8-31-2005

Profiling Deployed Software: Strategic Probe Placement

Madeline Diep
University of Nebraska - Lincoln

Sebastian Elbaum
University of Nebraska - Lincoln, elbaum@cse.unl.edu

Myra B. Cohen
University of Nebraska - Lincoln, mcohen2@unl.edu

University of Nebraska–Lincoln, Computer Science and Engineering
Technical Report TR-UNL-CSE-2005-0016
Issued August 31, 2005

Profiling Deployed Software: Strategic Probe Placement

Madeline Diep, Sebastian Elbaum, Myra Cohen
Department of Computer Science and Engineering
University of Nebraska
Lincoln, Nebraska
{mhardojo, elbaum, myra}@cse.unl.edu

Abstract

Profiling deployed software provides valuable insights for quality improvement activities. The probes required for profiling, however, can cause an unacceptable performance overhead for users. In previous work we have shown that significant overhead reduction can be achieved, with limited information loss, through the distribution of probes across deployed instances. However, existing strategies for probe distribution do not account for several relevant factors: acceptable overheads may vary, the distributions to be deployed may be limited, profiled events may have different likelihoods, and the user pool composition may be unknown. This paper evaluates strategies for probe distribution while manipulating these factors through an empirical study. Our findings indicate that for tight overhead bounds: 1) deploying numerous versions with complementary probe distributions can compensate for the reduction of probes per deployed instance, and 2) techniques that balance probe allocation, consider groupings and prior-allocations, can generate distributions that retain significantly more field information.

1 Introduction

Software profiling consists of observing, gathering, and analyzing data to characterize a program's behavior. Profiling deployed software is valuable because it can provide insights into how the software is actually utilized [8, 14], which configurations are being employed [13], what development assumptions may not hold [4], where validation activities are lacking [5], or which scenarios are most likely to lead to a failure [11].

Profiling usually requires instrumentation of the program, that is, the addition of probes to enable the examination of the program's run-time behavior. As such, the act of profiling penalizes the targeted software with execution overhead. The magnitude of the overhead depends, at least to some extent, on the number, the location, and the type of inserted profiling probes.

To reduce profiling overhead, instrumentation techniques may choose to insert just a subset of the probes, but do so strategically to reduce data loss (e.g, avoid inserting probes that capture redundant or inferable data). When instrumenting a program to be profiled in the field, techniques may also reduce the number of probes by distributing them across a pool of deployment sites. Such an approach consists of generating multiple program variants to be deployed across sites, where each variant contains a subset of probes, and where the subset size can be bounded to meet the overhead requirements.

Clearly, overhead reduction through probe distribution across variants implies that less data is collected from each deployed site. Still, given enough sites and an appropriate distribution of probes across variants, the aggregation of the data collected from all sites can generate an accurate representation of how the software is exercised in the field. In practice, however, generating a distribution that is likely to capture an accurate representation from the field is challenging due to, at least, the following factors:

- Acceptable overhead levels of deployed instances, which bounds the number of probes that can be allocated to each variant.
- Deployment costs, which may limit the number of variants that can be deployed and maintained in the field.
- Targeted behavior, which defines probe allocation process parameters such as event likelihood or event groupings.
- Varying attributes of the potential deployed sites (e.g., size, types, change rate, response rate), which continually redefines how the probes should be allocated across subsequent releases.

This paper investigates how to effectively perform probe distribution across variants within the constraints imposed by these four factors so that the likelihood of collecting valuable field information is increased.

Our contribution is two-fold. First, we define the probe allocation problem across variants while taking into consideration the factors just listed, and we present a meta-algorithm that can be instantiated in multiple ways to accommodate various probe distribution strategies that attempt to account for these factors. Second, we empirically evaluate various probe distribution strategies applied to a deployed system while manipulating the four factors in order to assess their impact on the effectiveness of the strategies.

2 Background

Previous studies have investigated ways to enhance the efficiency of profiling techniques by: (1) performing up-front analysis to optimize the amount of instrumentation required [2, 19], (2) sacrificing accuracy by monitoring entities of higher granularity or by sampling program behavior [6, 7], (3) encoding the information to minimize memory and storage requirements [17], and (4) repeatedly targeting the entities that need to be profiled [1, 3]. These mechanisms reduce the amount of instrumentation inserted or executed in a program through the analysis of the properties the program exhibits in a controlled in-house environment.

When profiling deployed software, we can also leverage the opportunities introduced by a potentially large pool of users. The following research efforts, for example, focus in that direction: 1) Perpetual Testing, which uses the residual testing technique to reduce instrumentation based on previous coverage results [16, 18], 2) EDEM, which provides a semi-automated way to collect user-interface feedback from remote sites when it does not meet an expected criterion [9], 3) Skoll, which presents an architecture and a set of tools to distribute different job configurations across users [13], 4) Gamma, which introduces an architecture to distribute and manipulate instrumentation across deployed instances [8, 15], and 5) Bug Isolation, which provides an infrastructure to efficiently collect field data and identify likely failure conditions [11, 12]. Our work is closely related to the last two projects.

It is similar to Gamma in that the overall mechanism to reduce overhead utilizes probe distribution across deployed instances. However, our focus has been on developing techniques to achieve distributions that increase the likelihood of retaining field data, and their empirical validation [5]. The work in this paper builds on our previous studies by considering instrumentation constraints, event groupings, an unknown number of deployed instances, and deployment costs that limit the number of variants, to guide the probe distribution process. Furthermore, continuing with the empirical focus, we have conducted a study on a truly deployed non-trivial system to address our research questions (Section 4).

It is related to the bernoulli-sampling used by the Bug-Isolation project since both attempt to reduce overhead through some form of sampling. However, while the bernoulli-sampling approach includes additional instrumentation to dynamically determine *when* to sample, our objective is to determine before deployment *what* to sample in each variant. Our approach aims to produce a fair sample across variants, without incorporating the additional instrumentation required to adjust the sampling at run-time. In Section 6 we discuss how these approaches can be complementary.

3 Probe Distribution on Variants

This section characterizes the problem of probe distribution across variants to control overhead while maximizing the likelihood of capturing field information. It illustrates different distribution approaches, states the research questions, and provides a meta-algorithm that can be instantiated to generate distributions with different properties.

3.1 Problem Characterization

A software engineer is interested in profiling program P after deployment to learn about certain aspects of the program behavior that manifests in the field (e.g., coverage, hot-spots, usage patterns). To capture such behavior, the engineer can generate an instrumented variant of P called v_0 with a complete set of probes H, one for each instrumentable unit u in P. This approach, however, may cause an unacceptable performance overhead. As previously mentioned, program analysis techniques can be used to reduce the number of units worth profiling, resulting in a decreased H. The overhead reduction achieved through such analysis, however, may not be sufficient to support the intended granularity and type of profiling without disturbing the user. To address this problem, the engineer can specify an acceptable maximum number of probes h_{bound} that can be included in variant v_0 to enable the profiling activities. h_{bound} can be computed such that, for example, profiling overhead is hidden in the program's operating noise (performance variation observed during various program executions).

Since generally $h_{bound} << |H|$, data collected from all the sites when *one* variant with h_{bound} probes is deployed will reflect only a part of the program behavior exercised in the field. Having multiple deployed variants of the program,

where each variant has a somewhat complementary subset of assigned probes, can offer a more representative reflection of the program behavior [4]. This approach will result in n variants v_0, v_1, ..., and v_{n-1}, where each variant contains a subset of probes Hv_0, Hv_1, ..., and Hv_{n-1}, such that the size of each subset is less or equal to h_{bound}. We simplify the discussion by assuming that $|Hv_0| = |Hv_1| = ... = |Hv_{n-1}| = h_{bound}$. In this scenario a minimum of $\lceil \frac{|H|}{h_{bound}} \rceil$ variants are needed to cover the entire probe space in H.

When determining the number of variants (n) that should be generated, additional issues to take into consideration include the potential value of the collected data, the cost of variant generation and deployment, and the predicted size and variation of the user pool. Note that the size of the user population is a practical limitation for the number of variants worth generating, and equates to each user having a unique variant. More commonly, however, the number of users is much greater than the number of variants, so each variant is likely to be utilized by multiple users. It is then up to the engineer to determine the right proportion of users per variant based on the assessed cost-benefits.

Once n and h_{bound} are defined, the challenge is to find a distribution of probes across variants that maximizes the likelihood of capturing a representative part of the program behavior exercised in the field. More formally, we define the probe distribution across variants problem as follows:

Given:

U, a set of units in P, $u \in U$, and where u identifies a potential location for probe h;

PD, the set of potential probe distributions across U on n variants such that $\forall$ variants : $|Hv_0|$=$|Hv_1|$=...=$|Hv_{n-1}| = h_{bound}$;

f, a function from PD to the real numbers that, when applied to any such distributions, yields an *award value*.

Problem: Find a distribution $D \in PD$ such that $\forall D' \in PD$, $D \neq D'$, $[f(D) \geq f(D')]$.

The definition of f will depend on the information that is being targeted. For example, f could be the number of blocks covered, the potential invariants violated, or the hot-spots identified. Depending upon the choice of f, the problem of finding the best distribution may be intractable. Thus, in practice, we resort to heuristics to approximate D.

