University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers | Computer Science and Engineering, Department of

1-1-2004

Infrastructure Support for Controlled Experimentation with Software Testing and Regression Testing Techniques

Hyunsook Do
University of Nebraska-Lincoln, dohy@cse.unl.edu

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

Gregg Rothermel
University of Nebraska-Lincoln, grother@cse.unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Do, Hyunsook; Elbaum, Sebastian; and Rothermel, Gregg, "Infrastructure Support for Controlled Experimentation with Software Testing and Regression Testing Techniques" (2004). *CSE Conference and Workshop Papers*. Paper 129.
http://digitalcommons.unl.edu/cseconfwork/129

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04.
Digital Object Identifier: 10.1109/ISESE.2004.1334894
Publication Year: 2004 , Page(s): 60 - 70

Infrastructure Support for Controlled Experimentation with Software Testing and Regression Testing Techniques

Hyunsook Do, Sebastian Elbaum, Gregg Rothermel
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE
{dohy, elbaum, rothermel}@cse.unl.edu

Abstract

Where the creation, understanding, and assessment of software testing and regression testing techniques are concerned, controlled experimentation is an indispensable research methodology. Obtaining the infrastructure necessary to support such experimentation, however, is difficult and expensive. As a result, progress in experimentation with testing techniques has been slow, and empirical data on the costs and effectiveness of techniques remains relatively scarce. To help address this problem, we have been designing and constructing infrastructure to support controlled experimentation with testing and regression testing techniques. This paper reports on the challenges faced by researchers experimenting with testing techniques, including those that inform the design of our infrastructure. The paper then describes the infrastructure that we are creating in response to these challenges, and that we are now making available to other researchers, and discusses the impact that this infrastructure has and can be expected to have.

1 Introduction

Testing is an important engineering activity responsible for a significant portion of the costs of developing and maintaining software [3, 21]. It is important for researchers and practitioners to understand the tradeoffs and factors that influence testing techniques. Some understanding can be obtained by using analytical frameworks, subsumption relationships, or axioms [29, 32, 38]. In general, however, testing techniques are heuristics and their performance varies with different scenarios; thus, they must be studied empirically.

The initial, development testing of a software system is important; however, software that succeeds evolves, and over time, more effort is spent re-validating a software system's subsequent releases than is spent performing initial, development testing. This re-validation activity is known as *regression testing*, and includes tasks such as re-executing existing tests [26], selecting subsets of test suites [6, 33], prioritizing test cases to facilitate earlier detection of faults [11, 34, 40], augmenting test suites to cover system enhancements [5, 31], and maintaining test suites [15, 16, 23]. These activities, too, involve many cost-benefits tradeoffs and depend on many factors, and must be studied empirically.

Many testing and regression testing techniques involve activities performed by engineers, and ultimately we need to study the use of such techniques by those engineers. Much can be learned about testing techniques, however, through studies that focus directly on those techniques themselves. For example, we can measure and compare the fault-revealing capabilities of test suites created by various testing methodologies [13, 17], the cost of executing the test suites created by different methodologies [4], or the influence of choices in test suite design on testing cost-effectiveness [30]. Such studies provide important information on tradeoffs among techniques, and they can also help us understand the hypotheses that should be tested, and the controls that are needed, in subsequent studies of humans, which are likely to be more expensive.

Empirical studies of testing techniques, like studies of engineers who perform testing, involve many challenges and cost-benefits tradeoffs, and this has constrained progress in this area. In general, two classes of empirical studies can be considered: case studies and controlled experiments. Controlled experiments focus on rigorous control of variables in an attempt to preserve internal validity and support conclusions about causality, but the limitations that result from exerting control can limit the ability to generalize results [36]. Case studies sacrifice control, and thus, internal validity, but can include a richer context [43]. Each of these classes of studies can provide insights into software testing techniques, and together they are complementary; in this paper, however, our focus is controlled experimentation.

Controlled experimentation with testing techniques depends on numerous software-related artifacts, including software systems, test suites, and fault data; for re-

Table 1. Research articles involving testing and empirical studies in four major venues, 1994-2003.

