2016 IEEE International Conference on Software Quality, Reliability and Security

DFL: Dual-service Fault Localization

C. M. Tang, Jacky Keung, Y. T. Yu, W. K. Chan
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Kowloon,
Hong Kong
c.m.tang@my.cityu.edu.hk, {Jacky.Keung, csytyu, wkchan}@cityu.edu.hk

***Abstract*—In engineering a service, software developers often construct and deploy a newer (forthcoming) version of the service to replace the current version. A forthcoming version is often placed online for users to consume and report feedback. In the case of observed failures, the forthcoming version should be debugged and further evolved. In this paper, we propose the model of *dual-service fault localization* (DFL) to aid this evolution process. Many prior research studies on spectrum-based fault localization (SBFL) consider each version separately. The DFL model correlates the dynamic execution spectra of the current and the forthcoming versions of the same service placed for live test of the forthcoming version, and dynamically generates an adaptive fault localization formula to estimate the code regions in the forthcoming service responsible for the observed failures. We report an experiment in which we initialized the DFL model into six instances, each using an ensemble technique dynamically composed from 11 existing SBFL formulas, and applied the model to four benchmarks. The results show that DFL is feasible and multiple instances are statistically more effective than, if not as effective as, the best of these individual SBFL formulas on each benchmark.**

Keywords—debugging; spectrum-based fault localization; ensemble techniques; dual-service fault localization

I. INTRODUCTION

Over the last decades, the advent of the Internet motivates companies to turn their legacy offline programs into online services. It not only enables the execution of their programs to be accessible to users outside the companies, but also supports software deployment and upgrades centralized at the backend side of these services. The time-to-market pressure may further motivate companies to place some beta version of their services (that is, trial services) online for collecting feedback from users who trial use (or test) the beta service. In many cases, these trial services are less qualified than the deployed service for production. Nonetheless, many developers use the execution statistics of these trial services to help improve the quality of the production version of the same service before it is finally released online.

Debugging is an opportunity available in such a scenario. Similar to a conventional software debugging process, when a failure is seen, the typical first step in debugging a trial service is *fault localization*, that is, to identify the faults in code that lead to failure.

Fault localization is one of the major bottlenecks in software development [2][33]. The state-of-the-art fault localization techniques include *slicing* [26], *delta debugging* [32] and *spectrum-based fault localization* (hereafter abbreviated as SBFL) [10]. SBFL recommends a priority to each program code region of the service under debug. It utilizes a *program spectrum* (that is, an execution profile) that indicates the parts of the program that are active for a given program execution trace. A typical SBFL technique, such as Naish2 (called Optimal (O^P) in the original paper) [18], uses a SBFL formula to assign a value known as *suspiciousness score* [10] to each program entity based on the program spectra obtained by executing the program over a test suite. Then a list of all program entities, each annotated with a suspiciousness score, is ordered by ranks, for subsequent code examination. The program entity could be a statement [2], branch [34], path [3] or object code instruction [24].

We put forward a question: Is there any new fault localization scenarios applicable to trial services rather than merely following the conventional paths suggested in the SBFL literature?

In this paper, we propose the model of ***Dual-service Fault Localization*** (DFL). DFL monitors a pair of services on serving a stream of inputs. The two services in the pair are: (1) the trial service under debug, and (2) a shadow copy of the current service to be replaced by the production version of the trial service. Whenever a user consumes (invokes) the trial service via a request, DFL also diverts a copy of the request to the shadow copy of the current service for execution. The execution profiles of the two services are separately collected and aggregated to the program spectra of these two services. Once a failure on the trial service is detected, if the shadow copy of the current version had once experienced a failure, our DFL fault localization procedure is invoked, which *adaptively* computes the suspiciousness scores of the program entities of the trial service for SBFL-based debugging purpose.

In services computing, a service provider offers a tailored service experience to each user. The DFL procedure firstly synthesizes a customized **ensemble** SBFL formula based on a suite of underlying individual SBFL formulas (hereafter called **solo formulas**) utilizing the program spectra of both the forthcoming service and a shadow copy of the current service. The key elements of the DFL model are based on two prior results as follows.

978-1-5090-4127-5/16 $31.00 © 2016 IEEE
DOI 10.1109/QRS.2016.53

First, the notion of ensemble technique is commonly used in item recommendation and recently in software effort estimation to overcome the limitation of using a single (solo) technique, the latter being found to be often unstable across different datasets [11]. The use of some moderated results of multiple solo techniques can often achieve relatively good results (on average) on a larger amount of datasets than the use of the results of each solo technique [11]. Second, theoretical analysis results on a set of solo formulas in SBFL research has identified that some of them can be superior to some others, while there are still many other formulas whose (relative) effectiveness is still unknown from a theoretical perspective [30]. DFL adopts the *multi-methods* [11] approach to synthesize its *adaptive fitness fault localization formula* (denoted as F_{AFFL}). After an instance of F_{AFFL}, DFL runs the formula instance on tailored program spectra of the trial service under test to make recommendation on potential faulty program code regions.

We conducted an experiment to test the concepts of the DFL model by initializing it into two instances of *multi-methods* and two instances of the construction of the tailored program spectra mentioned above. We compared our DFL instances with 11 solo formulas. We used four UNIX subjects as the benchmarks. The empirical results show that two instances of DFL are always among the most effective formulas, that is, they are consistently either more effective than or as effective as each formula on all benchmarks.

The main contribution of this paper is threefold. First, it proposes the notion of dual service fault localization. Second, it is the *first* work to propose a fault localization approach that dynamically compose a SBFL formula on the fly for given program spectra using both the notions of ensemble methods and spectra differences. Third, it reports an empirical study that reveals effective and concrete instances of the composed SBFL formula for use in our DFL model.

The rest of this paper is organized as follows. Section II introduces the background and preliminaries. Section III overviews our DFL model and its key notions. Section IV presents the research question, the different instances of DFL under study in the experiment, and the experimental setup, and then discuss the results and threats to validity. Section V reviews the related work. Section VI concludes the paper.

II. PRELIMINARIES

A SBFL formula F_x utilizes the following four variables (called **coverage variables** and denoted as ***cov***) to compute the fault ***suspiciousness score*** (denoted as $\boldsymbol{susp_x}$) of each program entity **s**.

- a_{ef} = number of failed test cases in which s is executed
- a_{nf} = number of failed test cases in which s is skipped
- a_{ep} = number of passed test cases in which s is executed
- a_{np} = number of passed test cases in which s is skipped

For instance, the SBFL formula Naish2 [18] is defined as follows.

$$a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$$

```
s1  if ( variable_trailing_context_rules )
s2          reject = true;
s3  if ( fulltbl || (fullspd && reject) ) // faulty
    {
s4          if ( real_reject )
s5                  flexerror( "..." );
            else
s6                  flexerror( "..." );
    }
```

Figure 1. A code excerpt of the *flex* program version 1 obtained from *SIR* [5]. It consists of 6 executable statements corresponding to the source code lines 847–861 of the fault-seeded version.