3.2 Potential Distributions Strategies

Figure 1 illustrates various probe distribution strategies across variants for a program with: $|U| = 9$, $h_{bound} = 3$, and $n = 3$. The first alternative, *Pattern*, inserts probes in three neighboring units in each variant, taking into consideration the probes inserted in previous variants to minimize overlapping of probes per unit across variants. This alternative is simple, but it may be subject to bias due to the concentration of probes in certain sections of a program variant (e.g., variants with probes in rarely executed units will provide no information from the field, variants with probes in well executed areas will bring repeated data with limited value added).

The second alternative, *Random*, is likely to avoid the previous type of biases. However, a random distribution can still lead to an uneven distribution of probes. Some units may be instrumented in many variants while others may be instrumented in none. In Figure 1 under *Random*, no probes are allocated to capture information relative to units u3, u7 or u9, but all variants allocate one probe to profile the activity of u2.

Strategies may benefit when considering two additional attributes: *unit-balancing* and *unit-grouping*. Unit-balancing attempts to balance the probes assigned to units across variants. *Random Balanced* in Figure 1, solves the inequities of *Random* by keeping track of the number of probes previously allocated for each unit, and performing random allocation considering only the units that have fewer probes. Keeping a balanced distribution may be more challenging in the presence of changing profiling requirements. As profiling requirements evolve (e.g., specific program sections may require more data to explain certain behavior, or expectations for what constitutes acceptable overhead may vary), variants fitting different policies may have been deployed, which makes balancing infeasible. Still, we conjecture that strategies that consider previous allocations, even when performed under different profiling requirements, are likely to be beneficial.

Unit-grouping consists of clustering units (e.g., per scope, per frequency, per usage) into q groups, where each group j may be assigned a different number of probes ($g_j h_{bound}$), to better match the data collection interests while keeping the same h_{bound} per variant. For example, an engineer may assign a higher bound to a group of untested components in order to learn more about their usage context [16]. Figure 1 provides an example of grouping called *Grouped Random Balanced*, where there are two groups, one with $g_1 h_{bound} = 1$ and one with $g_2 h_{bound} = 2$. More generally, when units in U are clustered into groups, the set of potential distributions, PD, is further constrained by the $g_j h_{bound}$ across the g groups in all variants.

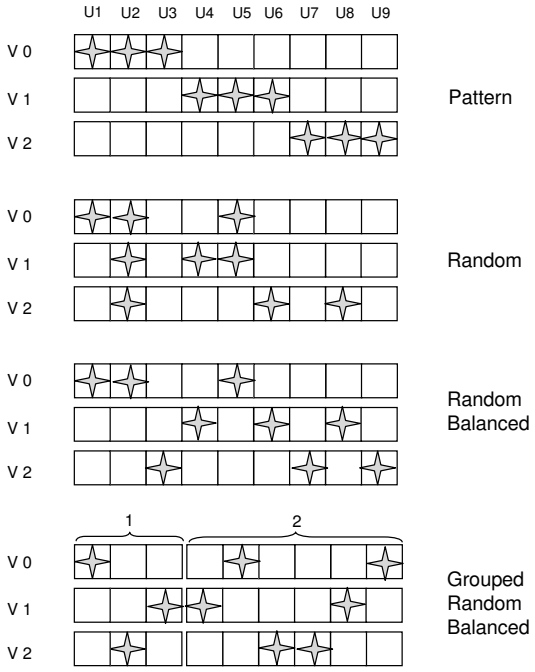

Figure 1: Sample Distributions.

3.3 Research Questions

Our overall objective is to learn how to best allocate a bounded number of probes into multiple deployed program variants to capture the maximum amount of field information. In this paper we assess a series of strategies that utilize different probe distribution mechanisms. Our assessment is guided by the following research questions which explore the multiple dimensions of this problem:

RQ1: Does the amount of information collected vary across the probe distribution strategies illustrated in Section 3.2?

RQ1a: What is the effect of varying bound levels?

RQ1b: What is the effect of varying the number of variants?

RQ2: Do unit-groupings that leverage probe execution likelihood make any difference?

RQ3: Does the utilization of prior distribution knowledge change our results?

3.4 Distribution Algorithms

To help us address the research questions, we have developed a meta-algorithm for probe distribution that can be instantiated to implement different distribution strategies that can accommodate different h_{bounds} (RQ1a) and number of variants (RQ1b). The algorithm can also incorporate groupings to allow gh_{bound} to be set individually within groups (RQ2). In addition, the algorithm uses an incremental approach of probe distribution that allows to leverage prior distribution information (RQ3): it allocates probes on a *release* basis, where a release R_i contains n_i variants with the same h_{bound} and consists of q_i groups.

The algorithm first separates the program units into the defined groups. This leads to the formation of "blocks', where a block is a single group in a single release. Figure 2 shows a set of $m \times q$ blocks, and zooms in on block B_{i2} which has 4 variants and 5 units, with $g_2 h_{bound} = 3$. The algorithm proceeds to distribute $g_2 h_{bound}$ probes to the units within that block, one variant at a time. A matrix representation is used to model this distribution. When a probe is placed in a unit of interest in a variant, it is represented by a 1. Otherwise it is represented as a 0.

The meta-algorithm is presented in Algorithm 1. The bold capitalized text in the algorithm represents specialization subroutines. The first specialization subroutine, **SET** gh_{bound}, allows for different strategies to bound the number of

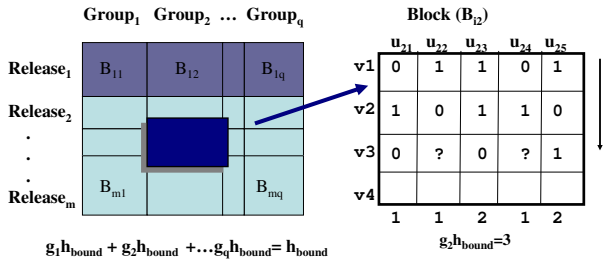

Figure 2: Building a Release.

probes in each group. The second subroutine, **INITIALIZE BLOCK**, enables different setups of the probe vector to accommodate results from previous distributions. For example, the algorithm could be instantiated as "memoryless", which would simply initialize all of the units equally, or it can provide "memory" such that it takes into account where probes have been allocated in previous releases in order to drive the current distribution. The third subroutine, **ALLOCATE PROBES**, determines where to allocate probes in a target variant. To better exemplify this subroutine, we instantiated three of the algorithms discussed in Section 3.2 which will also be the target of the following evaluation study.

The first instantiation, Algorithm 2, implements a type of *Pattern* distribution. We randomly select a starting unit and then sequentially allocate gh_{bound} probes, wrapping to the beginning of the unit set when we have reached the last unit. The next instantiation, Algorithm 3, represents a *Random* distribution where probes are allocated in random units without repetition within a variant until gh_{bound} probes have been placed.

Algorithm 1 Meta Algorithm to Build a Release

Divide *units* into *groups*
for each *group* **do**
    SET gh_{bound}
    INITIALIZE BLOCK
    for each variant **do**
        ALLOCATE PROBES
Re-assemble *groups*

Algorithm 2 Pattern Distribution

ALLOCATE PROBES (gh_{bound}, *offset*) {
if *offset* not initialized
    offset= selectRandomUnit(*units*)
for $i = 1$ to gh_{bound} **do**
    $s = (i+$ *offset*) mod $|units|$
    place probe in $unit\ s$
}

Algorithm 3 Random Distribution

ALLOCATE PROBES (gh_{bound}) {
$count$ = 0
$AvailableUnits$ = units in block
while $count < gh_{bound}$ and $AvailableUnits$ not empty **do**
    s = selectRandomUnit($AvailableUnits$)
    place probe in $unit\ s$
    remove $unit\ s$ from $AvailableUnits$
    count++
}

Algorithm 4 Random Balanced Distribution

```
ALLOCATE PROBES(gh_bound, minProbeUnitCount){
count = 0
AvailableUnits = SelectUnits(minProbeUnitCount)
while count < gh_bound do
    s = selectRandomUnit(AvailableUnits)
    place probe in unit s
    remove unit s from AvailableUnits
    if AvailableUnits is empty
        minProbeUnitCount++
        AvailableUnits =SelectUnits(minProbeUnitCount)
    count++
}
```

The last instantiated strategy, *Random Balanced*, (Algorithm 4), places probes in random units while keeping a balanced distribution at all times[1]. To maintain a balanced distribution, the algorithm tracks the number of probes assigned to the units with the least number of probes ($minProbeUnitCount$). The algorithm starts by adding all units that have been allocated a $minProbeUnitCount$ probes to the $AvailableUnits$ set. Next, it randomly selects a unit s from $AvailableUnits$, assigns a probe to s, and removes s from $AvailableUnits$. When $AvailableUnits$ is empty (same number of probes have been assigned to all units), $minProbeUnitCount$ is incremented and $AvailableUnits$ is re-initialized. This process is repeated until gh_{bound} probes are assigned.

4 Empirical Study

The following sections introduce the variables and metrics, the hypothesis, the object of study, the design and implementation of the study, and the threats to validity that may affect our findings.

