Prevalence of Single-Fault Fixes and its Impact on Fault Localization

Alexandre Perez, Rui Abreu, Marcelo d'Amorim

`alexandre.perez@fe.up.pt`, `rui@computer.org`, `damorim@cin.ufpe.br`

Motivation

- Coverage-based software fault localization is effective at pinpointing bugs when only one fault is being exercised.

Motivation

- Coverage-based software fault localization is effective at pinpointing bugs when only one fault is being exercised.
- Approaches that diagnose more that one fault have been proposed.
  - However, they involve computationally expensive tasks.
  - May require system modelling.

Motivation

- Coverage-based software fault localization is effective at pinpointing bugs when only one fault is being exercised.
- Approaches that diagnose more that one fault have been proposed.
  - However, they involve computationally expensive tasks.
  - May require system modelling.
- **In practice, how often are developers faced with fixing single faults versus multiple faults at once?**

Single-fault Diagnosis

Spectrum-based Fault Localization

- Given:
  - A set $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$ of M system *components*[1].
  - A set $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$ of N system tests with binary outcomes stored in the error vector e.
  - A $N \times M$ coverage matrix $\mathcal{A}$, where $\mathcal{A}_{ij}$ is the involvement of component c_j in test t_i.

$$
\begin{array}{c|cccc|c}
\mathcal{T} & c_1 & c_2 & \cdots & c_M & e \\
\hline
t_1 & \mathcal{A}_{11} & \mathcal{A}_{12} & \cdots & \mathcal{A}_{1M} & e_1 \\
t_2 & \mathcal{A}_{21} & \mathcal{A}_{22} & \cdots & \mathcal{A}_{2M} & e_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
t_N & \mathcal{A}_{N1} & \mathcal{A}_{N2} & \cdots & \mathcal{A}_{NM} & e_N
\end{array}
$$

[1] A component can be any source code artifact of arbitrary granularity such as a class, a method, a statement, or a branch.

Single-fault Diagnosis

Spectrum-based Fault Localization

- The next step consists in determining the likelihood of each component being faulty.
- A component frequency aggregator is leveraged:

$$n_{pq}(j) = |\{i \mid \mathcal{A}_{ij} = p \wedge e_i = q\}|$$

  - Number of runs in which c_j has been active during execution ($p = 1$) or not ($p = 0$), and in which the runs failed ($q = 1$) or passed ($q = 0$).
- Fault likelihood per component is achieved by means of applying different fault predictors.
- Components are then ranked according to such likelihood scores and reported to the user.

Fault Predictors

Tarantula

- Designed to assist fault-localization using a visualization.
- Intuition: components that are used often in failed executions, but seldom in passing executions, are more likely to be faulty.

Tarantula

$$\frac{\frac{n_{11}(j)}{n_{11}(j)+n_{01}(j)}}{\frac{n_{11}(j)}{n_{11}(j)+n_{01}(j)}+\frac{n_{10}(j)}{n_{10}(j)+n_{00}(j)}}$$

James A. Jones and Mary Jean Harrold. “Empirical Evaluation of the Tarantula Automatic Fault-localization Technique”. In: *20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2005, pp. 273–282

Fault Predictors

Ochiai

- Calculates the *cosine similarity* between each component's activity ($\mathcal{A}_j$) and the error vector (e).

Ochiai

$$\frac{n_{11}(j)}{\sqrt{n_{11}(j)+n_{01}(j)+}\sqrt{n_{11}(j)+n_{10}(j)}}$$

Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. "An Evaluation of Similarity Coefficients for Software Fault Localization". In: *12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), 18-20 December, 2006, University of California, Riverside, USA*. 2006, pp. 39–46

Fault Predictors

D^*

- The likelyhood of a component being faulty is:
  1. Proportional to the number of failed tests that cover it;
  2. Inversely proportional to the number of passing tests that cover it;
  3. Inversely proportional to the number of failed tests that do not cover it.
- D^* provides a $*$ parameter for changing the weight carried by term (1).