TABLE I. COVERAGE VARIABLES, SUSPICIOUSNESS SCORES AND RANKS OF THE STATEMENTS IN THE SAMPLE CODE EXCERPT IN FIGURE 1

Stmt	t_1	t_2	t_3	t_4	t_5	t_6	a_{ef}	a_{nf}	a_{ep}	a_{np}	$susp_{\text{Naish2}}$	Rank
s_1	1	1	1	1	1	1	2	0	4	0	1.2	2
s_2	1	0	0	1	0	0	0	2	2	2	–0.4	6
s_3	1	1	1	0	1	1	2	0	3	1	1.4	1
s_4	1	0	1	0	0	1	1	1	2	2	0.6	4
s_5	1	0	1	0	0	0	0	2	2	2	–0.4	6
s_6	0	0	0	0	0	1	1	1	0	4	1.0	3
	p	p	p	p	f	f						

A suspiciousness score computed by using the formula Naish2 is denoted as $susp_{\text{Naish2}}$.

The suspiciousness scores of all the program entities in the faulty program are ordered into *ranks* by using a certain *ranking scheme* [17]. A ranking scheme assigns a rank value to each item (suspiciousness score in this paper's context) in an ordered list. By convention, the higher the suspiciousness score of an entity, the lower is its rank value. Ranking schemes differ by how they assign ranks to tie cases.

Figure 1 shows a code excerpt of the *flex* program version 1 downloaded from the well-known *Software-artifact Infrastructure Repository* (*SIR*) [5]. The code excerpt consists of 6 executable statements (s_1–s_6) in the `void readin()` function, in which s_3 is the faulty statement. The correct implementation for s_3 is as follows.

```
if ( (fulltbl || fullspd) && reject )
```

Table I illustrates how the Naish2 formula identifies the potential fault locations for the code excerpt in Figure 1. The column **S** refers to the statements s_1–s_6. The next six columns show the coverage statistics (computed from executing the six test cases t_1–t_6) for each executable statement. Each cell in these columns indicates that the corresponding statement is *covered* (if the cell value is 1) or *not covered* (if the cell value is 0) when the program is executed by the particular test case. The bottom row shows the test verdict, *pass* (p) or *failure* (f), of executing each of the six test cases (t_1–t_6).

For example, consider the statement s_3, which is covered by 2 failed (t_5 and t_6) and 3 passed (t_1, t_2 and t_3) test cases. One other passed (t_4) test case does not cover it, and there is no other failed test case. Thus the values of a_{ef}, a_{nf}, a_{ep} and a_{np} for s_3 are 2, 0, 3 and 1, respectively. The SBFL formula, Naish2, thus computes the suspiciousness score of s_3 to be 1.4. The suspiciousness scores of the other five statements are computed in the same way. Statements s_2 and s_5 received the

same (tied) suspiciousness score of –0.4.

The last column of Table I shows the assignment of ranks to the program entities (statements in this example) by using the *modified competition ranking scheme* (also called the "1-3-3-4" scheme) [17], which has been used to represent the ***worst case*** scenario [29] that the SBFL formula recommends the faulty program entity only after all other entities of the same rank have been exhausted.

In Table I, the true faulty statement s_3 is assigned the rank value 1, which indicates that the SBFL formula Naish2 recommends s_3 as the 1st priority suspected faulty statement.

III. Dual-service Fault Localization

A. Overview

Extending from typical SBFL techniques, the proposed DFL model locates faults of the forthcoming (newer) version of a service by utilizing the coverage profiles and bug reports of the current and forthcoming versions of the same service. Figure 2 shows the architecture of the proposed DFL. $\mathsf{Service}_1$ is the released service (current version) in production, which serves normal user requests. $\mathsf{Service}_2$ is the prospective service upgrade (forthcoming version) that had been published for beta testing for the user to experience. Whenever $\mathsf{Service}_2$ responds to a user's request, the request is also redirected to a *shadow* copy of $\mathsf{Service}_1$. From the perspective of debugging, each service request can be viewed as a test input, which is collected together with the execution profiles for the subsequent SBFL process.

One possible scenario is as follows. $\mathsf{Service}_2$ correctly computes the results for the service requests so that the test verdicts are passes unless a bug report (feedback by a service user) is received. In receiving a bug report, developers verify whether it really indicates the existence of a bug and, if so, whether the bug affects only one version or both versions. The test verdicts for the two versions are marked and associated with the test input. The test suites and the test verdicts are then passed to DFL for fault localization.

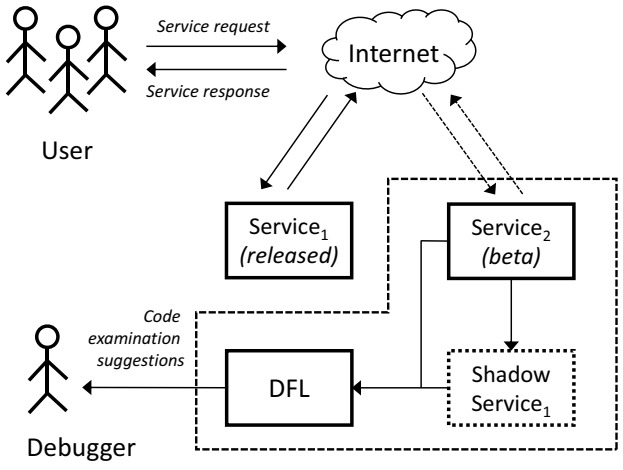


Figure 2. The architecture of the DFL model.

Most of the SBFL techniques require at least one failing test and one passing test to work effectively. As $\mathsf{Service}_2$ is supposed to have passed developers' tests prior to being published as a beta release, it should result in many passing test cases with only occasional failing test cases. So DFL can be triggered only when a failing test case is encountered, which often comes with a large amount of passing test cases to aid the fault localization process. Similarly, both services may directly compute results to service consumers, in which cases, many passed test cases can be collected.

DFL is able to utilize the program spectra of both $\mathsf{Service}_1$ and $\mathsf{Service}_2$ to compute the fault suspiciousness scores of program entities for $\mathsf{Service}_2$. The entities are then ranked according to the list of suspiciousness scores as in a typical SBFL process.

B. Adaptive Fitness Fault Localization Formula

Inspired by the multi-methods approach proposed by Kocaguneli et al. [11] that constructs an ensemble of multiple methods for the purpose of software effort estimation, we propose to firstly construct an ensemble SBFL formula, F_{ensemble}. For each program entity s, its values of the coverage variables are passed to each solo SBFL formula F_x for computing the suspiciousness score $susp_x(\text{s})$. As the suspiciousness scores $susp_x(\text{s})$ computed from different solo SBFL formulas F_x may range widely, each score is normalized to a maximum value of 1 by the factor $\max\_susp_x$, which is the maximum possible suspiciousness score computed by F_x. The ***normalized suspiciousness score*** corresponding to $susp_x(\text{s})$ is denoted by $nsusp_x(\text{s})$ and defined as:

$$nsusp_x(\text{s}) = \frac{susp_x(\text{s})}{\max\_susp_x}$$

The value of $\max\_susp_x$ is calculated by applying F_x to the following values of the coverage variables: $a_{ep} = 0$, $a_{ef} = N_{\text{f}}$, $a_{np} = N_{\text{p}}$ and $a_{nf} = 0$, where N_{f} and N_{p} are the total number of failing and passing test cases, respectively.