4.1 Independent Variables

We manipulated the following four independent variables.

Probes Placement Techniques. We implemented the three probe allocation techniques presented in Section 3.4: Random (R), Pattern (P), and Random Balanced (RB). We assumed a probe allocation at a block level, where a block is a sequence of statements without any type of branches or jumps.

Bounds. We defined four levels of overhead: 5%, 10%, 25%, and 50% over the non-instrumented program. To approximate the number of probes corresponding to the chosen levels of overhead, we inserted a number of probes in random blocks of the program, ran the in-house test suite, measured the overhead, and adjusted the number of inserted probes until we converged to the target overhead level. This resulted in four bound levels (h_{bound})as defined by the following number of probes: 50, 100, 350, and 1000.

Groupings. In this study, we defined block groupings based on their likelihood of execution. We obtained the execution frequency for each block when running the in-house test suite, and then partitioned the blocks into 4 groups, with group 1 containing the least executed blocks, and group 4 contains the most executed blocks.

Number of variants. The maximum number of variants we can have is 36, which is the number of participants in this study, where each participant gets exactly one variant of the deployed program. The minimum number of variants is 1 where the all users get the same set of probes. In this study, we considered seven values of variants: 1, 6, 12, 18, 24, 30, and 36.

4.2 Dependent Variables

The dependent variables are associated with the captured field information value and the profiling activity costs. We have selected three metrics to quantify the value of the captured information. The metrics utilize the default allocation strategy, $Full$ (the complete set of probes is inserted in P), as a baseline for their computation.

The first metric is the percentage of program blocks covered in the field when using a probe allocation technique with respect to the coverage obtained by $Full$. This information is valuable to assess the thoroughness and representativeness of the in-house test suite [4], to predict the impact of program changes [14], or to revise reliability estimates [10].

[1]For the memoryless algorithm, $AvailableUnits$ is initialized with all the units, while in the version with memory the initialization considers the distribution performed in previous releases.

The second metric is the percentage of program hot-spots correctly identified, where hot-spots are defined as the 5% most frequently executed blocks. This information is valuable to guide test resource allocation [20]. To obtain such a percentage we identified the 140 (5% of the total number of blocks) most executed blocks using $Full$, we identified the 140 most executed blocks when a probe distribution technique is used, and we divided the number of common blocks in the lists by 140.

The third metric aims to capture an aspect of the profiling cost. Even though we constrain the number of probes inserted in a variant through the bound value, the number of probes actually executed in the field may vary, for example, due to the location of the probes. Hence, we computed the probe execution frequency in the field when a technique is applied, and divided by the probe execution frequency when $Full$ is utilized.

4.3 Hypotheses

We formally state the primary null hypotheses (assumed to be true until statistically rejected) in Table 1, corresponding to the research questions in Section 3.3.

Table 1: Hypotheses for Research Questions

RQ	Null Hypotheses: There is **no significant ...**
1a	$H1a1$: performance[2]difference between Random (R), Pattern (P), and Random-Balanced (RB) techniques. $H1a2$: difference when using different h_{bounds}. $H1a3$: interaction between techniques & h_{bounds}.
1b	$H1b1$: difference in performance when deploying different number of variants. $H1b2$: interaction between number of variants & h_{bound}.
2	$H21$: difference in a technique's performance when grouping is applied. $H22$: interaction between grouping & h_{bound}.
3	$H31$: difference in a technique's performance when information from previous distributions is utilized. $H32$: interaction between technique & number of variants in previous distributions. $H32$: interaction between technique & h_{bound} in previous distributions.

4.4 Object of Study

We chose MyIE as our object of study. MyIE is a web browser that wraps the core of Internet Explorer to add features such as a configurable front end, mouse gesture capability, and URL aliasing. MyIE was particularly attractive because its source code is available, it introduces many of the challenges of other large systems (e.g., size, interaction with 3rd party components, complex event handler, highly configurable), it is similar but not identical to other web browsers, and it has a small user base that we can leverage for our study. The version of MyIE source code we utilized, available for download at sourceforge.net, has approximately 41 KLOC, 64 classes, 878 methods, and 2793 blocks.

Object Preparation. To collect the field data, we instrumented MyIE source code to generate several types of traces, including a complete block trace utilizing the $Full$ strategy. During a user's execution, the block trace is recorded in a buffer. When the buffer is full, the information is compressed, packaged with a time and site ID stamp, and sent to the in-house repository if a connection is available or locally stored otherwise. An in-house server is responsible for accepting the package, parsing the information, and storing the data into the database.

Test Suite. We required a test suite to generate an initial assessment of the overhead and bounds, and identify the hotspots. To obtain such a test suite, we first generated a set of test cases to exercise all the menu items in MyIE at least once. After examining the coverage results, we added test cases targeting blocks that had not yet been exercised. We automated a total of 243 test cases yielded 79% of block and 90% of function coverage.

4.5 Deployments and Data Collection

We first performed a set of preliminary deployments to verify the correctness of the installation scripts, data capture process, magnitude and frequency of data transfer, and the transparency of the de-installation process. After this initial refinement period, we proceeded to perform a full deployment and started with the data collection. We sent e-mail to the members of our Department and various testing newsgroups (e.g. comp.software.testing, comp.edu, comp) inviting them to participate in the study and pointing them to our MyIE deployment web site for more information.[3] After

[2]As defined by the dependent variables in Section 4.2.

[3]http://cse-sgodd-009.unl.edu:8080/mapstext/index.html

3 months, there were 114 downloads, and 36 deployed sites that qualified for this study, which generated 378 user sessions, corresponding to 60,836 packages or 87MB of compressed data.

After the data was collected, different scenarios were simulated to help us answer the research questions. The particular simulation details such as the manipulated variables and the assumptions vary depending on the research question, so we address them individually within each study.

4.6 Threats to validity

There are several threats to the validity of this study. We explain the most likely threats and how we controlled them next.

From a generalization perspective, our findings are limited by the object of study, the data-collection process, and the user population. Although it is arguable whether the selected program is representative of the population of all programs, there are many similar browsers of similar size, implemented in the same programming language, and with a similar feature set which make our choice at least reasonable. The instrumentation mechanisms, and deployment and data collection processes were designed to capture as much data as possible to enable different simulation scenarios. Applying the treatments directly in the field (without using a-posteriori simulation) would have implied prohibitive collection costs at this stage of the investigation given the large number of treatments we considered. We attempted to balance data representativeness and power through the utilization of full data capture combined with simulation. The deployment and download process, as perceived by the users, was identical to many other sites offering software downloads and on-line patches. The users of the application were primarily students in our Department, which does not constitute a threat in itself as long as they utilize MyIE as they would other browsers. Since we did not exercise any control during the period of study, we expect the participants behaved as any other user would under similar circumstances. Further studies with other programs and subjects are necessary to confirm our findings.

From an internal validity perspective, the quality of the collected field data is the biggest threat as packages may be lost, corrupted, or not sent. We controlled this threat by adding different stamps to the packages to ensure their authenticity and to detect any anomalies in the collection process. Most of the lost data corresponds to the packages collected at the end of a session, but stored for transfer until a next session that never occurred. The large number of sessions (and packages) received makes such lost insignificant (less than 3 dozen packages).

From a construct validity perspective, we have chosen a set of metrics to quantify the value of the collected information that captures only a part of its potential meaning. Our choice of coverage, top 5%, and overhead is a function of our interest in exploiting field data for validation purposes, and our experience and infrastructure to analyze such data. We have chosen a subset of the potential levels for our treatments (e.g., levels for the number of variants, grouping strategy). Our choices of levels attempt to operationalize a spectrum of values that allow us to characterize the effects of the treatments. Future studies could mitigate these threats by considering complementary performance measures and treatment levels.

From a conclusion validity perspective, we are making inferences based on a few hundred sessions which may have limited the power to detect significant differences. However, we were able to reject various hypotheses and discover interesting findings. Future studies will have to consider a large number of sessions as the research questions become more focused.

5 Results and Analysis

In this section, we provide further details on the experimental settings, the results, and the analysis for each research question. For each research question, we first use graphical characterizations to perform an exploratory analysis of the hypothesis. Then, we formally test our hypothesis through an analysis of variance (ANOVA) to determine what independent variables had a significant effect on the dependent variables. If an ANOVA F-test suggests that the factors or their interactions cause significant variations on the dependent variables, we proceed to perform a Bonferroni multiple test analysis to investigate what levels of factors contributed significantly to the difference.

5.1 Study 1a: Effectiveness vs. Bounds

This first study addresses RQ1a, comparing the performance of three probe placement techniques across different bound levels that restrict the number of probes that can be placed on a given variant.

5.1.1 Simulation Setting

Since the deployed MyIE contained a profiling probe in each program unit, the data collection process resulted in a complete block trace from each deployed instance (as per Section 4.4). We utilized this rich data set to simulate the three probe allocation techniques.

