History-based Test Case Prioritization with Software Version Awareness

Chu-Ti Lin, National Chiayi University, Taiwan

Cheng-Ding Chen, Industrial Technology Research Institute, Taiwan

Chang-Shi Tsai, National Chiayi University, Taiwan

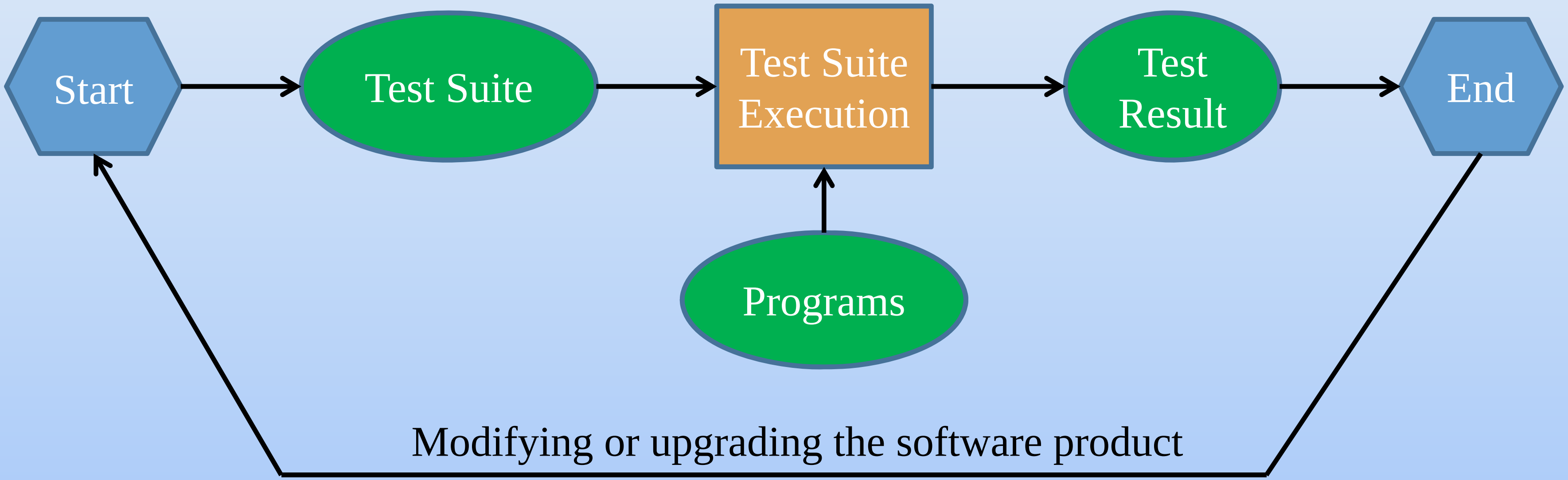
Gregory M. Kapfhammer, Allegheny College, USA

June 18, 2013

The 18th International Conference on Engineering of Complex Computer Systems

Introduction

- Regression testing
  - Regression testing is used to validate the modified software product.
  - Software engineers often reuse test suites in regression testing.