F_{ensemble} is then defined as the formula which computes the ***ensemble suspiciousness score*** $susp_{\text{ensemble}}(\text{s})$ of a program entity s as the weighted sum of its normalized suspiciousness scores computed from n solo SBFL formulas F_x ($x = 1 \ldots n$):

$$susp_{\text{ensemble}}(\text{s}) = \sum_{x=1}^{n} w_x \times nsusp_x(\text{s}) = \sum_{x=1}^{n} \frac{w_x \times susp_x(\text{s})}{\max\_susp_x}$$

where w_x is the *weight* assigned to the solo SBFL formula F_x.

DFL may further dynamically adjust the ensemble suspiciousness score $susp_{\text{ensemble}}(\text{s})$ computed from an ensemble formula F_{ensemble} to produce an ***adaptive fitness fault localization score***, $susp_{\text{AFFL}}$, which defines the formula F_{AFFL} in DFL. Specifically, for each program entity (such as a statement) s common to both the current and forthcoming versions of a service, the ***delta spectrum*** $\Delta(\text{s})$ refers to the list of differences between the normalized suspiciousness scores, $nsusp_{x,\text{forthcoming}}(\text{s})$ and $nsusp_{x,\text{current}}(\text{s})$, respectively, of the forthcoming and current versions computed from a SBFL formula, F_x, and aggregated over all x ($x = 1 \ldots n$), as follows:

TABLE II. BENCHMARK PROGRAMS

Subject Program	Lines of Code (LOC)	No. of Versions Available	Total No. of Seeded Faults	No. of Test Cases
flex	12423–14244	5	81	567
grep	12653–13372	5	57	809
gzip	6576–7996	5	59	214
sed	6671–11990	7	32	360–370

$$\Delta(\mathrm{s}) = \sum_{x=1}^{n} (nsusp_{x,\mathrm{forthcoming}}(\mathrm{s}) - nsusp_{x,\mathrm{current}}(\mathrm{s}))$$

Our study experiments with two ***weighting schemes*** and three ***delta spectrum schemes***, as to be detailed in Section IV.

IV. EMPIRICAL STUDY

In this section, we present the research question and procedures of the exploratory experiment in this research.

A. Research Question

The DFL model has two key elements in relation to SBFL: (1) composing an ensemble formula from a set of solo SBFL formulas using a *weighting scheme*, and (2) dynamically adjusting the ensemble formula on the fly by using a *delta spectrum scheme* that utilizes the program spectra of both $\mathsf{Service}_1$ (current version) and $\mathsf{Service}_2$ (forthcoming version) of a service.

Research Question (RQ): To what extent can the two key elements of the DFL model help improve the effectiveness of fault localization of $\mathsf{Service}_2$?

Answering this question is important in this research to establish a supporting basis of the DFL model and understand the potential effect of the model's two key elements.

B. Benchmark Suites

To mimic the service upgrade scenario, we chose four UNIX utility programs as benchmarks, namely, ***flex***, ***grep***, ***gzip*** and ***sed***, which have been widely used in SBFL and related studies such as regression testing. They were retrieved from SIR [5], together with their accompanied test pools. A summary of the descriptive statistics of the four benchmark programs is shown in Table II. Each of them consists of five to seven release versions and each version was activated with one seeded fault constituting a set of seeded fault versions.

We executed each of these versions with the accompanied test cases. Our study included those seeded fault versions that could be reached by at least one passing test and one failing test. The seeded fault versions that were adopted in the empirical study are listed in Table III. In this study, the source code statements are taken as the program entities.

C. SBFL Formulas and Weighting Schemes

Naish et al. [18] evaluated a set of 33 SBFL formulas that commonly appeared in the literature, together with six new formulas. Xie et al. [30] theoretically analyzed the relationships of effectiveness of 30 such formulas and proved that two of the formulas are *maximal*, that is, they are never outperformed by other formulas under study. However, the

TABLE III. SEEDED FAULT VERSIONS ADOPTED IN THE EMPIRICAL STUDY

Subject Program	Release Version	Seeded Fault Version Adopted in the Study
flex	1	1, 3, 4, 5, 6, 7, 9, 10, 11, 14, 15, 17, 18, 19
	2	2, 3, 5, 6, 7, 9, 11, 12, 14, 15, 16, 18, 19
	3	6, 9, 10, 11, 14, 15, 17
	4	5, 6, 7, 8, 12, 13, 14, 15, 16
	5	6, 9
grep	1	3, 7, 8, 11, 14
	2	1, 2, 6, 7
	3	2, 3, 8, 12, 13, 16, 18
	4	2, 10, 12
	5	*nil*
gzip	1	2, 4, 5, 14, 15, 16
	2	1, 3, 6
	3	*nil*
	4	1, 6, 10
	5	7, 9
sed	1	*nil*
	2	1, 2, 3, 4, 5
	3	1, 2, 3, 4, 5, 6
	4	*nil*
	5	1, 2, 3, 4
	6	1, 3, 4, 5, 6
	7	1, 2, 3, 4

ID	Name	Group	Formula	Ref.
F_1	Naish2	ER1	$a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$	[18]
F_2	Russell & Rao	ER5	$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$	[22]
F_3	Ample		$\left\lvert \frac{a_{ef}}{a_{ef} + a_{nf}} - \frac{a_{ep}}{a_{ep} + a_{np}} \right\rvert$	[4]
F_4	Geometric mean		$\frac{a_{ef}a_{np} - a_{nf}a_{ep}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$	[16]
F_5	Harmonic Mean		$\frac{(a_{ef}a_{np} - a_{nf}a_{ep})\left((a_{ef} + a_{ep})(a_{np} + a_{nf}) + (a_{ef} + a_{nf})(a_{ep} + a_{np})\right)}{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}$	[21]
F_6	Kulczynski1		$\frac{a_{ef}}{a_{nf} + a_{ep}}$	[13]
F_7	M1		$\frac{a_{ef} + a_{np}}{a_{nf} + a_{ep}}$	[6]
F_8	Ochiai2		$\frac{a_{ef}a_{np}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$	[19]
F_9	Overlap		$\frac{a_{ef}}{\min(a_{ef}, a_{nf}, a_{ep})}$	[7]
F_{10}	Rogot2		$\frac{1}{4}\left(\frac{a_{ef}}{a_{ef} + a_{ep}} + \frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{np}}{a_{np} + a_{ep}} + \frac{a_{np}}{a_{np} + a_{nf}}\right)$	[21]
F_{11}	Zoltar		$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + \frac{10000a_{nf}a_{ep}}{a_{ef}}}$	[1]

Figure 3. Solo SBFL formulas included in this study.

effectiveness relationships among many of the remaining formulas are still unknown from a theoretically perspective.

Our study included the two maximal formulas identified in [30] and nine other SBFL formulas, as listed in Figure 3. These 11 formulas were used to construct the ensemble formula F_{ensemble} by using two different weighting schemes, W0 and W1. The first scheme, W0, assigns a uniform value (e.g., 1) as the weight of each solo formula. The other scheme, W1, firstly computes the normalized suspiciousness scores of each program entity according to all the solo formulas, then ranks the solo formulas by these scores, and finally assigns the rank value of each solo formula as its weight in F_{ensemble}. Details of the weights adopted in the weighting scheme W1 are described in an example in Section IV.*E* and illustrated in Figure 4 below.

D. Applying Delta Spectra in DFL

Let Δ denote the delta spectrum of the current version (Service_1) and forthcoming version (Service_2) of a service. Denote by average(Δ) the values of Δ(s) averaged over all statements s common to both versions. We adopted two schemes D1 and D2 of applying delta spectra, as follows.