Each simulation generated n variants, where each variant consisted of a vector of size equal to the number of units in the program. Cells in each vector were initialized with zeroes, and then populated with h_{bound} probes according to the allocation rules specified by the simulated technique. Each vector was then utilized to mask the data collected from a specific deployed instance, simulating what would have been collected if a true variant would have been deployed to that particular instance. We performed the variant generation and assignment process ten times to account for potential variations due to the random assignment of probes to variants, and from variants to instances.

For this study, we assume that the number of variants $n = 36$, that is, we have as many variants as deployed instances which constitutes an upper bound on the potential distribution. (We explore the effect of varying the number of variants in the next section).

5.1.2 Results

The top box plot[4] in Figure 3 depicts the percentage of block coverage achieved by the deployed instances when utilizing the Random (R), Pattern (P), and Random Balanced (RB) techniques compared to a technique that puts all the probes which we call the *Full* technique. The middle figure depicts the box plots of the three techniques in accurately identifying the top 5% most executed block probes. The bottom figure shows the execution overhead of the three techniques when compared to *Full*.

As can be seen from Figure 3-top, both P and RB perform better than R in achieved block coverage for the smaller bound levels. However, while the performance of P and RB seems to be equal, P appears to have a larger variance than RB suggesting that perhaps RB is more reliable than P in providing a more even representation of the coverage achieved by each deployed unit.

To formally determine whether the observed tendencies were not the result of chance (e.g., getting lucky with the random assignment), we performed an analysis of variance (ANOVA) on the dependent variable (coverage) and the two independent variables (techniques and h_{bound}). The summary of the analysis is shown in Table 2. The columns represent the hypotheses associated with the treatments, the treatments or effects applied to the test, sum of squares, degree of freedom, mean of squared, F-value, and p-value, respectively. The p-values in Table 2 show that the achieved coverage in the deployed sites varies significantly when different techniques are used, when different bounds are set, and that certain combinations of techniques and bounds exhibit distinct behavior in terms of coverage, rejecting all the null hypothesis of $H1a$'s.

Table 2: Univariate Tests of Significance for Coverage of RQ1a

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	399970.8	1	399970.8	156697.6	0.00
$H1a1$	Tech	673.2	2	336.6	131.9	0.00
$H1a2$	h_{bound}	53752.1	3	17917.4	7019.5	0.00
$H1a3$	Tech*h_{bound}	573.7	6	95.6	37.5	0.00
-	Error	275.7	108	2.6		

We then conducted a Bonferroni analysis, a conservative comparison of all pairwise combinations of techniques and bounds, to quantify when a technique did really have an effect on the achieved coverage. The result of this analysis, as presented in Table 3, consists of a list of all potential combinations of techniques and bound, their mean values, and a letter representing the group to which the combination belongs (i.e., two combinations that are assigned different letters are significantly different). We found that when h_{bound} is 50 and 100, P and RB enabled significantly higher field coverage than R. However, for greater h_{bound}, we could not detect differences in the techniques' performance.

The second plot of Figure 3 serves to identify the 5% most executed blocks. We see that the average correctly identified blocks resulting from the RB allocation seems to perform consistently better R or P. We again observe that the P allocation generates variants that obtain information which value can vary quite a bit (e.g., for $h_{bound} = 50$ the standard deviation for RB is 3.8 while the standard deviation for P is 11.3).

[4]In the box plots the middle line of a box represents the average of the 10 observations, the outer box represents the (average value $\pm$ standard deviation), and the whiskers represent minimum and maximum observed values.

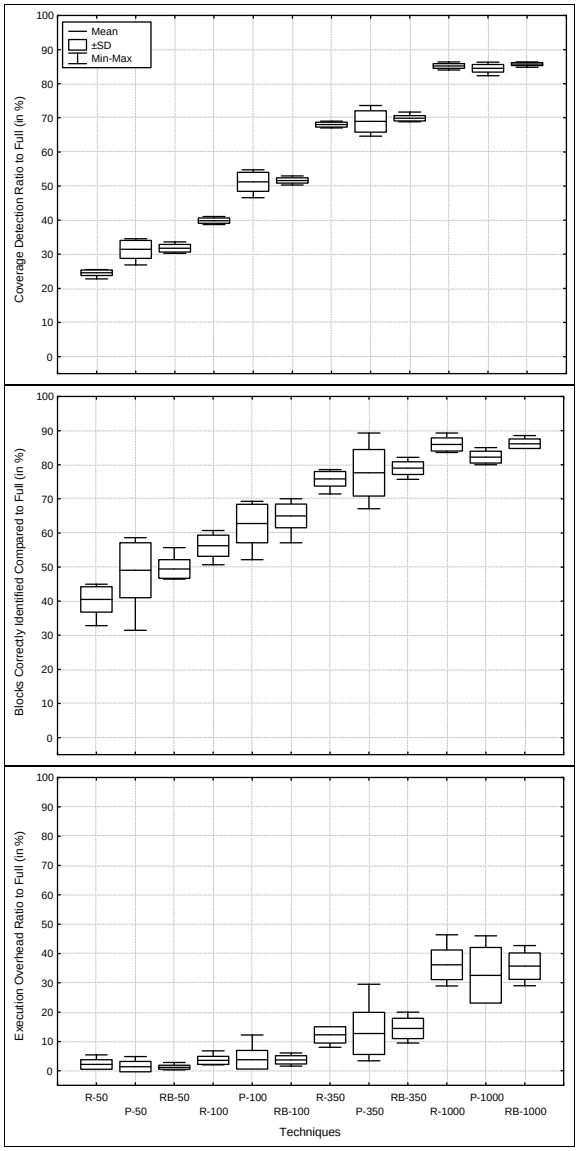

Figure 3: RQ1a.

Table 3: Bonferroni Test for Coverage of RQ1a

Tech	h_{bound}	Mean of Coverage Ratio	Homogeneous Group
R	50	25	A
P	50	31	B
RB	50	32	B
R	100	40	C
P	100	51	D
RB	100	52	D
R	350	68	E
P	350	69	E
RB	350	70	E
P	1000	84	F
R	1000	85	F
RB	1000	86	F

The p-values of the ANOVA test for identifying the top 5% most executed blocks, as shown in Table 4, again rejects all null hypothesis of $H1a$'s in Table 1, revealing that the technique, h_{bound}, and interaction between the technique and h_{bound} have a significant impact in accurately identifying 5% most executed blocks. The Bonferroni analysis in Table 5 shows that when h_{bound} values are 50 and 100, utilizing R results in worse information than when using P and RB, but not significant difference between these last two allocation techniques was significant. For higher bound levels the techniques did not seem to affect the accuracy of the identification.

Table 4: Univariate Tests of Significance for Identifying 5% Most Executed Blocks of RQ1a

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	546557.2	1	546557.2	32282.83	0.00
$H1a1$	Tech	562.9	2	281.4	16.62	0.00
$H1a2$	h_{bound}	26512.1	3	8837.4	521.99	0.00
$H1a3$	Tech*h_{bound}	505.7	6	84.3	4.98	0.00
-	Error	1828.5	108	16.9		

Table 5: Bonferroni Test for Identifying 5% Most Executed Blocks of RQ1a

Tech	h_{bounds}	Mean of % of Block	Homogeneous Group
R	50	40	A
P	50	49	B
RB	50	49	B
R	100	56	C
P	100	63	D
RB	100	65	D
R	350	76	E
P	350	78	E
RB	350	79	E
P	1000	82	E F
R	1000	86	F
RB	1000	86	F

Last, for the cost of the execution overhead, we see in the bottom plot of Figure 3 that there seems to be little overhead difference across techniques when h_{bound} is fixed. This confirms our intuition that more valuable information can be obtained through strategic probe distributions without significant additional execution overhead. The ANOVA analysis presented in Table 6 further confirms our intuition by not rejecting hypotheses $H1a1$ and $H1a3$. The ANOVA also rejects $H1a2$. We shows the result of Bonferroni test for hypothesis $H1a2$ in Table 7. The result shows that the overhead is significantly different as the bound values increases.

Table 6: Univariate Tests of Significance for Execution Overhead of RQ1a

Hypo-thesis	Effect	SS	DF	MS	F	p
-	Intercept	21240.37	1	21240.37	1120.723	0.00
$H1a1$	Tech	28.80	2	14.40	0.760	0.47
$H1b2$	h_{bound}	20749.41	3	6916.47	364.939	0.00
$H1b3$	Tech* h_{bound}	78.62	6	13.10	0.691	0.66
-	Error	2046.86	108	18.95		

Table 7: Bonferroni Test for Execution Overhead of RQ1a

h_{bounds}	Mean of Overhead	Homogeneous Group
50	2	A
100	4	A
350	13	B
1000	35	C

Implications. *When the specified acceptable overhead that bounds the profiling effort is small (less than 10% in our study), probe allocation techniques that balance the placement across units are more effective in retaining the value of field data. When many probes can be placed in a variant, a random allocation technique seem to suffice.*

5.2 Study 1b: Effectiveness vs. Variants

This study addresses RQ1b, investigating the performance of the RB probe distribution technique (the most promising from RQ1a) under a different number of variants.