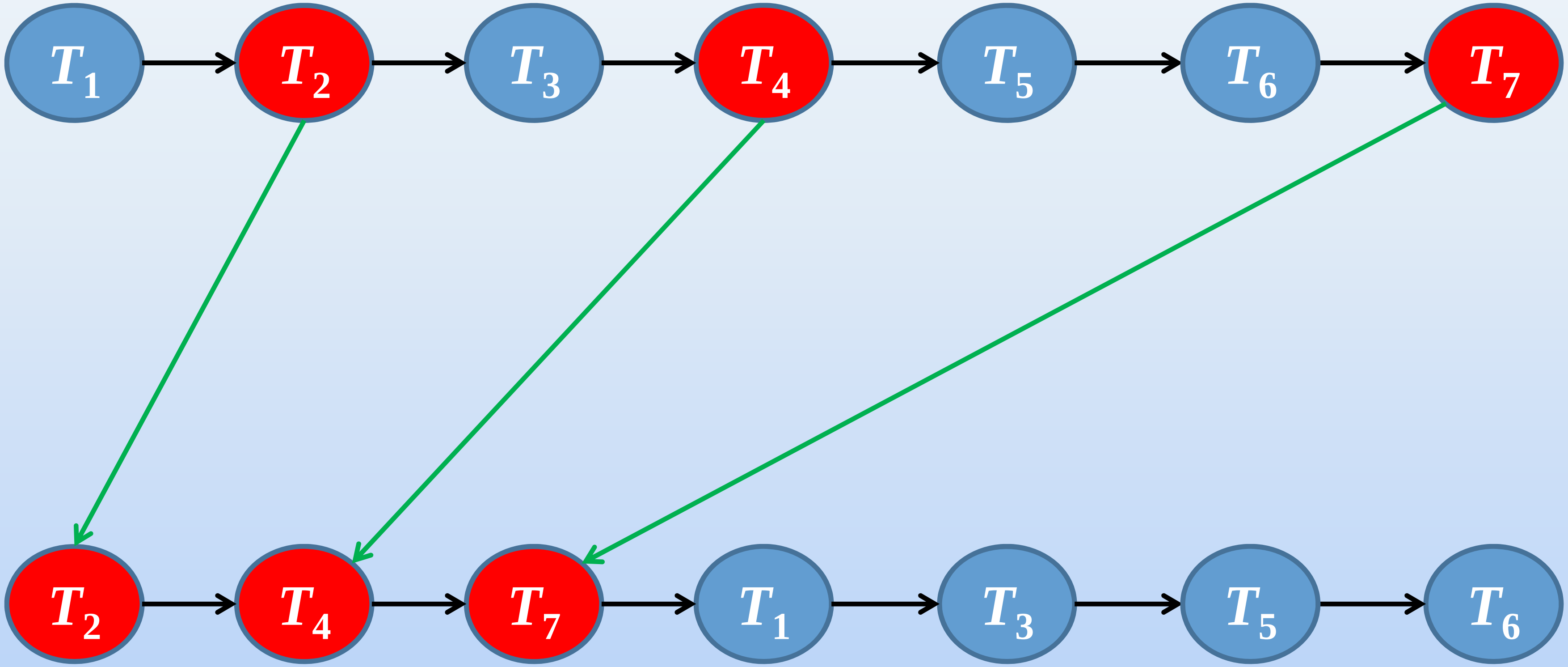
Test case prioritization

- Software developers can start to remove faults early if faults can be detected in early stage of testing.
- Scheduling the test cases in an order so that the tests with better fault detection capability are executed at an early position in the regression test suite.

Example for test case prioritization

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5 \rightarrow T_6 \rightarrow T_7$$

Example for test case prioritization



Criterion used to evaluate prioritization

- Average Percentage of Fault Detected per Cost (APFDc)

$$APFDc = \frac{\sum_{i=1}^{m} f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i}$$

  - f_i: fault severity of fault i

Criterion used to evaluate prioritization

- Average Percentage of Fault Detected per Cost (APFDc)

$$APFDc = \frac{\sum_{i=1}^{m} f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i}$$

  - f_i: fault severity of fault i
  - t_j: execution cost of test case j

Criterion used to evaluate prioritization

- Average Percentage of Fault Detected per Cost (APFDc)

$$APFDc = \frac{\sum_{i=1}^{m} f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i}$$

  - f_i: fault severity of fault i
  - t_j: execution cost of test case j
  - n: the number of test cases in the test suite

Criterion used to evaluate prioritization

- Average Percentage of Fault Detected per Cost (APFDc)

$$APFDc = \frac{\sum_{i=1}^{m} f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i}$$

  - f_i: fault severity of fault i
  - t_j: execution cost of test case j
  - n: the number of test cases in the test suite
  - m: the number of faults that are revealed by the test suite