YEAR	TSE			TOSEM			ISSTA			ICSE		
	Total	Testing	Empir.	Total	Testing	Empir.	Total	Testing	Empir.	Total	Testing	Empir.
2003	74	8	7	7	0	0	-	-	-	75	7	3
2002	74	8	4	14	2	0	23	11	4	57	4	4
2001	55	6	5	11	4	3	-	-	-	61	5	3
2000	62	5	2	14	0	0	21	10	2	67	5	2
1999	46	1	0	12	1	0	-	-	-	58	4	2
1998	73	4	3	12	1	1	16	9	1	39	2	2
1997	50	5	4	12	1	1	-	-	-	52	4	2
1996	59	8	2	13	5	2	29	13	1	53	5	3
1995	70	4	1	10	0	0	-	-	-	31	3	1
1994	68	7	1	12	3	1	17	10	1	30	5	2
Total	631	56	29	117	17	8	106	53	9	523	44	24

gression testing experimentation, multiple versions of software systems are also required. Obtaining such artifacts and organizing them in a manner that supports controlled experimentation is a difficult task. These difficulties are illustrated by the survey of recent articles reporting experimental results on testing techniques presented in Section 2 of this paper. Section 3 further discusses these difficulties in terms of the challenges faced by researchers wishing to perform controlled experimentation, which include the needs to generalize results, ensure replicability, aggregate findings, isolate factors, and amortize the costs of experimentation.

To help address these challenges, we have been designing and constructing infrastructure to support controlled experimentation with software testing and regression testing techniques.[1] Section 4 of this paper presents this infrastructure, describing its organization and primary components, and our plans for making it available and augmenting it. Section 5 concludes by reporting on the impact this infrastructure has had, and can be expected to have, on further controlled experimentation.

2 A Survey of Studies of Testing

To provide an initial view on the state of the art in empirical studies of software testing, we surveyed recent research articles following approaches used by Tichy et al. [35] and Zelkowitz et al. [44]. We selected two journals and two conferences recognized as pre-eminent in software engineering research and known for including papers on testing: IEEE Transactions on Software Engineering (TSE), ACM Transactions on Software Engineering and Methodology (TOSEM), the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), and the ACM/IEEE International Conference on Software Engineering (ICSE). We considered all issues and proceedings from these venues, over the period 1994 to 2003.

Table 1 summarizes the results of our survey with respect to numbers of research articles appearing in each venue per year.[2] The table contains three columns of data per venue: Total (the total number of articles published in that year), Testing (the number of articles about software testing published in that year), and Empir. (the number of articles about software testing that contained some type of empirical study). Note that ISSTA proceedings appear bi-annually. As the table shows, 12.3% (170) of the articles in the venues considered concern software testing, a relatively large percentage attesting to the importance of the topic. (This includes papers from ISSTA, which would be expected to have a large testing focus, but even excluding ISSTA, 9.3% of the articles in the other venues, whose focus is software engineering generally, concern testing.) Of the testing-related articles, however, only 41% (70) report empirical studies.[3]

We next analyzed the 70 articles on testing that reported empirical studies, considering the following categories, which represent factors important to controlled experimentation on testing and regression testing:

- The type of empirical study performed.
- Number of programs used as sources of data.
- Number of versions used as sources of data.
- Whether test suites were utilized.
- Whether fault data was utilized.
- Whether the study involved artifacts provided by or made available to other researchers.

[1] This work shares many similarities with the activities being promoted by the International Software Engineering Research Network (ISERN). ISERN, too, seeks to promote experimentation, in part through the sharing of resources; however, ISERN has not to date focused on controlled experimentation with software testing, or produced infrastructure appropriate to that focus.

[2] Due to space limitations we cannot provide full details of the survey here; however, these are available in [8].

[3] The authors of this paper have been responsible for several of the papers considered in this survey; however, if those papers are eliminated from consideration, the foregoing percentages become 11.2%, 8.1%, and 37%, respectively, and continue to support our conclusions.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE

Table 2. Further classification of published empirical studies

Publication	Empirical Papers	Example	Case Study	Controlled Experiment	Multiple Programs	Multiple Versions	Tests	Faults	Shared Artifacts
TSE (1999-2003)	18	0	9	9	15	6	16	7	5
TSE (1994-1998)	11	2	8	1	6	2	8	2	1
TSE (1994-2003)	29	2	17	10	21	8	24	9	6
TOSEM (1999-2003)	3	0	0	3	3	3	3	3	2
TOSEM (1994-1998)	5	1	1	3	4	2	5	3	1
TOSEM (1994-2003)	8	1	1	6	7	5	8	6	3
ISSTA (1999-2003)	6	0	4	2	5	2	6	1	1
ISSTA (1994-1998)	3	0	2	1	3	1	3	1	0
ISSTA (1994-2003)	9	0	6	3	8	3	9	2	1
ICSE (1999-2003)	14	1	6	7	9	6	14	8	6
ICSE (1994-1998)	10	0	7	3	6	7	10	5	2
ICSE (1994-2003)	24	1	13	10	15	13	24	13	8
Total (1999-2003)	41	1	19	21	32	17	39	19	14
Total (1994-1998)	29	3	18	8	19	12	26	11	4
Total (1994-2003)	70	4	37	29	51	29	65	30	18

Determining the type of empirical study performed required a degree of subjective judgement, due to vague descriptions by authors and the absence of clear quantitative measures for differentiating study types. However, previous work [1, 20, 39, 44] provides guidelines for classifying types of studies, and we used these to determine whether studies should be classified as controlled experiments or case studies (for details, see [8]). On close analysis, some observational work described by authors as "empirical studies" should not have been described as such, being essentially just descriptions of the application of a technique on a single extended example; following [44] we classified these as "examples".