5.2.1 Simulation Setting

For this study, RB is used to generate releases with 1, 6, 12, 18, 24, 30, 36 variants, repeating the same simulation procedure described for RQ1a in Section 5.1.1. We repeated this process for each of the bound values (50, 100, 350, 1000). Each variant was then utilized to mask the data obtained from one or more deployed sites to simulate a true-variant deployment across sites. Because the number of sites was not always divisible by the number of generated variants, variants might get assigned to a different number of sites (e.g., when deploying 24 variants, 12 variants were deployed twice, and 12 variants were deployed once). To avoid nuisance variables (e.g., the random assignment of probes to variants, the random assignment of variants to sites) we performed the whole process 10 times.

5.2.2 Results

The coverage achieved by the deployed instances with the probes distributed by RB for bound values 50, 100, 350, and 1000 is shown in Figure 4-top. Each point in the graph represents the average of overall coverage detection values from the 10 runs. The graph suggests that when deploying just one variant, the coverage obtained suffers even under a high h_{bound} values. For $h_{bound} = 1000$, the average coverage achieved with 1 variant is 36% of the one achieved with $Full$ technique, while the average coverage with 6 variants rises to 85%. However, coverage gains resulting from increases in the number of variants seems to be asymptotic and limited with higher h_{bound} levels.

The ANOVA for the coverage metric, summarized in Table 8, indicates that the number of variants, and the interactions between the number of variants and bound values have a statistically significant effect. Hence, rejecting all the null hypothesis for $H1b$'s. The Bonferroni results, given in Table 11, confirm our previous conjectures: for low h_{bound} the number of variants is important, and having just 1 variant is consistently loosing significant amounts of valuable information. The Bonferroni test indicates that we can counter low h_{bound} values by deploying more variants (and vice versa). For example, one could lower the h_{bound} from 100 to 50 probes and still obtain better coverage data by increasing the number of variants from 6 to 18.

Table 8: Univariate Tests of Significance for Coverage of RQ1b

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	625856.3	1	625856.3	1438401	0.00
$H1b1$	# of Variants	62390.6	6	10398.4	23899	0.00
-	h_{bound}	153036.0	3	51012.0	117241	0.00
$H1b2$	# of Variants*h_{bound}	13012.2	18	722.9	1661	0.00
-	Error	109.6	252	0.4		

Figure 4-middle plots the effectiveness of the RB technique in identifying top 5% most executed blocks. The line plots seem to follow the same trend as the ones for coverage. The ANOVA and Bonferroni results, summarized in Table 9 and Table 13 respectively, confirms the significant effect of the number of variants and its interaction with h_{bound} on the accuracy of the identification of the top executed blocks.

Table 9: Univariate Tests of Significance for Identifying 5% Most Executed Blocks of RQ1b

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	807019.0	1	807019.0	109817.0	0.00
$H1b1$	# of Variants	84796.6	6	14132.8	1923.2	0.00
-	h_{bound}	92741.8	3	30913.9	4206.7	0.00
$H1b2$	# of Variants*h_{bound}	19636.1	18	1090.9	148.4	0.00
-	Error	1851.9	252	7.3		

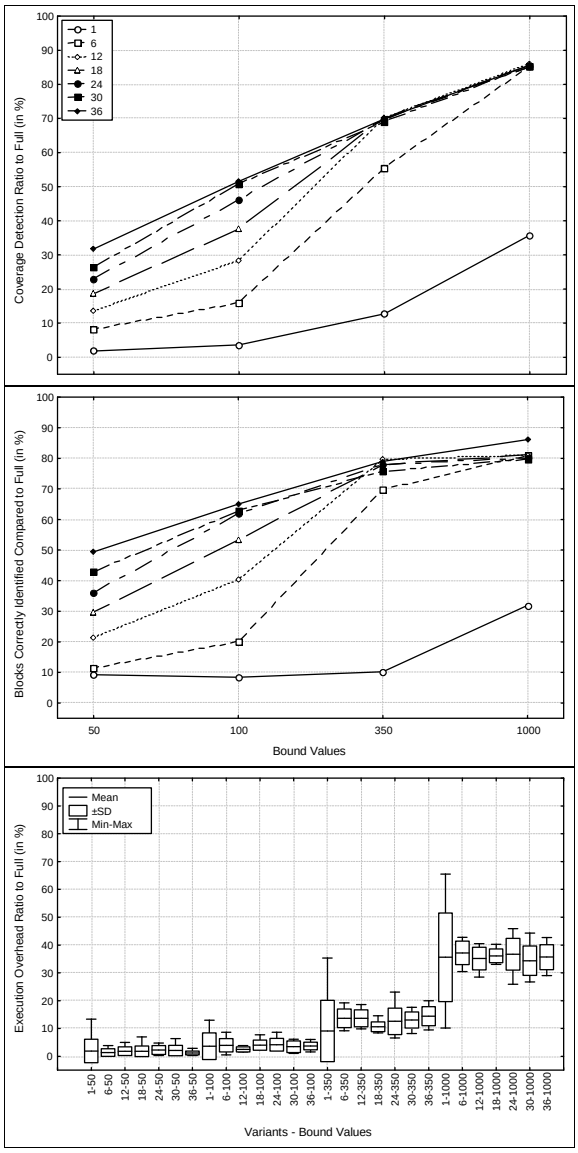

Figure 4: RQ1b.

Figure 4-bottom depicts the box plot for the overhead execution of RB. For each h_{bound} value, the execution overhead across releases with the different number of variants seem to be about the same. Note, however, that when one variant is deployed the variance in the execution overhead at least doubles the variation that is observed when more variants are deployed. For example, the standard deviation when the bound value is 50 for 1 variant is 4 and at most 1.9 when a greater number of variants is deployed. The ANOVA, given in Table 10, shows that only the difference in h_{bound} values had a significant effect in execution overhead. The Bonferroni test over the h_{bound} value confirms the analysis.

Table 10: Univariate Tests of Significance for Execution Overhead of RQ1b

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	50478.54	1	50478.54	2211.277	0.00
$H1b1$	# of Variants	64.49	6	10.75	0.471	0.83
-	h_{bound}	51387.31	3	17129.10	750.362	0.00
$H1b2$	# of Variants*h_{bound}	233.37	18	12.96	0.568	0.92
-	Error	5752.60	252	22.83		

Table 11: Bonferroni Test for Coverage of RQ1b

# of Variants	h_{bound}	Mean of Coverage Ratio	Homogeneous Group
1	50	2	A
1	100	4	B
6	50	8	C
1	350	13	D
12	50	14	D
6	100	16	E
18	50	19	F
24	50	23	G
30	50	26	H
12	100	28	I
36	50	32	J
1	1000	36	K
18	100	37	L
24	100	46	M
30	100	51	N
36	100	52	N
6	350	55	O
30	350	69	P
24	350	70	P
36	350	80	P
18	350	70	P
12	350	70	P
30	1000	85	Q
24	1000	85	Q
6	1000	85	Q
18	1000	85	Q
36	1000	85	Q
12	1000	86	Q

Table 12: Bonferroni Test for Execution Overhead of RQ1b

h_{bounds}	Mean of Overhead	Homogeneous Group
50	2	A
100	4	A
350	12	B
1000	36	C

Implications. *When the acceptable overhead is low, it is always beneficial to deploy more variants in order to capture more information from the field. Furthermore, software engineers intending to reduce h_{bound} while retaining the value of collected field data may be able to do so by deploying additional variants.*

5.3 Study 2: Unit-Grouping on Distributions

Groupings enable the prioritization of the probe allocation across units so that certain units are profiled in more deployed instances than others. Each group of units may have different weight, reflecting our degree of interest in the behavior they may expose in the field. This section addresses RQ2, investigating the impact of unit-grouping based on previous knowledge about software usage.

5.3.1 Simulation Setting

In this study, we considered groupings based on the units' previous execution frequency. We first collected the blocks' execution frequency when running the in-house test suite. Then, we classified the blocks into four groups defined by the execution frequency quartiles. Last, we computed the group weights that determine what percentage of the h_{bound} probes are assigned to each group j ($g_j h_{bound}$).

Table 13: Bonferroni Test for Identifying 5% Most Executed Blocks of RQ1b

# of Variants	h_{bound}	Mean of Num. of Block	Homogeneous Group
1	100	8	A
1	50	9	A
1	350	10	A
6	50	11	A
6	100	20	B
12	50	21	B
18	50	30	C
1	1000	32	C D
24	50	36	D E
12	100	40	E F
30	50	43	F
36	50	49	G
18	100	53	G
24	100	62	H
30	100	63	H
36	100	65	H I
6	350	70	I
30	350	76	J
18	350	78	J K
24	350	78	J K
36	350	79	J K
12	350	79	J K
30	1000	80	J K
24	1000	80	J K
12	1000	81	K
6	1000	81	K
18	1000	81	K
36	1000	86	L

The assignments of weights is tied to the target dependent variable. For coverage it seems reasonable, for example, to assign higher priority to units that have lower execution frequency because they are the least observed (and perhaps the least understood so we want more variants with probes in those units). In the case of the identification of the top 5% executed blocks, or other variables, the assignment of priorities may be different.