As the implementations of two versions of the service differ, their lists of program entities are also non-identical. Since the objective is to identify faults in the forthcoming version (Service_2), DFL only computes the suspiciousness scores $susp_{\text{AFFL}}(s)$ of program entities s in the forthcoming version Service_2. There are two cases.

Case 1. For an entity s which appears in both current and forthcoming versions, $susp_{\text{AFFL}}(s)$ is computed by adjusting $susp_{\text{ensemble}}(s)$ with Δ(s) as follows.

$$susp_{\text{AFFL}}(s) = susp_{\text{ensemble}}(s) + \Delta(s) \quad \text{(Scheme D1)}$$

$$susp_{\text{AFFL}}(s) = \Delta(s) \quad \text{(Scheme D2)}$$

Case 2. For an entity s which appears in the forthcoming version only, both schemes D1 and D2 compute the value of $susp_{\text{AFFL}}(s)$ by adjusting $susp_{\text{ensemble}}(s)$ with the average value of Δ(s), as follows.

$$susp_{\text{AFFL}}(s) = susp_{\text{ensemble}}(s) + \text{average}(\Delta)$$

We also experimented with the scheme D0 of *not* applying delta spectra for adjusting suspiciousness scores in order to contrast its effect with that of *applying* schemes D1 and D2. Table IV summarizes the six combinations of weighting and delta spectrum schemes applied in our study.

E. Illustrative Example

Figure 4 illustrates a simplified example of how the weighting scheme W1 and the delta spectra schemes synthesize the program spectra in the dual-service scenario. The figure consists of 6 subfigures (a)–(f) showing each step of the computation that leads to the adaptive fitness fault localization scores and their ranks.

In Figure 4(a), the table shows the normalized suspiciousness scores, $nsusp_{x,\text{current}}$ (x = 1, 2, 3), computed by the corresponding solo SBFL formulas F_x for each of the 5 statements s_a–s_e in a segment of code (represented by the letters A–E, respectively) of the current version.

TABLE IV. SCHEMES ADOPTED IN THE IMPLEMENTATION OF F_{AFFL}

Formula	Weighting Scheme	Delta Spectrum Scheme	
F_{W0D0}	Weighted uniformly by a constant (e.g., 1)	Do not apply delta spectra.	
F_{W0D1}		Add the delta spectrum to the suspiciousness scores of the common code.	Add the average delta value to the suspiciousness scores of codes that appear in the forthcoming version only.
F_{W0D2}		Use the delta spectrum as the suspiciousness scores of the common code.	
F_{W1D0}	Weighted by ranks of suspiciousness scores	Do not apply delta spectra.	
F_{W1D1}		Add the delta spectrum to the suspiciousness scores of the common code.	Add the average delta value to the suspiciousness scores of codes that appear in the forthcoming version only.
F_{W1D2}		Use the delta spectrum as the suspiciousness scores of the common code.	

Consider the forthcoming version of the same service shown in Figure 4(b). This new version evolves from the current version with the statements s_a (code A) and s_d (code D) deleted and the new statements s_3 (code F), s_4 (code G) and s_6 (code H) added. For ease of reference, the statements in the new version are relabelled as s_1–s_6 such that s_1 and s_b are the same statements (with the unchanged code B) and so are s_2 and s_c (unchanged code C) as well as s_5 and s_e (unchanged code E). The rows of the statements common to both versions are connected by dotted arrows between Figures 4(a) and 4(b), and are shaded in yellow in Figures 4(a), 4(b) and 4(e) for ease of tracing and identification.

Similar to Figure 4(a), the table in Figure 4(b) shows the normalized suspiciousness scores, $nsusp_{x,\text{forthcoming}}$ (x = 1, 2, 3), computed by the corresponding solo SBFL formulas F_x for each of the 6 statements s_1–s_6 of the forthcoming version.

For each statement s_i in the forthcoming version, its normalized suspiciousness scores computed by using the solo formulas F_1–F_3 are compared and ranked, with the lowest score ranked 1, the second lowest score ranked 2 and the highest score ranked 3 when the three scores are all different. For instance, the statement s_1 receives the scores 0.5, 0.6 and 0.7 from the solo formulas F_1, F_2 and F_3, respectively. Since $0.5 < 0.6 < 0.7$, the formulas F_1, F_2 and F_3 are assigned the rank values 1, 2 and 3, respectively.

In case of equal scores, the corresponding solo formulas are assigned the same fractional rank values by the fractional ranking scheme (or called "1-2.5-2.5-4" scheme) [17]. For instance, the statement s_4 receives the scores of 0.3, 0.2 and 0.2 from F_1, F_2 and F_3, respectively. Since the score of F_1 is larger than the equal scores of F_2 and F_3, the first formula F_1 is ranked 3 while the latter two are assigned the same rank value of $(1 + 2) \div 2 = 1.5$.

The rank values thus assigned to the solo formulas are

used as their weights in our DFL model with weighting scheme W1 for computing the ensemble suspiciousness score $susp_{\text{ensemble}}$. Figure 4(c) tabularizes all the weights w_x thus assigned to F_x (x = 1, 2, 3) for each statement in this example.

Once the weights are fixed, the ensemble suspiciousness score $susp_{\text{ensemble}}(s_i)$ for each statement s_i can be calculated as a weighted sum of its individual suspiciousness scores received from the solo formulas. In Figure 4(d), the ensemble suspiciousness scores $susp_{\text{ensemble}}(s_i)$ of the statements s_i (i = 1, 2, …, 6) are shown in the rightmost column, which are respectively the sum of the weighted normalized suspiciousness scores in the three preceding columns.

Recall in Section III.*B* that for each statement s_i (i = 1, 2, 5) common to the current and forthcoming versions of the same service, the delta spectrum $\Delta(s_i)$ is defined as the sum of all differences between the normalized suspiciousness scores of the two versions across all the solo formula F_x (x = 1, 2, 3).

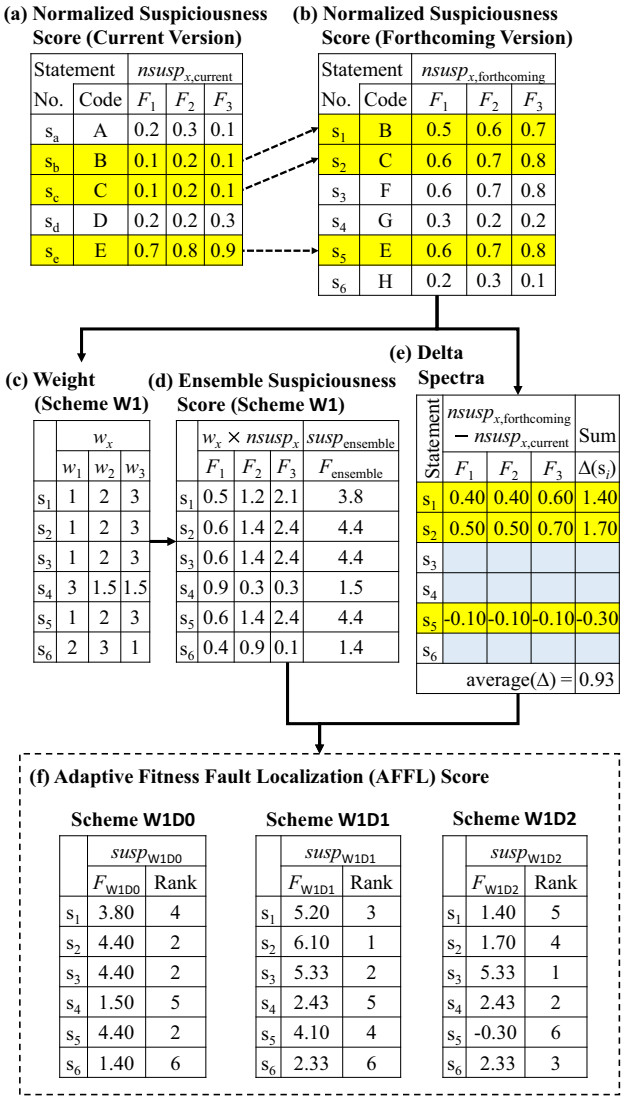


Figure 4. An illustrative example for F_{AFFL}.