D^*

$$\frac{n_{11}(j)^*}{n_{01}(j)+n_{10}(j)}$$

W. Eric Wong et al. "The DStar Method for Effective Software Fault Localization". In: *IEEE Transactions on Reliability* 63.1 (2014), pp. 290–308

Fault Predictors

O

- Assuming there is only one fault in the system:
  - $n_{01}(j)$ should always be zero for the faulty component.
  - $n_{11}(j) + n_{01}(j)$ always equals the number of failing tests.
  - $n_{10}(j) + n_{00}(j)$ always equals the number of passing tests.
  - Only one degree of freedom left, expressed by assigning $n_{00}(j)$ as the predictor's value.
- Proven to be optimal under the single-fault assumption.

O

$$\begin{cases} -1 & \text{if } n_{01}(j) > 0 \\ n_{00}(j) & \text{otherwise} \end{cases}$$

Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. "A model for spectra-based software diagnosis". In: *ACM Trans. Softw. Eng. Methodol.* 20.3 (2011), p. 11

Fault Predictors

O^P

- Relaxes the assumptions held by the O predictor.
- Does not immediately assign $n_{01}(j) > 0$ a low score.

O^P

$$n_{11}(j) - \frac{n_{10}(j)}{n_{10}(j)+n_{00}(j)+1}$$

Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. "A model for spectra-based software diagnosis". In: *ACM Trans. Softw. Eng. Methodol.* 20.3 (2011), p. 11

Multiple-fault Diagnosis

- Fault predictors assign a one-dimensional score to each component in the system.
- May abstract away relevant information to properly score multiple-faulted systems.

Example

$\mathcal{T}$	c_1	c_2	e
t_1	1	0	fail
t_2	0	1	fail

Both c_1 and c_2 are faulty but are given a low O score.

Multiple-fault Diagnosis

- Several approaches were proposed to accurately diagnose multiple faults:
  - Model-based Debugging[2];
  - Spectrum-based Reasoning[3]; and
  - Debugging in Parallel[4].
- These approaches are computationally much more expensive and some partial modelling of the system may be required.

---

[2] Wolfgang Mayer and Markus Stumptner. "Model-Based Debugging - State of the Art And Future Challenges". In: *Electr. Notes Theor. Comput. Sci.* 174.4 (2007), pp. 61–82

[3] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. "Spectrum-Based Multiple Fault Localization". In: *24th IEEE/ACM International Conference on Automated Software Engineering, ASE*. 2009, pp. 88–99

[4] James A. Jones, Mary Jean Harrold, and James F. Bowring. "Debugging in Parallel". In: *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA*. 2007, pp. 16–26

Single-Fault Prevalence

How often are developers faced with the task of having to diagnose and fix multiple bugs?

Single-Fault Prevalence

How often are developers faced with the task of having to diagnose and fix multiple bugs?

Our hypothesis is that the majority of bugs are detected and fixed one-at-a-time when failures are detected in the system.

Single Fault Prevalence

Methodology

1. Mine repositories to collect fixing commits.
2. Classify fixing commits according to the number of faults they fix.

Mining Fixing Commits

- Reverse chronological analysis of commits in a repository.
- For any given commit I:
  - Run tests in I's source tree.
  - If the suite is passing, restore each parent commit P that only modifies existing components and run I's suite.
  - A runtime error means that there are functionality changes between the two source code versions.
  - A failing test suite reveals that I's suite has detected errors in P's source tree.
  - $\langle P, I \rangle$ is labeled as a faulty/fixing commit pair.

Classifying Fault Cardinality

Spectra Gathering

- Given a pair of faulty/fixing commits, run the fixing commit's test suite on faulty's source tree and gather the hit spectrum.

Example

$\mathcal{T}$	c_1	c_2	c_3	c_4	c_6	c_7	c_8	e
t_1	1	1	0	0	1	0	0	pass
t_2	0	1	1	0	1	1	0	fail
t_3	1	0	0	1	0	0	1	pass
t_4	0	0	1	0	0	1	0	fail
	Δ		Δ					

Classifying Fault Cardinality

Unchanged Code Removal

- All components not in Δ can be safely exonerated from suspicion.