For this simulation we focused on the coverage achieved by deployed instances when grouping is utilized (the results will not include the top-5%). We compare the results obtained from RB and GRB (Grouped Random Balanced). The weights are computed based on the group's size (bigger groups get more probes) and the total frequency of the units in the group (least executed groups in-house get more probes when deployed). We also ensured that all units executed by the in-house suite received at least one probe in one variant (so that even the units most likely to be executed are instrumented and have a change to be covered in the field).

5.3.2 Results

The graphs in Figure 5 compare the performance of GRB versus RB at bound values of 50, 100, 350, and 1000. The top graph shows the coverage achieved by RB versus GRB, the bottom graph of Figure 5 shows their execution overhead.

We can observe that applying GRB consistently yields greater coverage gains than RB. However, this benefit diminishes as the bound values increases (the prioritization through grouping becomes less relevant as more probes can be placed in a variant). We tested these observations more formally through an ANOVA, presented in Table 14, which shows that the coverage achieved through GRB is significantly different from RB, rejecting null hypothesis $H21$, but that the degree of difference depends on the utilized h_{bound}, rejecting null hypothesis $H22$. Still, the Bonferroni test, in Table 15, placed each combination of technique and h_{bound} in different groupings. This confirmed that even with a large number of probes can be allocated across variants, grouping can lead to higher coverage gains.

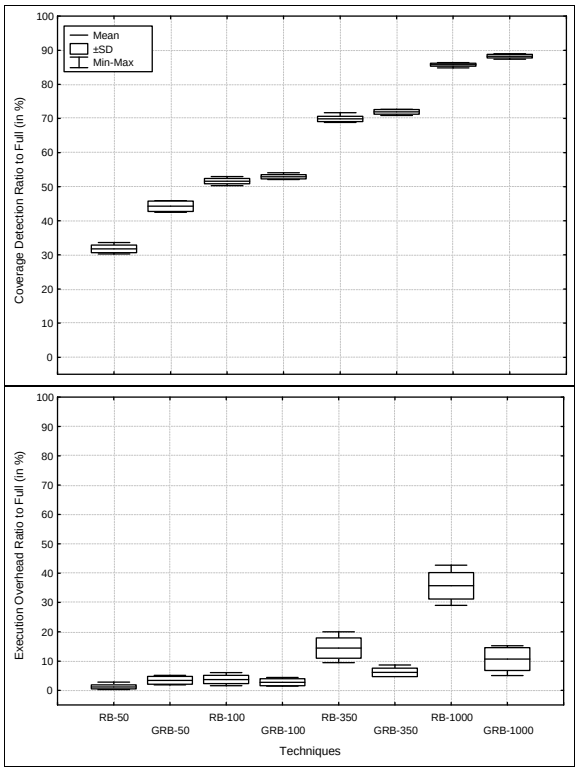

Figure 5: RQ2.

Table 14: Univariate Tests of Significance for Coverage of RQ2

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	307987.6	1	307987.6	400738.1	0.00
$H21$	Tech	422.0	1	422.0	549.1	0.00
-	h_{bound}	27498.1	3	9166.0	11926.4	0.00
$H22$	Tech*h_{bound}	422.5	3	140.8	183.2	0.00
-	Error	55.3	72	0.8		

Table 15: Bonferroni Test for Coverage of RQ2

Techniques	h_{bound}	Mean of Coverage Ratio	Homogeneous Group
RB	50	32	A
GRB	50	44	B
RB	100	52	C
GRB	100	53	D
RB	350	70	E
GRB	350	72	F
RB	1000	86	G
GRB	1000	88	H

Table 16: Univariate Tests of Significance for Execution Overhead of RQ2

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	7633.046	1	7633.046	1114.584	0.00
$H21$	Tech	1276.680	1	1276.680	186.422	0.00
-	h_{bound}	5564.932	3	1854.977	270.865	0.00
$H22$	Tech*h_{bound}	2212.932	3	737.644	107.711	0.00
-	Error	493.080	72	6.848		

When we apply groupings to RB, we bias the probe allocation toward the least executed units, so that they are profiled in more deployed instances. We expect that, by putting emphasis in the least executed units, the execution overhead of GRB would be less than for RB. Figure 5-bottom confirms our conjecture which seems more evident for higher h_{bound} values. The ANOVA and Bonferroni analysis, presented in Tables 16 and 17, formally confirm our findings, showing that GRB overhead is significantly lower than the one for RB, especially toward higher h_{bound} value.

Table 17: Bonferroni Test for Execution Overhead of RQ2

Techniques	h_{bound}	Mean of Overhead	Homogeneous Group
RB	50	1	A
GRB	100	3	A B
GRB	50	3	A B
RB	100	4	A B
GRB	350	6	B
GRB	1000	11	C
RB	350	14	C
RB	1000	36	D

Implications. *Distributions that consider unit-grouping can capture significantly more field coverage information, while providing reduction in the overhead execution. This could enable further probe allocation within the constraints of an acceptable overhead.*

5.4 Study 3: Memory vs. Memory-less

This section addresses RQ3, investigating the effect of utilizing prior probe distribution knowledge to increase the effectiveness of a probe distribution across variants.

5.4.1 Simulation Setting

We simulate a common scenario where the utilization of previous probe distribution with different h_{bound} is likely to occur: beta testing. Beta testing is a relatively common practice in which the software is distributed to a set of friendly sites for its preliminary assessment. These friendly users are willing to tolerate higher levels of profiling overhead (higher h_{bound}) in order to provide additional information about the software behavior to the developers.

In this study, we assumed that the regular users are willing to operate with $h_{bound} = 50$, while for the friendly beta users we experimented with h_{bound} values of 100, 350, and 1000 probes. We explored various ratios of beta users to field users: 10-90 (10% of 36 users as beta, which is 3 users, and 90% as field users, which is 33), 30-70, and 50-50.

We again utilized the Random Balanced (RB) technique to generate the probe distributions across variants. We considered two variations of RB: Random Balanced with memory (RBM), and Random Balanced without memory or memory-less ($RBML$). For both techniques we placed probes through the generation of two sets of distributions: (1) with the number of variants equal to the number of beta users and the target h_{bound} value, and (2) with the number of field users as the number of variants with $h_{bound} = 50$. The two techniques differ in how they generate the second distribution, where probe distribution history in previous variants is considered for RBM but not for $RBML$ (recall that the **INITIALIZE BLOCK** in Algorithm 1 for RBM calculates which probes have been allocated in previous variants and uses this information when determining which probes to put in the next variant).

5.4.2 Results

The top and middle graphs in Figure 6 plot the percentage of coverage achieved and percentage of correctly identified top 5% blocks for RBM and $RBML$ respectively. For both graphs, we can observe a similar trend. When h_{bound} for beta users is 100, RBM detects more coverage than $RBML$ technique regardless the ratio of beta users to regular users. This is true also when the h_{bound} is 350 and the beta users are 10% of all users population. For other combinations of values, $RBML$ performs as well as RBM. This seems to suggest that having a technique with memory is more important if the previous distributions were limited either in terms of variants or in terms of number of allocated probes.

The ANOVA test, summarized in Table 18 and 20, indicates that all factors considered (technique, bound, and ratio) and their interactions have a significant effect on coverage and identification accuracy. The Bonferroni test, summarized in Table 19 and 23, confirms the previous conjectures. When using RBM, the number of beta users can be reduced and still achieve the same result as the $RBML$ techniques. The Bonferroni analysis also confirms that there is a tradeoff

between the h_{bound} value and the beta users ratio; if we can afford to put more probes on the variants deployed to beta users, we do not need as many beta users.

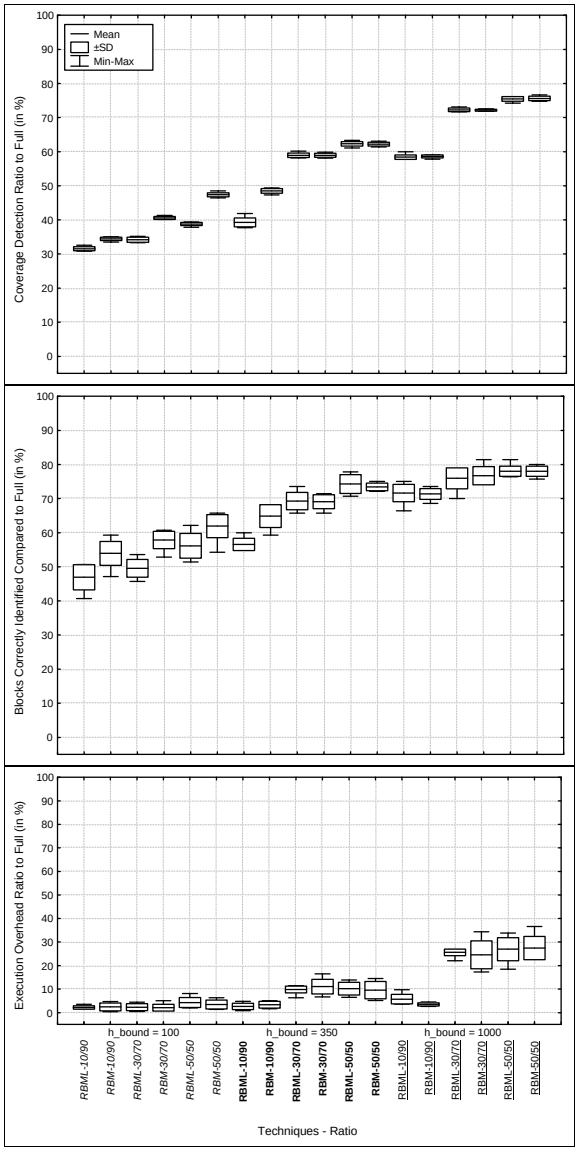

Figure 6: RQ3.