Figure 4(e) shows the delta spectra for this example in the rightmost column. Note that for statements in the forthcoming version but not the current version, the delta spectrum is undefined. The average of all delta spectra values of common statements, denoted by average(Δ), is equal to (1.40 + 1.70 + –0.30) ÷ 3 = 0.93 in this example.

If the delta spectra are not applied for adjustment (which corresponds to our "null" delta spectrum scheme D0), the ensemble suspiciousness scores $susp_{\text{ensemble}}$ will be directly used in our DFL model as the final adaptive fitness fault localization scores $susp_{\text{AFFL}}$ for further ranking and the rest of the SBFL process. In Figure 4(f), the table on the left shows that the suspiciousness scores $susp_{\text{W1D0}}$ are simply the same as the scores $susp_{\text{ensemble}}$ in Figure 4(d).

In our experiment, we consider the two delta spectra schemes D1 and D2 defined in Section IV.*D* above and summarized in Table IV. Basically, the schemes adjust the ensemble suspiciousness score of each statement by either adding or replacing it with the respective delta spectrum value or adding the average delta value. The final calculated adaptive fitness fault localization scores $susp_{\text{AFFL}}$ corresponding to the schemes W1D1 and W1D2 for this example are shown in Figure 4(f).

Our experiment also studies the effectiveness of our DFL model with the use of the uniform weighting scheme W0. We do not show the computations of $susp_{\text{AFFL}}$ for the schemes W0D0, W0D1 and W0D2, as the methods of computation are the same as those of W1D0, W1D1 and W1D2 except that the weights in Figure 4(c) are all replaced by 1 or any other uniform value.

F. Effectiveness Metric

We used *expense* (also called *code examination effort* or *EXAM score*) to measure the degree of precision of a SBFL formula in recommending a true positive (faulty statement).

$$expense = \frac{\text{Smallest rank of the faulty statements}}{\text{Total number of statements}}$$

A smaller *expense* value indicates higher effectiveness.

G. Experimental Setup

1) Environment and Tools

The services were deployed in a virtual machine configured with Intel Xeon CPU X5560 @ 2.8GHz×4, 16GB of physical memory and 1TB disk storage. Ubuntu 12.04 LTS (64-bit) was used as a platform for compiling and instrumenting the subjects. We compiled the benchmark programs using the default (no optimization) option of GCC version 4.6.3 with argument "`-gdwarf-2`" for injecting debugging information in *Debugging With Attributed Record Formats* (DWARF) version 2. *Pin* [15] version 4.13 was used to extract program structural information, code coverage and debugging information such as line numbers. Data analysis was conducted in a virtual machine with the same hardware as above. Microsoft Server 2012 was used as the operating system platform. The programs for compiling empirical data

TABLE V. RELATIVE EFFECTIVENESS OF 11 SOLO FORMULAS AND 6 IMPLEMENTATIONS OF F_{AFFL} FORMULAS UNDER STUDY

flex			*grep*			*grip*			*sed*		
L	**Formula**	**Est. Mean**	**L**	**Formula**	**Est. Mean**	**L**	**Formula**	**Est. Mean**	**L**	**Formula**	**Est. Mean**
A	F_{W1D2}	0.0177	A	F_{W1D2}	0.0125	A	F_{W1D2}	0.0119	A	F_1	0.0121
B	F_{W0D2}	0.0214	A	F_{W0D2}	0.0133	A	F_{W0D2}	0.0122	A	F_6	0.0121
C	F_{W0D1}	0.0348	B	F_1	0.0174	B	F_{W1D0}	0.0152	A	F_{11}	0.0121
C	F_1	0.0350	B	F_6	0.0174	B	F_6	0.0154	A	F_{W1D0}	0.0121
C	F_6	0.0350	B	F_{11}	0.0174	B	F_{11}	0.0154	A	F_{W0D0}	0.0121
C	F_{11}	0.0350	B	F_{W1D0}	0.0174	B	F_8	0.0154	AB	F_{W1D2}	0.0127
C	F_{W1D0}	0.0350	B	F_{W0D0}	0.0174	B	F_1	0.0155	ABC	F_4	0.0130
C	F_{W0D0}	0.0350	B	F_4	0.0176	B	F_{W0D0}	0.0155	ABC	F_5	0.0130
D	F_4	0.0383	B	F_5	0.0176	B	F_4	0.0155	BC	F_{W0D2}	0.0141
D	F_5	0.0383	B	F_8	0.0178	B	F_5	0.0155	BCD	F_8	0.0144
E	F_{W1D1}	0.0431	B	F_{10}	0.0180	B	F_{10}	0.0156	CD	F_{W1D1}	0.0148
F	F_8	0.0469	C	F_{W0D1}	0.0219	B	F_3	0.0156	D	F_{10}	0.0162
G	F_{10}	0.0579	D	F_{W1D1}	0.0237	C	F_{W0D1}	0.0192	E	F_{W0D1}	0.0187
H	F_3	0.0701	E	F_2	0.0362	C	F_{W1D1}	0.0203	F	F_2	0.0479
H	F_2	0.0713	E	F_9	0.0362	D	F_2	0.0317	F	F_9	0.0479
H	F_9	0.0713	F	F_3	0.0375	D	F_9	0.0317	G	F_3	0.0525
I	F_7	0.0829	G	F_7	0.0747	E	F_7	0.0368	H	F_7	0.0908
Std. Errors		0.0004	**Std. Errors**		0.0002	**Std. Errors**		0.0003	**Std. Errors**		0.0004
df		380528	**df**		178041	**df**		39270	**df**		195687
s		0.0531	**s**		0.0255	**s**		0.0161	**s**		0.0414

and computing the SBFL formulas were coded in C# using Microsoft Visual Studio 2012. MATLAB R2104b (8.4.0.150421) was used for analyzing data.

2) Procedures

We adopted the fault seeded versions of each release version of the four benchmark programs in which their faulty *statements* can be reached by at least one failing test case. Such faulty versions are known as *fault revealing versions*, which are shown in Table III. We paired up the fault revealing versions of two consecutive released versions as the current and forthcoming versions of a service. For instance, released version 1 of the *flex* program has 14 fault revealing versions while version 2 has 13 fault revealing versions, yielding 14 × 13 = 182 pairs of current and forthcoming versions of services for the empirical study.