Example

$\mathcal{T}$	c_1	c_2	c_3	c_4	c_6	c_7	c_8	e
t_1	1	1	0	0	1	0	0	pass
t_2	0	1	1	0	1	1	0	fail
t_3	1	0	0	1	0	0	1	pass
t_4	0	0	1	0	0	1	0	fail
	Δ		Δ					

Before.

$\mathcal{T}$	c_1	c_3	e
t_1	1	0	pass
t_2	0	1	fail
t_3	1	0	pass
t_4	0	1	fail

After.

Classifying Fault Cardinality

Passing Tests Removal

- Passing tests are discarded as they do not reveal information about faulty components.

Example

$\mathcal{T}$	c_1	c_3	e
t_1	1	0	pass
t_2	0	1	fail
t_3	1	0	pass
t_4	0	1	fail

Before.

$\mathcal{T}$	c_1	c_3	e
t_2	0	1	fail
t_4	0	1	fail

After.

Classifying Fault Cardinality

Hitting Set & Classification

- The final, filtered spectrum is subject to minimal hitting set analysis.
- Determine what (set of) components is active on every failing test.
- Cardinality of the hitting set corresponds to the number of faults.

Example

$\mathcal{T}$	c_1	c_3	e
t_2	0	1	fail
t_4	0	1	fail

$\{c_3\}$ is the minimal hitting set with cardinality 1.

Empirical Study

Setup

- We have applied our fault cardinality classification to several software projects.
- Subjects are open-source projects hosted on Github, gathered in the work of Gousios and Zaidman[5].
- The dataset was filtered so that considered projects
  - Are written in Java;
  - Are built using Apache Maven;
  - Contain JUnit test cases.
- In total we studied 279 subjects.

[5] Georgios Gousios and Andy Zaidman. “A Dataset for Pull-based Development Research”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. 2014, pp. 368–371

