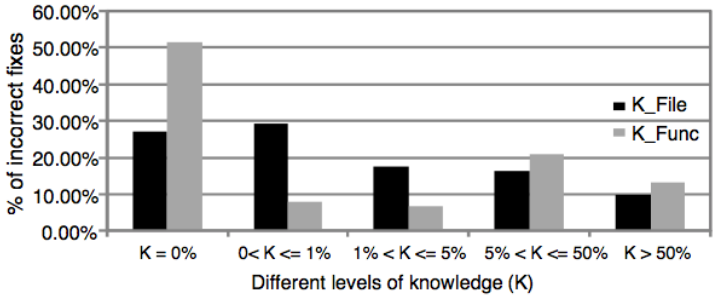
18

Common Mistakes in Bug Fixing

- buffer overflow
  - Increase buffer size or allocate larger buffer without checking
- memory leak
  - Fixing memory leak can introduce dangling pointer or null pointer dereference
- semantic bugs
  - Wrongly changed conditions

19

RQ4: Correlation between developers' knowledge and incorrectness



- The more knowledge a fixer has, the fewer incorrect bug fixes he commits

20

Conclusion

- This paper presents one of the most comprehensive characteristic studies on incorrect bug fixes
- It studies the common patterns of mistakes made in incorrect fixes
- 27% incorrect bug fixes are made by developers who haven't contributed a single line to the file under fix

21

Programmers' Build Errors: A Case Study (at Google) [3]

Problem Statement

- Building is important when developers use compilers, linkers, build files and scripts to assemble their code into executable units
- Slow and failed builds can affect programmer productivity
- Keeping the build process fast and understanding when and how it fails is essential to improve productivity

23

Google's Build Environment

- Google's cloud-based centralized build system
- BUILD files specify the build configuration
  - the list of targets (tests) to build
  - the sources and dependent targets to compile
  - the compilers to use
- Developers may build with the IDE's own compiler during development, but must ensure the code successfully builds with the centralized system

24

Research Questions

- How often do builds fail?
  - By reporting the failure-to-success build ratio, the findings help us understand the potential impact of changes
- Why do builds fail?
  - The findings can motivate or prioritize new automatic checking
- How long does it take to fix broken builds?
  - The findings highlight hard-to-fix errors

25

Methodology

- The build system saves all build logs describing
  - The result of each build (succeeded or failed)
  - The compilation errors
- Examine build logs in nine months for Java and C++ projects
- Focus on builds done by standard developers
  - Active developers without requesting automatic builds

26

How to Count Build Errors?

- Two ways investigated
  - Count-all: # of all error messages in one build
  - Count-distinct: # all unique error kinds in one build
- The second one is better
  - As one fault, such as missing import, can cause a lot of name resolution errors, the error should be counted only once

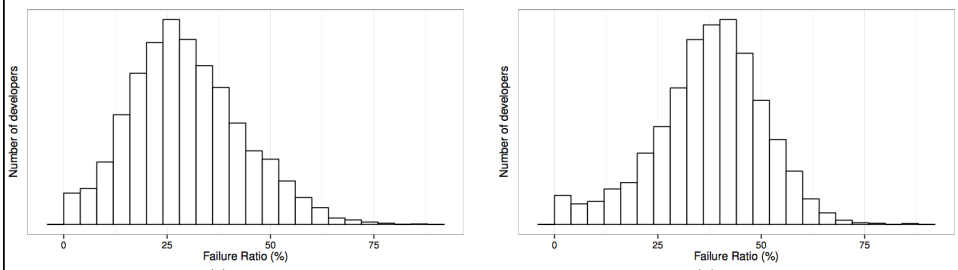
27

How to Calculate Resolution Time?

- If a developer has a series of failure builds before a successful one, and the failure only contains one error
  - Resolution time = $Time_{success} - Time_{1stFailure}$

28

RQ1: How often do builds fail?



(a) Java (b) C++

- The failure ratio is approximately normally distributed
- The median percentage is 38.4% for C++ and 28.5% for Java

29

RQ2: Why do builds fail?

- Normalize the errors
- Cluster them to report the top five errors, which constitute 80% of all errors

30

Categories of Error Messages - Java

Category	Name	Description
Dependency	cant.resolve	cannot find symbol
	doesnt.exist	package does not exist
	strict	Google-specific dependency check
Type mismatch	cant.apply.symbol	method called with incorrect argument list
	cant.apply.symbol.1	method called with incorrect argument list
	incompatible.types	incompatible types
	unchecked	unchecked conversion from rawtype to parameterized type
	cant.apply.symbols	method called with incorrect argument list
	rawtypes	use of raw type rather than parameterized type
	operator.cant.be.applied	wrong types for operator
Syntax	expected	certain keyword or symbol was expected and not found
	illegal.start.of.expr	illegal start of expression
	not.stmt	statement expected but not found
	missing.ret.stmt	missing return
	illegal.start.of.type	illegal start of type
	expected3	certain keyword or symbol was expected and not found
Semantic	method.does.not.override.superclass	method has @Override annotation but does not override a superclass or interface method
	does.not.override.abstract	concrete class must override abstract method in superclass or interface
	non-static.cant.be.ref	non-static cannot be referenced from static context
	report.access	access disallowed by visibility modifier
	static.imp.only.classes.and.interfaces	cannot static import this symbol
	not.def.public.cant.access	symbol is not public and cannot be accessed
Other	unreported.exception.need.to.catch.or.throw	checked exception must be caught or thrown
	already.defined	symbol already defined
	var.might.not.have.been.initialized	variable might not have been initialized