Each pair of current and forthcoming fault revealing versions for each benchmark program was instantiated as two concurrent services deployed in our sandbox environment. We iteratively executed a random test case to test both the current and forthcoming versions of the service and then recorded their coverage profiles and test verdicts. A test suite was formed from the accumulated random test cases. The values of the coverage variables kept evolving upon executing more test cases. As required by SBFL, the test suite should have at least one failing and one passing test cases. As soon as this requirement was satisfied, the SBFL process began by computing the weights and normalized suspiciousness scores of the solo formulas. When the current version also received at least one failing and at least one passing test cases, delta spectra were also computed where applicable as described in Section IV.*D*. to produce the (adjusted) ensemble suspiciousness scores (the F_{AFFL} values). Finally, the expense values of all the 11 solo formulas and the six implementations of the formula F_{AFFL} shown in Table IV were computed using the "1-3-3-4" ranking scheme as introduced in Section II. To reduce the effect of randomness, we repeated the experiment 10 times.

3) Statistical Tests Used in Data Analysis

We applied the *one-way analysis of variance* (ANOVA1) test with Bonferroni correction at the 0.05 significance level to assess whether the formulas under study had a common estimated mean at the required significance level. If the difference was significant, we further performed the *multiple comparison test* [9] to compare the mean and the corresponding mean intervals of such formulas.

By using the above tests, MATLAB automatically labelled each group of (solo or ensemble) SBFL formulas with a letter or more. Formulas sharing at least one letter in their labels indicate that their means are not statistically different (at the 0.05 significance level). MATLAB assigns the letter labels in alphabetical order, where 'A' refers to the group of the most effective formulas, 'B' refers to the next most effective group, and so on. A formula whose mean interval overlaps with those of two other formulas or more is labelled with multiple letters, such as 'AB' for formulas that overlap with group A and group B formulas.

H. Results

Table V summarizes the effectiveness of applying the 6 F_{AFFL} implementations and 11 solo SBFL formulas to the four

benchmark programs *flex*, *grep*, *gzip* and *sed*. The table consists of 3 columns for each benchmark program, showing the letter group labels (**L**) of each formula (**Formula**) and its estimated mean expense value (**Est. Mean**). The last 3 rows of the table show the standard errors (**Std. Errors**) of the estimated values of mean expense of the multiple comparison tests, error (within-groups) degrees of freedom (**df**) and square root (**s**) of the mean squared error of the ANOVA1 test.

In the table, formulas with non-overlapping letter group labels are statistically different in effectiveness (and separated by solid horizontal lines) while formulas within groups that share the same letter do not have significant difference in effectiveness (and separated by dashed horizontal lines).

Along the *ensemble formula construction* dimension, formulas F_{W1D2} and F_{W1D0} are consistently either more or equally effective compared to each solo formula (F_1 to F_{11}) as indicated by the letter groups across all benchmarks. On the other hand, F_{W1D1} is inferior to some solo formulas. The results show that using dynamic ensembles as indicated by F_{AFFL} alone is insufficient to ensure high effectiveness.

Moreover, from the table, it seems that we have the order of W1 ≥ W0 in terms of letter group ranking, where we use the symbol "≥" to denote the relationship that the former is more effective than or at least as effective as the latter.

Along the *delta spectrum* dimension, F_{W0D0} and F_{W1D0} are as effective as the solo formulas achieving the highest effectiveness on each benchmark. It is however sometimes less effective than F_{W1D2} on three benchmarks (*flex, gzip,* and *grep*). The result shows that the delta spectrum scheme D0 can be inferior to D2. Sometimes, the delta spectrum scheme D1 is less effective than D0.

Formulas F_{W0D1} and F_{W1D1} are not highly effective across all four benchmarks. Thus, it seems that we have the non-increasing order of effectiveness relationship D2 ≥ D0 ≥ D1 in terms of letter group ranking.

Interestingly, as indicated in the table, the three formulas F_{W0D0}, F_{W1D0} and F_{W1D2} are at least as effective as all studied solo formulas on all the studied benchmarks. It shows that 50% of the DFL model instances are effective in this experiment. Moreover, F_{W1D2} is the most effective formula among all studied formulas. Note that F_{W1D2} is ranked in the most effective group (labelled 'A') for the three benchmarks (*flex, gzip,* and *grep*), while for the fourth benchmark, it belongs to the group labelled 'AB', which means that statistically, there is no significant difference between F_{W1D2} and the formulas in the most effective group (labelled 'A').

The results in the experiment show that using ensemble formulas and applying a specific delta spectrum scheme (D2) can be a promising choice for instantiating DFL.

Next, we use *violin plots* [8] to show the spread and probability density of the expense values of each of the 6 F_{AFFL} implementations on each of the benchmark programs in Figure 5. Table VI shows the quartile values, interquartile ranges and a legend for interpreting the violin plots.

A violin plot can be read as a combination of the histogram and the boxplot where along its length the bar's height shows the spread and skewness of the data distribution, while along its width the bar's thickness shows the probability density (relative likelihood) of the data at different values.

From Figure 5 and Table VI, we see that F_{W1D2} has the least overall spread among the benchmark programs in terms of the interquartile range, while F_{W0D2} is the next (except for the benchmark program *sed*). On the other hand, F_{W0D0} and F_{W1D0} result in moderate spreads, while F_{W0D1} and F_{W1D1} are relatively less predictable. The results show that implementations that adopt different weighting schemes do not vary much in terms of spread.

On the other hand, comparing implementations that adopt different delta spectrum schemes shows that D2 generally results in smaller spreads than those of D0, while D1 results in wider spreads than those of both D2 and D0.

To answer the research question posed in Section IV.*A*, the empirical results reveal that applying delta spectra scheme D2 in conjunction with the ensemble implementation of F_{AFFL} improves the fault localization effectiveness of the forthcoming version of the service in a statistically significant manner.

I. Threats to Validity

Our empirical study models after a scenario of debugging over a service upgrade using four medium-scale benchmark programs taken from *SIR*. Studies of subjects of larger scale, diverse types of applications, different fault samples (such as real faults versus seeded faults), and various kinds of test suites can strengthen the generalizability of the result. Our ensemble formula included 11 solo SBFL formulas only. Inclusion of different sets of solo SBFL formulas can further reduce the external threats to validity. Similarly, other schemes of weighting and applying the delta spectra may be explored for higher generality of results. We randomly sampled test cases from the test pool to simulate the arbitrary service requests of users. The actual service request pattern in practice may not be purely random and may yield different results. We have mainly evaluated the effectiveness of DFL in the single-fault scenario. In practice, the quantity of faults is never known in advance. On the other hand, empirical studies [2][10][27] have provided evidence that the SBFL technique remains effective in locating a fault in multiple-fault programs. Whether our DFL model can achieve higher fault localization effectiveness in multiple-fault programs than traditional SBFL techniques remains to be investigated.

V. RELATED WORK

In this section, we review previous work that is related to our DFL model. Generally, SBFL utilizes program spectra for distinguishing faulty program entities. Santelices et al. [23] investigated three types of program entities in SBFL, namely, statements, branches and data dependencies. They concluded that the effectiveness of using different entities is different when localizing different kinds of faults. Tang et al. [24] empirically studied the performance of SBFL at object code instruction level and found that it could be higher than the performance at the source statement level.