Table 2 summarizes the results of our analysis. The table reports the data for each venue in terms of three time periods: 1994-1998, 1999-2003, and 1994-2003. Over the ten year period, 41% of the studies presented were controlled experiments and 53% were case studies. Separation of this data into time periods suggests that trends are changing: 28% of the studies in the first five years (1994-1998) were controlled experiments, compared to 51% in the second five years (1999-2003). This trend occurs across all venues other than ISSTA, and it is particularly strong for TSE (9% vs. 50%).[4]

The table also shows that only 26% of the studies involved artifact sharing. This figure exhibits an increasing trend from 14% in the early time period to 33% in the later period. Finally, the table shows that of the 70 studies, 27% utilize data from only one program (although this is not necessarily problematic for case studies). Also, only 44% of the studies utilize multiple versions and only 43% utilize fault data.[5]

Further investigation of this data is revealing. Of the 18 papers in which artifacts were shared among researchers, 17 use one or both of a set of programs known as the "Siemens programs", or a somewhat larger program known as the "space" program. (Four of these 17 papers also use one or two other large programs, but these programs have not to date been made available to other researchers as shared artifacts.) The Siemens programs, originally introduced to the research community by Hutchins et al. [17], and subsequently augmented, organized, and made available as sharable infrastructure by one of the authors of this paper, consist of seven C programs of no more than 1000 lines of code, 132 seeded faults for those programs, and several sets of test suites satisfying various test adequacy criteria. The space program, appearing initially in papers by other researchers [37, 41] and also processed and made available as sharable infrastructure by one of the authors of this paper, is a single application of nearly 10,000 lines of code, provided with various test suites, and 35 actual faults. In the cases in which multiple "versions" of software systems are used in studies involving these programs, these versions differ only in terms of faults, rather than in terms of a set of changes, of which some have caused faults; ignoring these cases, only four cases exist in which actual, realistic multiple versions of programs are utilized.

[4] The results of this analysis, too, remain stable when papers involving the authors of this paper are excluded; in that case, over the ten year period, 28% of the studies were controlled experiments and 64% were case studies; 17% of the studies in the first five-year period were controlled experiments compared to 38% in the second five-year period; and the increasing trend is visible across all venues, remaining strongest for TSE (0% to 40%).

[5] Excluding papers by the authors of this paper, 11% of the studies involve sharing, the increasing trend is from 4% to 17%, and the last three percentages become 32%, 44%, and 28%, respectively, continuing to support our conclusions.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE

3 Challenges for Experimentation

Researchers attempting to conduct controlled experiments examining the application of testing techniques to artifacts face several challenges. The survey of the literature just summarized provides evidence of the effects of these challenges. The survey also suggests, however, that researchers are becoming increasingly willing to conduct controlled experiments, and are increasing the extent to which they utilize shared artifacts.

These tendencies are related: utilizing shared artifacts is likely to facilitate controlled experimentation. The Siemens and space programs, in spite of their limitations, have facilitated a number of controlled experiments that might not otherwise have been possible. This argues for the utility of making additional infrastructure available to other researchers, as is our goal.

Before proceeding further, however, it is worthwhile to identify the challenges faced by researchers performing experimentation on testing techniques in the presence of limited infrastructure. Identifying such challenges provides insights into the limited progress in this area that goes beyond the availability of artifacts. Furthermore, identifying these challenges helps us define the infrastructure requirements for such experiments, and to shape the design of an experiment infrastructure.

1: Supporting replicability across experiments.

A scientific finding is not trusted unless it can be independently replicated. When performing a replication, researchers duplicate the experimental design of an experiment on a different sample to increase the confidence in the findings [39] or on an extended hypothesis to evaluate additional variables [2]. Supporting replicability for controlled experiments requires establishment of control on experimental factors and context; this is increasingly difficult to achieve as the units of analysis and context become more complex. When performing controlled experimentation with software testing techniques, several replicability challenges exist.