Criterion used to evaluate prioritization

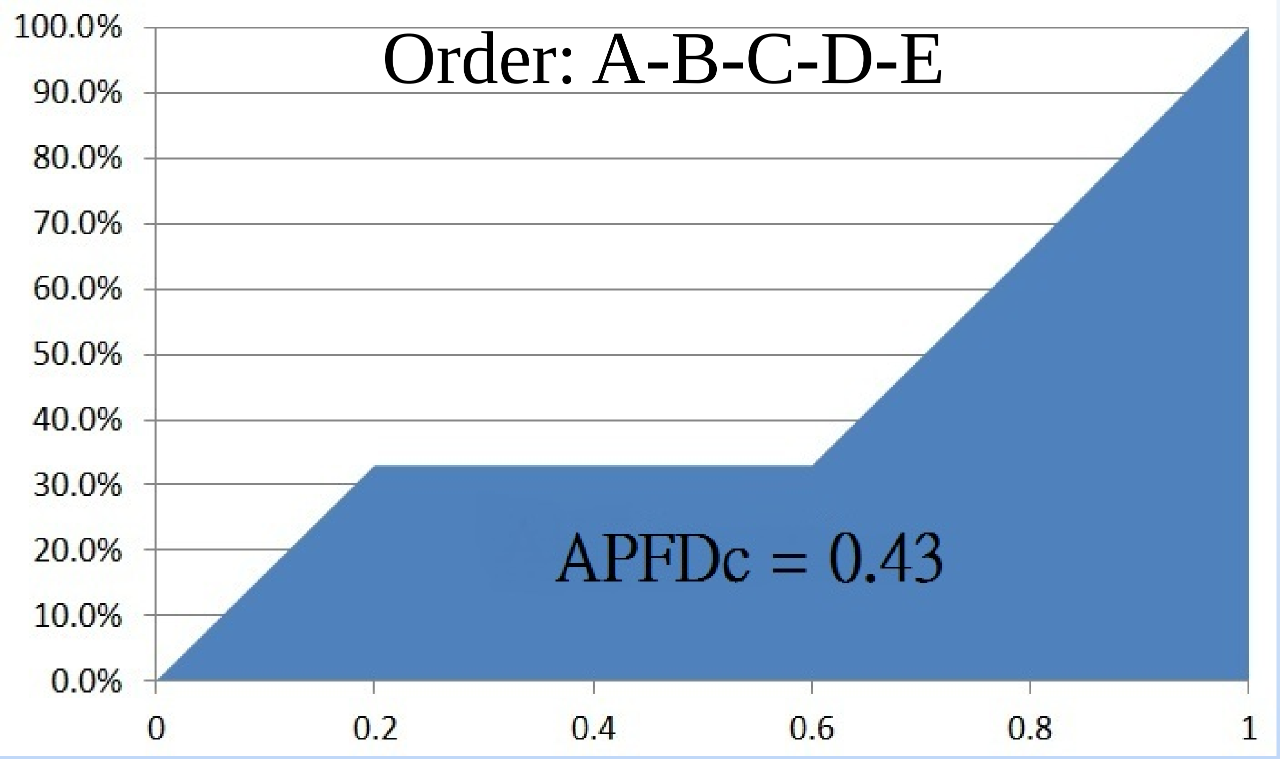
- Average Percentage of Fault Detected per Cost (APFDc)

$$APFDc = \frac{\sum_{i=1}^{m} f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i}$$

  - f_i: fault severity of fault i
  - t_j: execution cost of test case j
  - n: the number of test cases in the test suite
  - m: the number of faults that are revealed by the test suite
  - TF_i: the first test case in an ordering test suite that reveals fault i

Criterion used to evaluate prioritization

Detected fault(%)