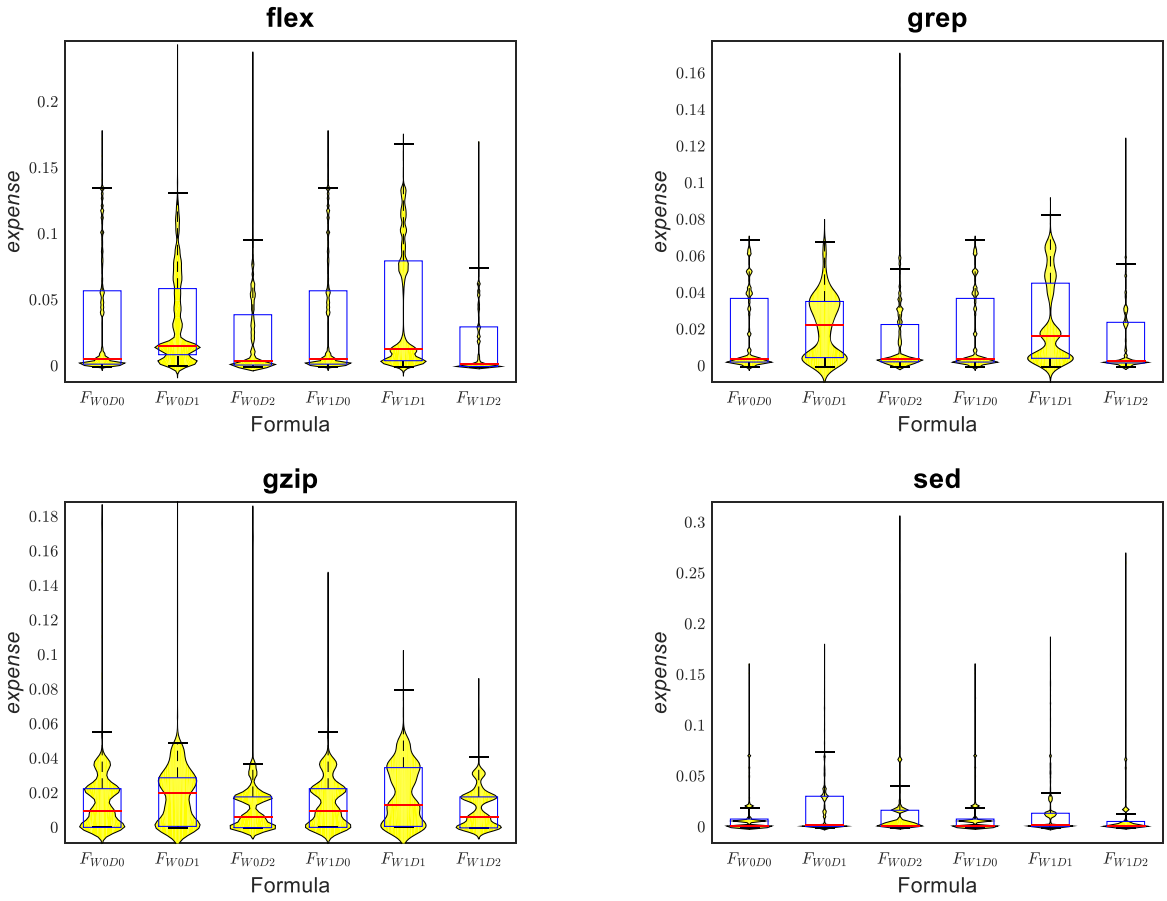
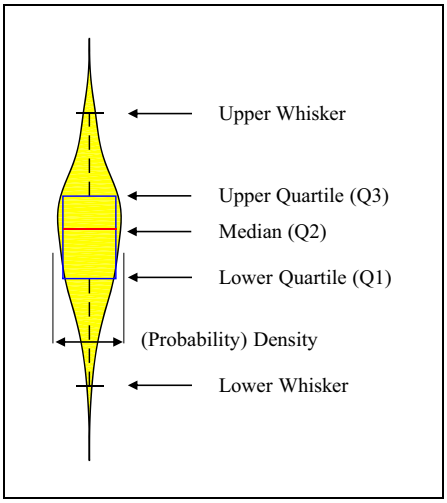


Figure 5. Violin plots showing the effectiveness spreads of different implementations of F_{AFFL}.

TABLE VI. STATISTICS OF THE VIOLIN PLOTS SHOWN IN FIGURE 5

		F_{W0D0}	F_{W0D1}	F_{W0D2}	F_{W1D0}	F_{W1D1}	F_{W1D2}
flex	Q3	0.0577	0.0594	0.0396	0.0577	0.0804	0.0304
	Q2	0.0064	0.0162	0.0049	0.0064	0.0139	0.0024
	Q1	0.0022	0.0093	0.0015	0.0022	0.0048	0.0004
	IQR	**0.0555**	**0.0501**	**0.0381**	**0.0555**	**0.0756**	**0.0300**
grep	Q3	0.0375	0.0358	0.0232	0.0375	0.0458	0.0244
	Q2	0.0046	0.0227	0.0043	0.0046	0.0170	0.0031
	Q1	0.0027	0.0051	0.0028	0.0027	0.0048	0.0022
	IQR	**0.0348**	**0.0307**	**0.0204**	**0.0348**	**0.0410**	**0.0222**
gzip	Q3	0.0231	0.0295	0.0184	0.0231	0.0353	0.0184
	Q2	0.0103	0.0204	0.0068	0.0103	0.0139	0.0068
	Q1	0.0008	0.0013	0.0006	0.0008	0.0013	0.0004
	IQR	**0.0223**	**0.0282**	**0.0178**	**0.0223**	**0.0340**	**0.0180**
sed	Q3	0.0089	0.0313	0.0175	0.0089	0.0146	0.0065
	Q2	0.0021	0.0029	0.0018	0.0021	0.0029	0.0018
	Q1	0.0014	0.0016	0.0014	0.0014	0.0014	0.0014
	IQR	**0.0075**	**0.0297**	**0.0161**	**0.0075**	**0.0132**	**0.0051**

Legend for Figure 5



Note: A violin plot is similar to a box plot in that along its length it shows the median (Q2), lower and upper quartiles (Q1 and Q3), lower and upper whiskers (extreme values) of a data distribution, except that along its width the violin plot also shows the probability density (relative likelihood) of the data at different values as its width. In this study, the lower and upper whiskers are calculated by the commonly used values of (Q1 − 1.5 × IQR) and (Q3 + 1.5 × IQR), respectively, where IQR (= Q3 − Q1) is the interquartile range.

Various approaches have been proposed to employ multiple methods for enhancing the capability over individual methods. Many researchers adopted learning-based techniques that utilize historical fault information for model training and then use the trained models for fault localization. For instance, Xuan and Monperrus [31] proposed a learning-based framework called MULTRIC that combined 25 state-of-the-art SBFL formulas with weights for fault localization. The weight of each formula was learnt from the spectra of the faulty and non-faulty program entities with a given faulty program, its test cases and test verdicts.

Le et al. [12] proposed the multi-modal technique that utilized debugging information from different sources. In particular, they adopted information retrieval (IR)-based fault localization techniques that analyzed the possible fault localizations from bug reports together with SBFL formulas that utilized program spectra to improve the fault localization effectiveness over the techniques that used a single source of information. These techniques required apriori model training with historical training data. Similarly, Wang et al. [25] presented a search-based approach. Their approach applied weights to the SBFL formulas generated with *genetic algorithm* (GA) and *simulated annealing* (SA) that modelled fault localization as optimization problems. Nevertheless, training data is often unavailable and the faults in the training data may not be representative for debugging the program on hand.