First, artifacts utilized by researchers are rarely homogeneous. For example, programs may belong to different domains and have different complexities and sizes, versions may exhibit different rates of evolution, processes employed to create programs and versions may vary, and faults available for the study of fault detection may vary in type and magnitude.

Second, artifacts are provided in widely varying levels of detail. For example, programs freely available through the open source initiative are often missing formal documentation or rigorous test suites. On the other hand, confidentiality agreements often constrain the industry data that can be utilized in published experiments, especially data related to faults and failures.

Third, experiment design and process details are often not standardized or reported in sufficient detail. For example, different types of oracles may be used to evaluate technique effectiveness, different, non-comparable tools may be used to capture coverage data, and when fault seeding is employed it may not be clear who performed the activity and what process they followed.

2: Supporting aggregation of findings.

Individual experiments may produce interesting findings, but can claim only limited validity under different contexts. In contrast, a family of experiments following a similar operational framework can enable the aggregation of findings, leading to generalization of results and further theory development.

Opportunities for aggregation are highly correlated with the replicability of an experiment (Challenge 1); that is, a highly replicable experiment is likely to provide detail sufficient to determine whether results across experiments can be aggregated. (This reveals just one instance in which the relationship between challenges is not orthogonal, and in which providing support to address one challenge may impact others.)

Still, even high levels of replicability cannot guarantee correct aggregation of findings unless there is a systematic capture of experimental context [28]. Such systematic capture typically does not occur in the domain of testing experimentation. For example, versions utilized in experiments to evaluate regression testing techniques may represent minor internal versions or major external releases; these two scenarios clearly involve very distinct levels of validation. Although capturing complete context is often infeasible, the challenge is to provide enough support so that the evidence obtained across experiments can be leveraged.

3: Reducing the cost of controlled experiments.

Controlled experimentation is expensive, and there are several strategies available for reducing this expense. For example, experiment design and sampling processes can reduce the number of participants required for a study of engineer behavior, thereby reducing data collection costs. Even with such reductions, obtaining and preparing participants for experimentation is costly, and that cost varies with the domain of study, the hypotheses being evaluated, and the applicability of multiple and repeated treatments on the same participants.

Controlled experimentation in which testing techniques are applied to artifacts does not require human

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

participants, it requires objects such as programs, versions, tests, and faults. This is advantageous because artifacts are more likely to be reusable across experiments, and multiple treatments can be validly applied across all artifacts at no cost to validity. Still, artifact reuse is often jeopardized due to several factors.

First, artifact organization is not standardized. For example, different programs may be presented in different directory structures, with different build processes, fault information, and naming conventions.

Second, artifacts are incomplete. For example, open source systems seldom provide comprehensive test suites, and industrial systems are often "sanitized" to remove information on faults and their corrections.

Third, artifacts require manual handling. For example, build processes may require software engineers to configure various files, and test suites may require a tester to control execution and audit results.

4: Obtaining sample representativeness.

Sampling is the process of selecting a subset of a population with the intent of making statements about the entire population. The degree of representativeness of the sample is important because it directly impacts the applicability of the conclusions to the rest of the population. However, representativeness needs to be balanced with considerations for the homogeneity of the sampled artifacts in order to facilitate replication as well. Within the software testing domain, we have found two common problems for sample representativeness.

First, sample size is limited. Since preparing an artifact is expensive, experiments often use small numbers of programs, versions, and faults. Further, researchers trying to reduce costs (Challenge 3) do not prepare artifacts for repeated experimentation (e.g., test suite execution is not automated). Lack of preparation for reuse limits the growth of the sample size even when the same researchers perform similar studies.

Second, samples are biased. Even when a large number of programs are collected they usually belong to a set of similar programs. For example, as described in Section 2, many researchers have employed the Siemens programs in controlled experiments with testing. This set of objects includes seven programs with faults, versions, processing scripts, and automated test suites. The Siemens programs, however, each involve fewer than 1000 lines of code. Other sources of sample bias include the types of faults seeded or considered, processes used for test suite creation, and code changes considered.

5: Isolating the effects of individual factors.

Understanding causality relationships between factors is at the core of experimentation. Blocking and manipulating the effects of a factor increases the power of an experiment to explain causality. Within the testing domain, we have identified two major problems for controlling and isolating individual effects.

First, artifacts may not offer the same opportunities for manipulation. For example, programs with multiple faults offer opportunities for analyzing faults individually or in groups, which can affect the performance of testing techniques as it introduces masking effects. Another example involves whether or not automated and partitionable test suites are available; these may offer opportunities for isolating test case size as a factor.