Table 18: Univariate Tests of Significance for Coverage of RQ3

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	522292.0	1	522292.0	1274431	0.00
$H31$	Tech	409.8	1	409.8	1000	0.00
-	Ratio	7377.4	2	3688.7	9001	0.00
-	h_{bound}	28834.0	2	14417.0	35179	0.00
$H32$	Tech*Ratio	31.7	2	15.8	39	0.00
$H33$	Tech*h_{bound}	257.4	2	128.7	314	0.00
-	Ratio*h_{bound}	736.9	4	184.2	450	0.00
-	Tech*ratio*h_{bound}	338.3	4	84.6	206	0.00
-	Error	66.4	162	0.4		

Table 19: Bonferroni Test for Coverage of RQ3

Techniques	ratio	h_{bound}	Mean of Coverage Ratio	Homogeneous Group
$RBML$	10	100	31	A
$RBML$	30	100	34	B
RBM	10	100	34	B
$RBML$	50	100	39	C
$RBML$	10	350	39	C
RBM	30	100	41	D
RBM	50	100	47	E
RBM	10	350	48	F
$RBML$	10	1000	58	G
RBM	10	1000	59	G
RBM	30	350	59	G
$RBML$	30	350	59	G
RBM	50	350	62	H
$RBML$	50	350	62	H
RBM	30	1000	72	I
$RBML$	30	1000	72	I
$RBML$	50	1000	75	J
RBM	50	1000	76	J

Table 20: Univariate Tests of Significance for Identifying 5% Most Executed Blocks of RQ3

Hypothesis	Effect	SS	DF	MS	F	p
-	Intercept	780783.5	1	780783.5	110130.3	0.00
$H31$	Tech	460.3	1	460.3	64.9	0.00
-	Ratio	2686.8	2	1343.4	189.5	0.00
-	h_{bound}	13460.7	2	6730.4	949.3	0.00
$H32$	Tech*ratio	85.9	2	42.9	6.1	0.003
$H33$	Tech*h_{bound}	366.8	2	183.4	25.9	0.00
-	Ratio*h_{bound}	298.4	4	74.6	10.5	0.00
-	Tech*ratio*h_{bound}	193.2	4	48.3	6.8	0.00
-	Error	1148.5	162	7.1		

Table 21: Bonferroni Test for Identifying 5% Most Executed Blocks of RQ3

Techniques	ratio	h_{bound}	Mean of Num. of Block	Homogeneous Group
$RBML$	10	100	47	A
$RBML$	30	100	50	A B
RBM	10	100	54	B C
$RBML$	50	100	56	C
$RBML$	10	350	56	C
RBM	30	100	58	C D
RBM	50	100	62	D E
RBM	10	350	65	E F
RBM	30	350	69	F G
$RBML$	30	350	69	G
RBM	10	1000	71	G H
$RBML$	10	1000	72	G H I
RBM	50	350	73	G H I J
$RBML$	50	350	74	H I J K
$RBML$	30	1000	76	I J K
RBM	30	1000	77	J K
RBM	50	1000	78	K
$RBML$	50	1000	78	K

The bottom graph in Figure 6 plots the execution overhead of each technique. The graph suggests that the execution overhead across techniques are similar within the same bound and ratio values. The ANOVA (Table 22) confirms these findings, showing that only ratio, h_{bound}, and their interaction have a significant impact in execution overhead. The significance of h_{bound} was expected based on the findings of the previous studies, and the significance of ratio seems intuitive since the higher the ratio of the beta user to the field user, the higher the average of probes per deployed instance.

Table 22: Univariate Tests of Significance for Execution Overhead of RQ3

Hypo-thesis	Effect	SS	DF	MS	F	p
-	Intercept	17459.68	1	17459.68	2224.295	0.00
$H31$	Tech	3.13	1	3.13	0.399	0.53
-	Ratio	3874.46	2	1937.23	246.796	0.00
-	h_{bound}	8222.32	2	4111.16	523.746	0.00
$H32$	Tech*ratio	1.53	2	0.76	0.097	0.91
$H33$	Tech*h_{bound}	13.73	2	6.86	0.875	0.42
-	Ratio*h_{bound}	3063.35	4	765.84	97.565	0.00
-	Tech* ratio*h_{bound}	28.51	4	7.13	0.908	0.46
-	Error	1271.62	162	7.85		

Table 23: Bonferroni Test for Identifying 5% Most Executed Blocks of RQ3

Techniques	ratio	h_{bound}	Mean of Num. of Block	Homogeneous Group
$RBML$	10	100	47	A
$RBML$	30	100	50	A B
RBM	10	100	54	B C
$RBML$	50	100	56	C
$RBML$	10	350	56	C
RBM	30	100	58	C D
RBM	50	100	62	D E
RBM	10	350	65	E F
RBM	30	350	69	F G
$RBML$	30	350	69	G
RBM	10	1000	71	G H
$RBML$	10	1000	72	G H I
RBM	50	350	73	G H I J
$RBML$	50	350	74	H I J K
$RBML$	30	1000	76	I J K
RBM	30	1000	77	J K
RBM	50	1000	78	K
$RBML$	50	1000	78	K

Implications. *When the previous probe distribution across variants are limited, either in terms of the number of variants or the number of probes, techniques that consider how the probes were distributed in previous releases can yield more information.*

6 Discussion

The sampling process to dynamically determine which profiling probes will be executed during the program's execution [11] gives each probe a fair chance of being selected. This selection process requires random number generation, decrementing the value at multiple locations, and comparing the generated value to determine the path to follow (probe-free fast path or probe-loaded slow path). These requirements result in additional overhead on top of the probes execution. It can be difficult to guarantee that this additional overhead is within an acceptable bound because of its dependence to the program's runtime behavior.

On the other hand, the probe allocation techniques we have proposed *statically* determine where to place a subset of the probes, which makes them free from any additional and hardly predictable overhead, but as effective as the initial

allocation. In this section we explore what gains can be achieved through the combinations of a static technique (RB) and the dynamic-sampling technique.

Since the data we gathered from the field does not allow us to exactly reproduce the user sessions, we had to simulate the dynamic sampling process by performing a walk-through the collected field block traces following the specifications stated in [11]. We sampled each of the block trace files that are obtained using full profiling from the field with the following mechanisms. First, from a uniform distribution, we generated a random integer, i, between 0 and 99 to represent the inter-arrival time. This gave each probe the probability of 1/100 of being sampled. Then, we picked the ith block from the trace as if the block was chosen to be executed. This process of generating a random number and selecting the ith block, continuing from the previously picked block in the trace was repeated until the end of trace. Consequently, each block trace file from $Full$ yielded a subset of the block trace.

To simulate the combination technique, we first created subsets of the block traces utilizing the blocks instrumented by RB as filters, and then applied the dynamic sampling.

This simulation mechanism lets us measure the execution frequency of the probes (a proxy for overhead), but it does not account for the additional overhead caused by the extra logic required by dynamic sampling to select an execution path. To account for this overhead we considered three possible scenarios: 1) Light: probe execution overhead is an order of magnitude smaller than the extra logic overhead, 2) Medium: probe execution overhead is equivalent to the extra logic overhead, 3) Heavy: probe execution overhead is an order of magnitude greater than the extra logic overhead. (These arbitrarily selected values are just meant to illustrate the tendencies as the cost ratio of probe execution and extra logic required to dynamically sample probes changes.)

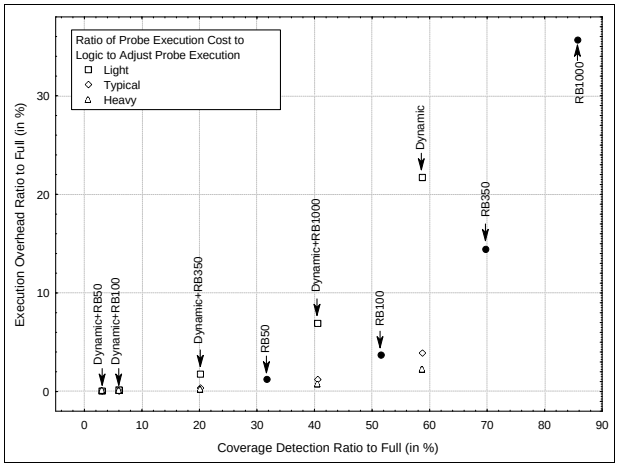

Figure 7: Overall Coverage Detection.