Some other researchers employed adaptive algorithms for locating faults in online faulty programs without requiring model training. Our DFL model was inspired by the work of Kocaguneli et al. [11] who applied multi-methods to construct ensemble effort estimation models. They concluded that some of the multi-methods significantly outperformed all the solo methods studied, with smaller error rates and higher stability in ranks. Lucia et al. [14] proposed the *data fusion methods* for constructing *fusion localizers* that fuse the adaptively selected SBFL formulas together for locating faults. Empirical results showed that certain variants of fusion localizers were more effective than individual formulas.

Qi et al. [20] presented a debugging approach with a tool called DARWIN that located faults for evolving programs, that is, a stable old version together with a faulty new version of the same program. They used symbolic execution to automatically synthesize the test inputs of two versions and identify the faulty code by analyzing the points where control flows diverged over the passing and failing inputs. A recent comprehensive survey [28] has provided an informed and annotated summary of key techniques and issues on the research on software fault localization in general, in addition to SBFL techniques.

VI. Conclusion

In this paper, we have proposed a novel model of dual-service fault localization (DFL). We have proposed a way to dynamically customize an ensemble formula specific to a given program spectrum, and a way to construct delta spectra for adjusting the suspiciousness scores. In the reported empirical study, some DSL instances have demonstrated their potentials to produce improvements in effectiveness through suitable application of delta spectra schemes.

Acknowledgment

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project numbers 111313, 11201114, 11200015, 123512, and 125113), and the research funds of City University of Hong Kong (project numbers 7200354, 7004222 and 7004474).

References

[1] R. Abreu, *Spectrum-based Fault Localization in Embedded Software*. PhD thesis, Delft University of Technology, Delft, Holland, 2009.

[2] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[3] T.M. Chilimbi, B. Liblit, K. Mehra, A.V. Nori, and K. Vaswani, "HOLMES: Effective statistical debugging via efficient path profiling," *Proceedings of the International Conference on Software Engineering* (*ICSE*), 2009, pp. 34–44.

[4] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight defect localization for Java," *Proceedings of the European Conference on Object-Oriented Programming* (*ECOOP*), 2005, pp. 528–550.

[5] H. Do, S.G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[6] B. Everitt, *Graphical Techniques for Multivariate Data*. North-Holland. 1978.

[7] M. Dunham, *Data Mining: Introductory and Advanced Topics*. Prentice-Hall, 2002.

[8] J.L. Hintze and R.D. Nelson, "Violin plots: A box plot-density trace synergism," *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998.

[9] Y. Hochberg and A.C. Tamhane, *Multiple Comparison Procedures*. Hoboken, NJ: John Wiley & Sons, 1987.

[10] J.A. Jones and M.J. Harrold, "Empirical evaluation of the Tarantula automatic fault-localization technique," *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), 2005, pp. 273–282.

[11] E. Kocaguneli, T. Menzies, and J.W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012.

[12] T.-D.B. Le, R.J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: Better together," *Proceedings of the Joint Meeting on Foundations of Software Engineering* (*ESEC/FSE*), 2015, pp. 579–590.

[13] F. Lourenco, V. Lobo, and F. Bação, "Binary-based similarity measures for categorical data and their application in Self-Organizing Maps," *Proceedings of the Conference on Classification and Analysis of Data* (*JOCLAD 2004*), 2004, pp. 121–138.

[14] Lucia, D. Lo, and X. Xia, "Fusion fault localizers," *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering* (*ASE*), 2014, pp. 127–138.

[15] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI*), 2005, pp. 190–200.

[16] A. Maxwell and A. Pilliner, "Deriving coefficients of reliability and agreement for ratings," *British Journal of Mathematical and Statistical Psychology*, vol. 21, no. 1, pp. 105–116, 1968.

[17] S.K. Mishra, "The most representative composite rank ordering of multi-attribute objects by the particle swarm optimization," *Journal of Quantitative Economics*, vol. 8, no. 2, pp. 165–200, 2010.

[18] L. Naish, H.J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, article no. 11, 2011.

[19] A. Ochiai, "Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions," *Bulletin of the Japanese Society for the Science of Fish*, vol. 22, pp. 526–530, 1975.

[20] D. Qi, A. Roychoudhury, Z. Liang, and K. Vaswani, "DARWIN: An approach to debugging evolving programs," *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 3, article no. 19, 2012.

[21] E. Rogot and I.D. Goldberg, "A proposed index for measuring agreement in test-retest studies," *Journal of Chronic Disease*, vol. 19, pp. 991–1006, 1966.

[22] P. Russel and T. Rao, "On habitat and association of species of anopheline larvae in South-Eastern Madras," *Journal of the Malaria Institute of India*, vol. 3, no. 1, pp.153–178, 1940.

[23] R. Santelices, J.A. Jones, Y. Yu, and M.J. Harrold, "Lightweight fault-localization using multiple coverage types," *Proceedings of the International Conference on Software Engineering* (*ICSE*), 2009, pp. 56–66.

[24] C.M. Tang, W.K. Chan, and Y.T. Yu, "Extending the theoretical fault localization effectiveness hierarchy with empirical results at different code abstraction levels," In *Proceedings of the Annual International Computers, Software and Applications Conference* (*COMPSAC*), 2014, pp. 161–170.

[25] S. Wang, D. Lo, L. Jiang, Lucia, and H.C. Lau, "Search-based fault localization," *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, (*ASE*), 2011, pp. 556–559.

[26] M. Weiser, "Program slicing," *Proceedings of the International Conference on Software Engineering* (*ICSE*), 1981, pp. 439–449.

[27] W.E. Wong, V. Debroy, Y. Li, and R. Gao, "Software fault localization using DStar (D*)," *Proceedings of the IEEE International Conference on Software Security and Reliability* (*SERE*), 2012, pp. 21–30.

[28] W.E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey of software fault localization," *IEEE Transactions on Software Engineering*, to appear, 2016, DOI:10.1109/TSE.2016.2521368

[29] W.E. Wong, Y. Qi, L. Zhao, and K. Cai, "Effective fault localization using code coverage," *Proceedings of the Annual International Computer Software and Applications Conference* (*COMPSAC*), 2007, pp. 449–456.

[30] X. Xie, T.Y. Chen, F.C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 4, article no. 31, 2013.

[31] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," *Proceedings of the IEEE International Conference on Software Maintenance and Evolution* (*ICSME*), 2014, pp. 191–200.

[32] A. Zeller, "Yesterday, my program worked. Today, it does not. Why?" *Proceedings of Eurpoean Software Engineering Conference / ACM SIGSOFT Symposium on the Foundations of Sonftware Engineering* (*ESEC*/*FSE*), 1999, LNCS, vol. 1687, pp. 253–267.

[33] A. Zeller, "Isolating cause-effect chains from computer programs," In *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering* (*FSE*), 2002, pp. 1–10.

[34] Z. Zhang, B. Jiang, W.K. Chan, T.H. Tse, and X. Wang, "Fault localization through evaluation sequences," *Journal of Systems and Software*, vol. 83, no. 2, pp. 174–187, 2010.