31

Categories of Error Messages – C++

Category	Name	Description
Dependency	undeclared_var_use	use of an undeclared identifier
	no_member	no member (e.g. class variable) in the given class
	undeclared_var_use_suggest	use of an undeclared identifier, with a suggested fix
	unknown_typename_suggest	unknown typename, with a suggested fix
	unknown_typename	unknown typename
	no_member_overloaded_arrow	no member (e.g. class variable) in the given class, with a suggested fix
	member_decl_does_not_match	out of line definition does not match any declaration
	pp_file_not_found	could not find header file
	typename_nested_not_found	no type with given name within the specified class or namespace
	incomplete_member_access	member access into incomplete type
Type mismatch	ovl_no_viable_function_in_call	calling an undeclared version of an overloaded function
	typecheck_nonviable_condition	cannot convert
	ovl_no_viable_function_in_init	no matching constructor for the initialization expression
	init_conversion_failed	cannot initialize with the passed-in value
	ovl_no_viable_member_function_in_call	no matching member function for the call
	typecheck_member_reference_suggestion	using '->' instead of '.' or vice-versa, with a suggested fix
	typecheck_member_reference_arrow	using '->' instead of '.' or vice-versa
	typecheck_invalid_operands	invalid operands to binary expression
	typecheck_call_too_few_args	too few arguments to a function call
Syntax	expected_expression	expected an expression
	expected_rparen	expected right parenthesis
	expected_unqualified_id	expected an identifier
	bound_member_function	reference to a non-static member function
Semantic	access	accessing a protected or private member
Other	non_virtual_dtor	virtual functions but no virtual destructor

32

Why Do These Errors Happen?

- Dependency
  - missing dependency between source code components due to significant usage of IDE
- Type mismatch
  - assign a variable to an incompatible type or call a function with wrong argument types
- Syntax
- Semantics
  - violating class access rule or unimplemented abstract methods
- Q1: Why do the last three happen?

33

RQ3: How long does it take to fix builds? - Java

- Top 5 expensive errors (freq. * median resolution time)
  - cant.resolve, doesnt.exist, strict, cant.apply.symbol, cant.apply.symbol.1
- Error with most resolution time: does.not.override.abstract

34

RQ3: How long does it take to fix builds? – C++

- Top 5 expensive errors (freq. * median resolution time)
  - undeclared_var_use, undeclared_var_use_suggest, no_member, ovl_no_viable_function_in_init, typecheck_nonviable_condition
- Error with most resolution time: expected_rparen

35

Implications

- 10% of the error types account for 90% of the build failures
- Better tools to resolve dependency errors have the greatest potential payoff

36

Threats to Validity

Man prefers to believe what he prefers to be true.

-- Francis Bacon

37

Threats to Validity

- Is the investigator's conclusion correct?
- Try to identify the factors which make your conclusion incorrect

38

External & Internal Validity

- External validity
  - The degree to which the results of an empirical investigation can be generalized to and across individuals, settings, and times
    - "Is the conclusion generalizable?"
- Internal validity
  - The degree to which a causal conclusion based on a study is warranted
    - "Is the experiment done correctly?"

39

Threats to External Validity

- Aptitude
  - If a medicine is effective for sample patients, will it also be effective for non-volunteers or all other people?
- Situation
  - time, location, scope and extent of measurement

40

Threats to External Validity

- Pre-test effects
  - The cause-effect relationship can be found when pre-tests are carried out
- Post-test effects
  - The cause-effect relationship can only be found when post-tests are carried out
- …

41

Examples

- The empirical study is performed within a single company with particular processes, constraints, resources, and tools
- The empirical study in done on operating system software/open source projects

42

Threats to Internal Validity

- Confounding
  - Changes in the observation may be related to multiple variables
- Selection bias
  - Samples should be chosen without bias
- Instrument change
  - The measurement may affect the result
- John Henry effect
  - John Henry was a worker who outperformed a machine under an experimental setting because he was aware that his performance was compared with that of a machine.

43

Examples

- The execution time reading may significantly affect the measured execution time
- The causal-effect relationship between bugs and bad variable names may be affected by factors like complexity of functionality, maturity of developers, and types of bugs

44

Importance of Threat Identification

- Help researchers decide how to propose research questions and do experiments in a plausible way
- Help people understand limitation of the research
- It is OK that you can't avoid all threats. However, you should try your best to make your results representative and meaningful

45

Reference

[1] Empirical research, https://en.wikipedia.org/wiki/Empirical_research

[2] Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, Lakshmi Bairavasundaram, How Do Fixes Become Bugs? ESEC/FSE 2011

[3] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, Robert Bowdidge, Programmers' Build Errors: A Case Study (at Google), ICSE' 14

46