Figure 7 plots the average of each technique's coverage against their average overhead, and it exposes some interesting points. First, the outcome of any performance comparison between $Dynamic$ and RB depends on the ratio. When utilizing $Dynamic$ sampling the collected coverage is close to 60%. If the utilized probes were light (e.g., update a counter in the case of most coverage tools), then the additional logic to control what probes to execute does not seem worth it since utilizing $RB350$ resulted in 11% more coverage with 7% less overhead. However, if the probes were heavy (e.g., a set of complex assertions), then $Dynamic$ can be almost as efficient as $RB50$ and result in 27% more coverage. Second, applying $Dynamic$ on top of RB reduced the overhead, specially for higher h_{bounds}. Although the reduction in overhead was expected, the magnitude of the loss in effectiveness was surprising ranging from 30% to almost 50%, indicating perhaps that the probe sampling rate utilized in the literature (each probe has a probability of 1/100 of being executed) may not be appropriate to effectively collect coverage information.

In identifying the top 5% most executed block probe, as we can observe from Figure 8, the Dynamic technique performs better than the RB techniques, both in the overhead execution and in the percentage of blocks correctly identified. Even with the Dynamic technique with a light ratio, the most expensive Dynamic technique in term of overhead, the reduction in overhead execution reaches 14% compared to RB1000, the most aggressive RB technique, and correctly identifies 11% more.

Similar to the trend we saw with the coverage detection, the combination techniques, however, always identify less correct blocks than the RB techniques with the same bound value; even though they are able to provide some reduction in overhead. This overhead reduction increases as the bound value increases.

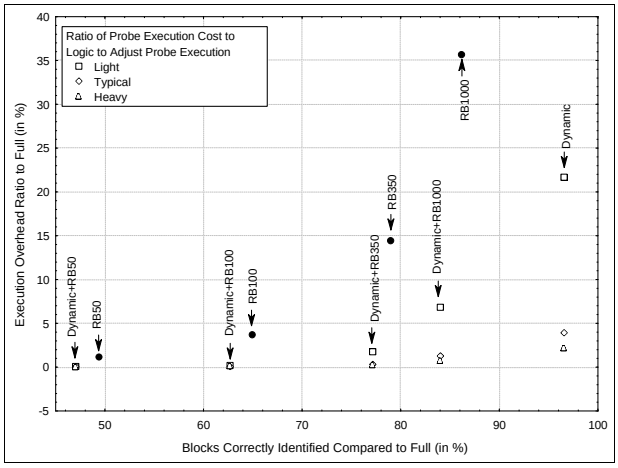

Figure 8: Identifying top 5%.

The reduction in the execution overhead gives the advantage of selecting a combination technique with a higher bound value while satisfying the allowable overhead constraint. For example, if we can afford only to put 50 probes into the variant, we can choose between the combination of Dynamic and RB50 techniques (Dynamic+RB50), Dynamic+RB100, Dynamic+RB350 with medium or heavy ratio, Dynamic+RB1000 with heavy ratio, or RB50 technique. If the ratio is heavy, we would prefer Dynamic+RB1000 technique as it gives higher coverage detection and identifies more correct blocks. If the ratio is medium or light, we would prefer $RB50$. Therefore, the ratio between the cost of probe execution and the cost of adjusting logic plays an important part in determining which technique is the most effective and efficient.

7 Conclusion and Future Work

Profiling overhead limits what we can observe and learn from deployed software. To address this limitation, this paper investigates how to distribute probes across variants to reduce profiling overhead while retaining the field information value. We have formalized the problem of distributing probes across variants, presented several distribution techniques, and examined their performance in a study that manipulated various levels of overhead, deployment costs, targeted behavior, and changes in the pool of deployed sites and corresponding profiling policies.

Our findings regarding the levels of acceptable overhead indicate that, when the bound is small, the probes placement strategies can significantly affect the value of the collected information (coverage and top hot-spots identification). Techniques that emphasized a balance distribution across variants (Pattern and Random Balanced) perform significantly better. Also, techniques with simple pattern-like allocations may generate variants that suffer a large variance in overhead and information value.

The deployment costs which constrain the number of manageable variants, show that a larger number of random-balanced variants is always beneficial. However, the benefit of deploying an additional variant decreases as the number of already deployed variants increases. It is also interesting to see the inverse relationship between the number of variants and bound. In our study, having 10 additional variants enabled up to a 10 fold reduction in the bound level.

When considering the particular program behavior in terms of execution likelihood, we discovered that when the bound value is small, grouping the units based on execution frequency causes a significant increase in the value of the collected information. By utilizing grouping, we are also able to reduce the execution overhead, where the reduction increases as the bound value increases. This suggest that utilizing proper grouping may enable the allocation of more probes in the deployed program.

Last, as variants are deployed and new probe allocation policies are put in place, we found that techniques taking into consideration how the probes were previously allocated perform significantly better under small bound levels. There is also a trade-off between the number of variants and the bounds constraint in the previous release. Having more variants compensates for a more restrictive constraint in the previous allocation.

Generally, if we can afford the high overhead and the cost of generating and maintaining many variants to deploy, distribution techniques do not seem to be a factor in the quality of collected information. However, if allowable overhead is low and the number of variants that can be generated is limited, distribution techniques with different characteristics can give significant differences in the information gain. A balanced distribution that consider grouping and retains memory

about allocation on previous releases seems to be better across the board in terms of keeping the overhead under control while retaining valuable information.

In the study we have introduced the issues of retaining the information from field due to the limited number of profiling probes that can be inserted. Another challenge is in knowing when we can be sure that the information we have gathered is an accurate representation of the field usage. As future work, we want to propose and evaluate metrics that can allow us to measure how confident we are with the aggregated information.

8 Acknowledgments

This work was supported in part by NSF CAREER Award 0347518 and an NSF EPSCoR First Award. We are thankful to Zhimin Wang for leading the preparation of MyIE, and the study's participants.

References

[1] M. Arnold and B. Ryder. A framework for reducing the cost of instrumented code. In *Conf. on Prog. Lang. Design and Impl.*, pages 168–179, 2001.

[2] T. Ball and J. Laurus. Optimally profiling and tracing programs. In *Symp. on Principles of Prog. Lang.*, pages 59–70, Aug. 1992.

[3] M. Dmitriev. Profiling java applications using code hotswapping and dynamic call graph revelation. In *Int. Workshop on Soft. and Performance*, pages 139–150, 2004.

[4] S. Elbaum and M. Diep. Profiling deployed software: Assessing strategies and testing opportunities. *IEEE Trans. Soft. Eng.*, 31(4):312–327, 2005.

[5] S. Elbaum and M. Hardojo. An empirical study of profiling strategies for released software and their impact on testing activities. In *Int. Symp. on Soft. Testing and Analysis*, pages 65 – 75. ACM, June 2004.

[6] A. Glenn, T. Ball, and J. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *ACM SIGPLAN Notices*, 32(5):85–96, 1997.

[7] S. Graham and M. McKusick. Gprof: a call graph execution profiler. *ACM SIGPLAN SCC*, 17(6):120–126, June 1982.

[8] M. Harrold, R. Lipton, and A. Orso. Gamma: Continuous evolution of software after deployment. www.cc.gatech.edu/aristotle/Research/Projects/gamma.html.

[9] D. Hilbert and D. Redmiles. An approach to large-scale collection of application usage data over the Internet. In *Int. Conf. on Soft. Eng.*, pages 136–145, 1998.

[10] S. Krishnamurthy and A. Mathur. On predicting reliability of modules using code coverage. In *Conf. of Centre for Advanced Studies on Collaborative Research*, page 22, 1996.

[11] B. Liblit, A. Aiken, Z. Zheng, and M. Jordan. Bug isolation via remote program sampling. In *Conf. on Prog. Lang. Design and Impl.*, pages 141–154. ACM, June 2003.

[12] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. Jordan. Scalable statistical bug isolation. In *Conf. Prog. Lang. Design and Impl.*, pages 15–26, June 2005.

[13] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. Schmidt, and B. Natarajan. Skoll: Distributed continuous quality assurance. In *Int. Conf. on Soft. Eng.*, pages 449–458, May 2004.

[14] A. Orso, T. Apiwattanapong, and M.J.Harrold. Leveraging field data for impact analysis and regression testing. In *Foundations of Soft. Eng.*, pages 128–137. ACM, September 2003.

[15] A. Orso, D. Liang, M. Harrold, and R. Lipton. Gamma system: Continuous evolution of software after deployment. In *Int. Symp. on Soft. Testing and Analysis*, pages 65–69, 2002.

[16] C. Pavlopoulou and M. Young. Residual Test Coverage Monitoring. In *Int. Conf. of Soft. Eng.*, pages 277–284, May 1999.

[17] S. Reiss and M. Renieris. Encoding Program Executions. In *Int. Conf. of Soft. Engineering*, pages 221–230, May 2001.

[18] D. Richardson, L. Clarke, L. Osterweil, and M. Young. Perpetual testing. www.ics.uci.edu/ djr/edcs/PerpTest.html.

[19] M. Tikir and J. Hollingsworth. Efficient instrumentation for code coverage testing. In *Int. Symp. on Soft. Testing and Analysis*, pages 86–96, 2002.

[20] E. J. Weyuker. Using failure cost information for testing and reliability assessment. *ACM Trans. Soft. Eng. Methodol.*, 5(2):87–98, 1996.