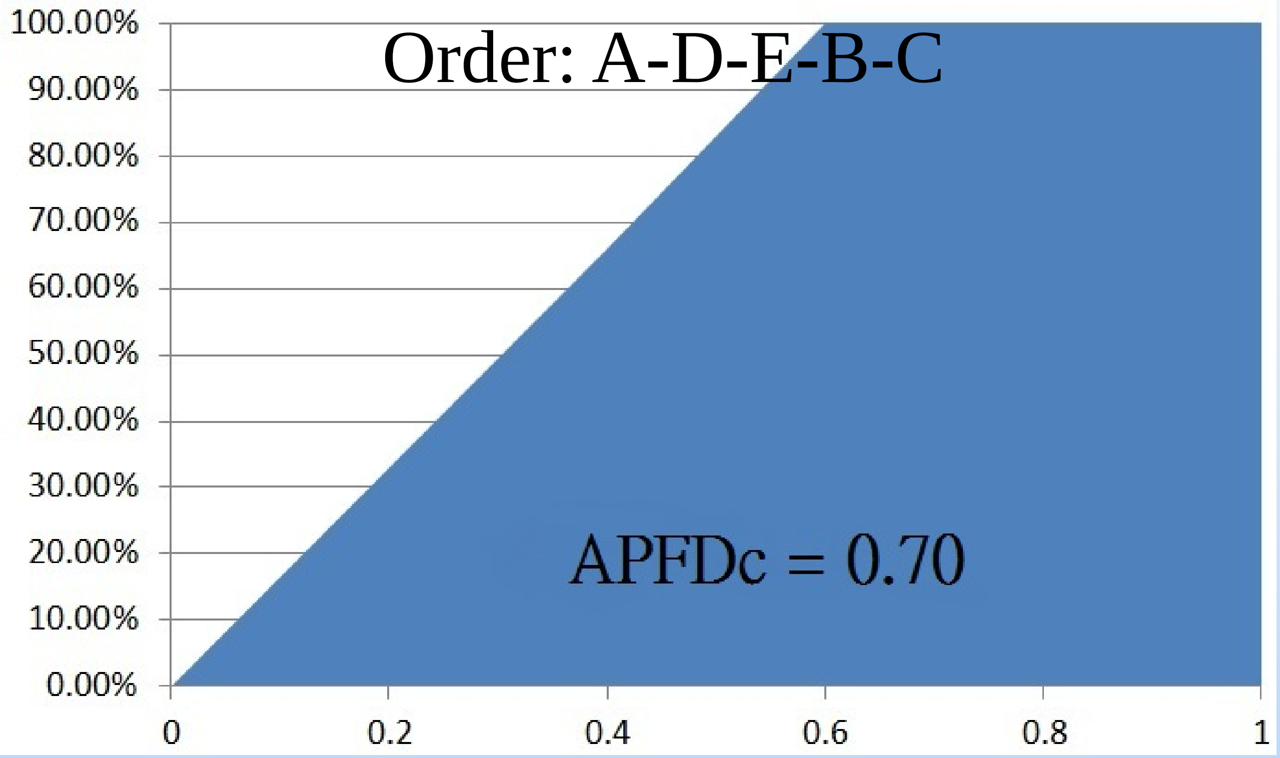


Test suite fraction

Test case	A	B	C	D	E
Detecting faults or not	✓			✓	✓

Criterion used to evaluate prioritization

Detected fault(%)



Test suite fraction

Test case	A	B	C	D	E
Detecting faults or not	✓			✓	✓

Historical information

- Software developer benefits from the historical data.
  - Historical fault data: fault detections of a specific test case in the previous versions

Test suite	Version 00 (Original)	Version 01	Version 02	Version 03
A	✓	✓		
B		✓	✓	✓
C			✓	✓
D	✓	✓		
E	✓		✓	

History-based test case prioritization

- Previous test results can provide useful information to make future testing more efficient.
- Kim and Porter proposed a history-based test case prioritization.
  - They prioritize test cases using historical test execution data.
- Liu et al. prioritize test cases based on information concerning historical faults and the source code.

Motivation

- The previous approaches assumed that the immediately preceding test result provides the same reference value for prioritizing the test cases of the successive software version.
- Open research question: is the reference value of the test result of the immediately preceding version of the software version-aware for the successive test case prioritization?
  - This research presents a test case prioritization approach based on our observations.

Subject programs

- Siemens programs
  - From Software-artifact Infrastructure Repository (SIR)
  - Benchmarks that are frequently used to compare different test case prioritization methods

Programs	Test pool size	# of branches	# of versions
printtokens	4,130	140	7
printtokens2	4,115	138	10
replace	5,542	126	32
schedule	2,650	46	9
schedule2	2,710	72	10
tcas	1,608	16	41
totinfo	1,052	44	23