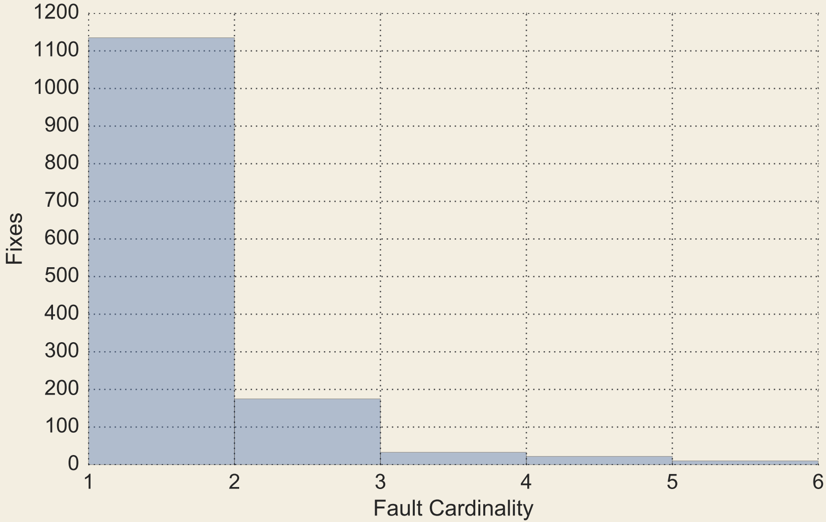
Empirical Study

Effort To Diagnose

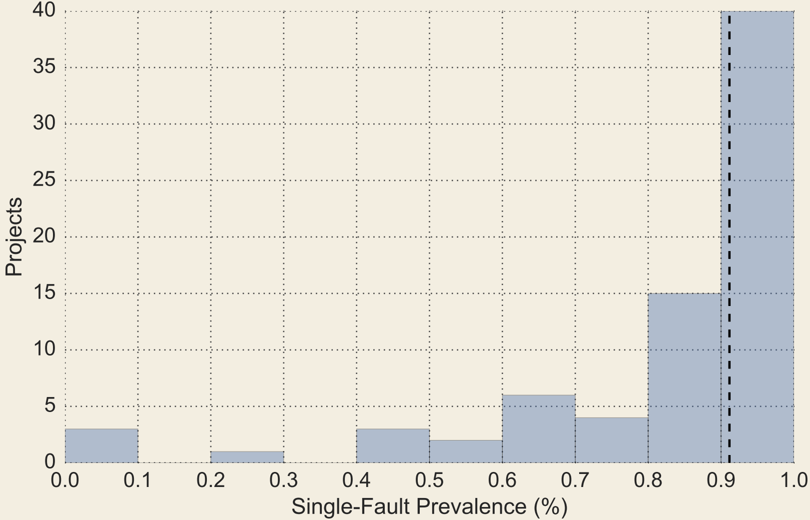
- To assess diagnostic performance, we resort to using the effort to diagnose metric.
- Also known as wasted effort.
- Since SFL outputs a ranked list of components sorted by predictor score, effort measures the average number of components to be inspected until the real faulty component is reached.
- Usually normalized by the total number of components in the system.

Friedrich Steimann, Marcus Frenkel, and Rui Abreu. “Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators”. In: *International Symposium on Software Testing and Analysis, ISSTA ’13, Lugano, Switzerland, July 15-20, 2013*. 2013, pp. 314–324