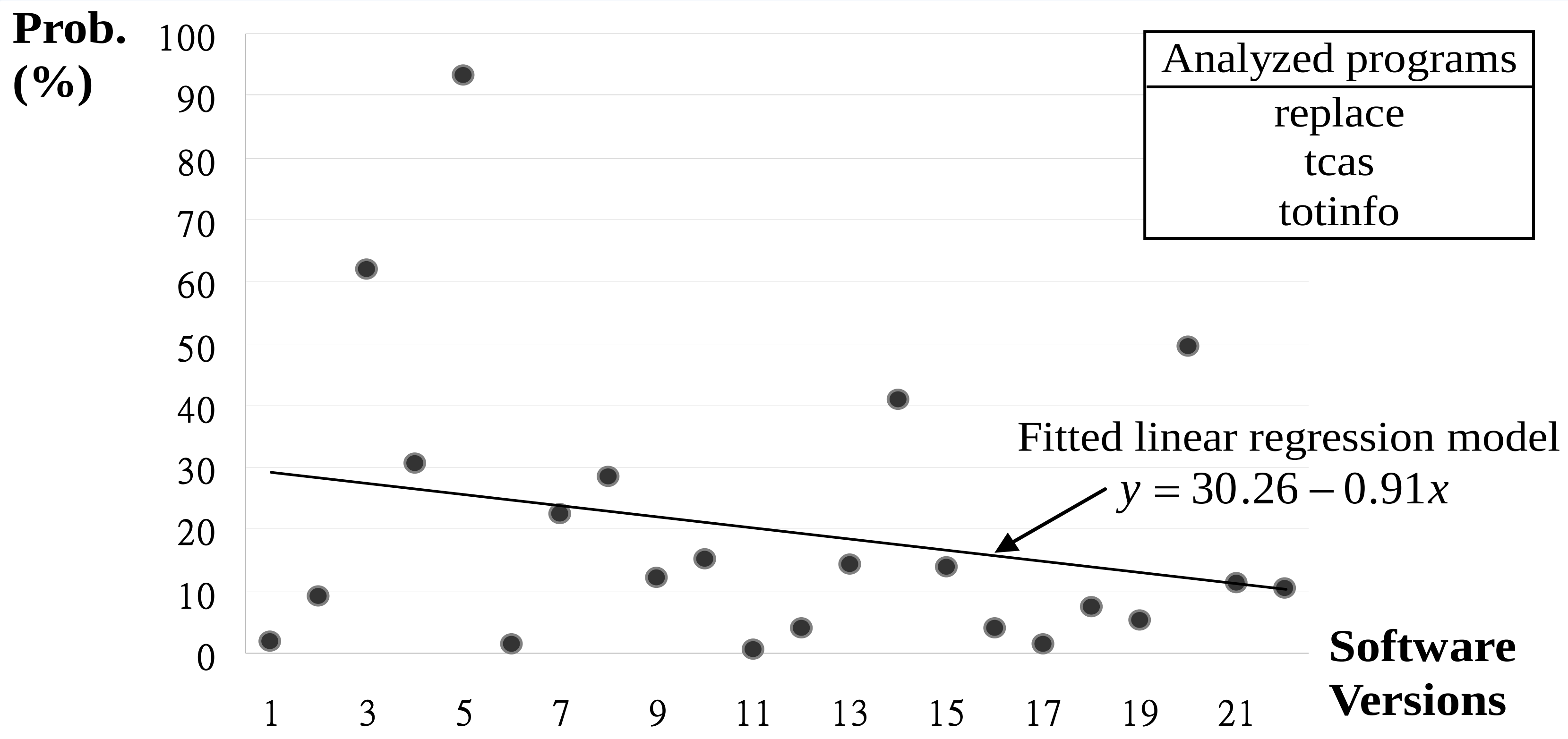
Analysis 1: Fault-prone test cases

- We found that, for the test cases detecting faults in a specific version, there is a higher probability that they will detect faults again in the successive version.

Analysis 1- Fault-prone test cases (Cont.)

Subject Programs	If a test case failed in a specific version	If a test case passed in a specific version
	Prob. that it fails in the next version	
printtokens	6.78%	2.05%
printtokens2	22.25%	3.95%
replace	7.39%	1.78%
schedule	3.79%	1.68%
schedule2	7.55%	0.81%
tcas	5.61%	2.78%
totinfo	21.30%	5.96%

Analysis 2: Repeated fault detection



- Prob. that a test case detects faults in two successive software versions as the programs evolve.

Analysis 2: Repeated fault detection (Cont.)

- The linear regression plot indicates that the probability tends to decrease as the programs evolve.
  - A test case detects faults in two successive versions may get less and less significant.

Assumptions of presented method

1. Both historical fault data and source code information are valuable for prioritizing test cases in the later software versions;
2. The priorities of the test cases that detected faults in the immediately preceding version should be increased;
3. The increment described in Assumption 2 is software-version-aware and will linearly decrease as the programs evolve.

Presented method

$$P_k = \begin{cases} C_{num}, \text{if } k = 0, \\ P_{k-1} + h_k \times C_{num} \times \boxed{[(Vers - k)/Vers]}, \text{if } k > 0, \end{cases}$$

- P_k: the priority of the test case in the *k*-th version
- h_k: the historical information that indicates whether the test case detected a fault in the (*k*-1)-th version
- C_{num}: the number of branches covered by the test case
- *Vers*: the number of versions of the subject program

Methods compared in the empirical study

- Kim and Porter’s history-based test case prioritization [Kim and Porter, ICSE 2002]
- Liu et al.’s history-based test case prioritization [Liu et al., Internetware 2011]
- Random prioritization
- Presented method

Preliminary experimental analyses

Programs	Kim & Porter's	Liu et al.'s	Random	Presented
printtokens	54.86%	70.12%	49.52%	70.11%
printtokens2	79.25%	72.65%	50.68%	81.95%
replace	72.62%	68.18%	49.42%	76.33%
schedule	67.41%	56.13%	49.94%	63.27%
schedule2	58.25%	51.05%	48.70%	60.27%
tcas	66.52%	60.31%	50.23%	74.13%
totinfo	69.83%	72.32%	48.96%	74.46%
Average	66.96%	64.39%	49.64%	71.50%

- The presented approach normally provides the best fault detection rates.

Conclusion and future work

- This paper presented a software-version-aware approach that considers both source code information and historical fault data.
- The presented approach provides better fault detection rates than the established methods.
- We intend to
  - use a full-featured model to adjust the software-version-aware test case priority more accurately.
  - conduct more experiments with case study applications that have more source code and tests.