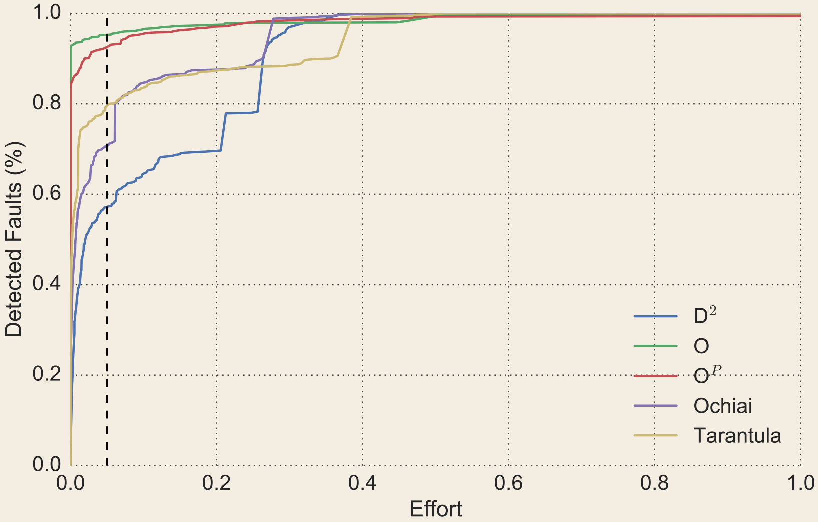
Fault Cardinality



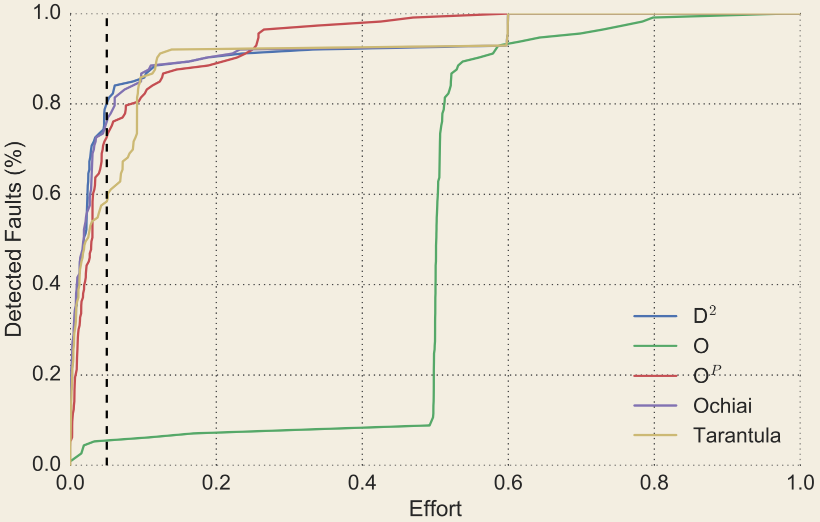
Single Fault Prevalence



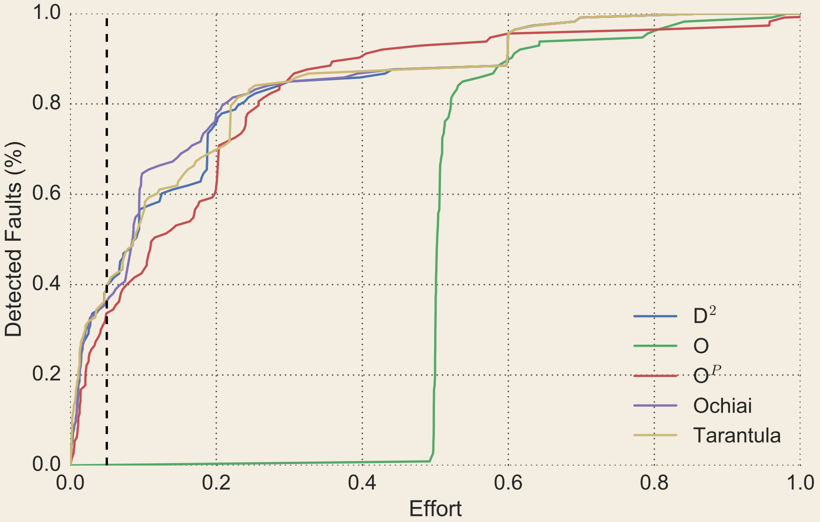
Effort To Diagnose Single Faults



Effort To Diagnose Multiple Faults – Best Case



Effort To Diagnose Multiple Faults – Worst Case



Conclusions

- Single-fault SFL is an inexpensive approach to fault localization, but does not take into account the possibility of failures due to multiple bugs.
- However, our hypothesis is that while software can have many dormant bugs, these are detected (and fixed) individually.
- Our empirical study found that 82.5% of the time, developers are faced with single faults.
- While the O predictor is theoretically optimal assuming a single faulted system, its diagnostic performance becomes random in the event of a multiple faults.
  - Other predictors are less sensitive to this issue.

Single-fault Diagnosis

Spectrum-based Fault Localization

- Given:
  - A set $\mathcal{C} = \{c_1, c_2, ..., c_M\}$ of M system *components*[1].
  - A set $\mathcal{T} = \{t_1, t_2, ..., t_N\}$ of N system tests with binary outcomes stored in the error vector e.
  - A $N \times M$ coverage matrix $\mathcal{A}$, where $\mathcal{A}_{ij}$ is the involvement of component c_j in test t_i.

$\mathcal{T}$	c_1	c_2	$\cdots$	c_M	e
t_1	$\mathcal{A}_{11}$	$\mathcal{A}_{12}$	$\cdots$	$\mathcal{A}_{1M}$	e_1
t_2	$\mathcal{A}_{21}$	$\mathcal{A}_{22}$	$\cdots$	$\mathcal{A}_{2M}$	e_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
t_N	$\mathcal{A}_{N1}$	$\mathcal{A}_{N2}$	$\cdots$	$\mathcal{A}_{NM}$	e_N

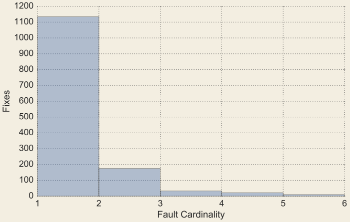
[1] A component can be any source code artifact of arbitrary granularity such as a class, a method, a statement, or a branch.

Single-Fault Prevalence

How often are developers faced with the task of having to diagnose and fix multiple bugs?

Our hypothesis is that the majority of bugs are detected and fixed one-at-a-time when failures are detected in the system.

Fault Cardinality



Conclusions

- Single-fault SFL is an inexpensive approach to fault localization, but does not take into account the possibility of failures due to multiple bugs.
- However, our hypothesis is that while software can have many dormant bugs, these are detected (and fixed) individually.
- Our empirical study found that 82.5% of the time, developers are faced with single faults.
- While the O predictor is theoretically optimal assuming a single faulted system, its diagnostic performance becomes random in the event of a multiple faults.
  - Other predictors are less sensitive to this issue.