Second, artifacts may make it difficult to decouple factors. For example, it is often not clear what program changes in a given version occurred in response to a fault, an enhancement, or both. Furthermore, it is not clear at what point the fault was introduced in the first place. As a result, the assessment of testing techniques designed to increase the detection of regression faults may be biased.

4 Infrastructure

We have described what we believe are the primary challenges faced by researchers wishing to perform controlled experimentation with testing techniques, and that have limited the progress in this area. Some of these challenges involve issues for experiment design, and guidelines such as those provided by Kitchenham et al. [19] address those concerns. Other challenges relate to the process of conducting families of experiments with which to incrementally build knowledge, and lessons such as those presented by Basili et al. [2] could be valuable in addressing these. All of these challenges can be traced, at least partly (and some primarily), however, to issues involving infrastructure.

To address these challenges, we have been designing and constructing infrastructure to support controlled experimentation with software testing and regression testing techniques. Our infrastructure includes a set of artifacts (programs, versions, tests, faults, and scripts) that enable researchers to perform controlled experimentation and replications. Also included is documentation on the processes used to select, organize, and further set up artifacts, and supporting tools that help with these processes. Together with our plans for sharing and extending the infrastructure, these objects, documents, tools, and processes help address the challenges described in the preceding section as summarized in Table 3.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE

Table 3. Challenges and Infrastructure.

Challenges	Infrastructure attributes				
	Artifact			Docs,	Share,
	Sel.	Org.	Setup	Tools	Extend
Support Replicability	X	X	X	X	X
Support Aggregation		X	X	X	X
Reduce Cost	X	X	X	X	X
Representativeness	X				X
Isolate Effects		X	X		

4.1 Object selection, organization, and setup

Our infrastructure provides guidelines for object selection, organization, and setup processes. The selection and setup guidelines assist in the construction of a sample of complete artifacts. The organization guidelines provide a consistent context for all artifacts, facilitating the development of generic experiment tools, and reducing the experimentation overhead for researchers.

4.1.1 Object selection

Object selection guidelines direct persons assembling infrastructure in the task of selecting suitable objects, and are provided through a set of on-line instructions that include artifact selection requirements. Thus far, we have specified two levels of required qualities for objects: 1st-tier required-qualities (minimum lines of code required, source freely available, five or more versions available) and 2nd-tier required-qualities (runs on platforms we utilize, can be built from source, allows automation of test input application and output validation). When assembling objects, we first identify objects that meet first-tier requirements, which can be determined relatively easily, and then we prioritize these, and for each, investigate second-tier requirements.

Part of the object selection task involves ensuring that programs and their versions can be built and executed automatically. Because experimentation requires the ability to repeatedly execute and validate large numbers of tests, automatic execution and validation must be possible for candidate programs. Thus, our infrastructure currently excludes programs that require graphical input/output that cannot easily be automatically executed or validated. We also require programs that execute, or through edits can be made to execute, deterministically; this too is a requirement for automated validation, and implies that programs involving concurrency and heavy thread use might not be directly suitable.

Our infrastructure now consists of 17 C and two Java programs, as shown in Table 4. The first eight programs listed are the Siemens and space programs, which constituted our first set of experiment objects; the remaining programs include nine larger C programs and two Java programs (nanoxml and siena), selected via the foregoing process. The other columns are as follows:

- The "Size" column presents the total number of lines of code, including comments, present in each program, and illustrates our attempts to incorporate progressively larger programs.
- The "No. of Versions" column lists how many versions each program has. The Siemens and space programs are available only in single versions (with multiple faults), a serious limitation, although the availability of multiple faults has been leveraged, in experiments, to create various alternative versions containing one or more faults. Our more recently collected objects, however, are available in multiple, sequential releases (corresponding to actual field releases of the systems.)
- The "No. of Tests" column lists the number of tests available for the program (for multi-version programs, this is the number available for the final version). Each program has one or more types of tests and one or more types of test suites (described below). The two Java programs are also provided with test drivers that invoke classes under test.
- The "No. of Faults" column indicates the total number of faults available for each of the programs; for multi-version programs we list the sum of faults available across all versions.
- The "Release Status" column indicates the current release status of each object as one of "released", "ready", or "near release". The Siemens and space programs, as detailed above, have been provided to and used by many other researchers, so we categorize them as released. Bash, emp-server, pine, vim, and siena are undergoing final formatting and testing and thus are listed as "near release". The rest of the programs listed are now available in our infrastructure repository.

Our object selection process helps provide consistency in the preparation of artifacts, supporting replicability. The same process also reduces costs by discarding earlier the artifacts that are not likely to meet the experimental requirements. Last, the selection mechanism lets us adjust our sampling process to facilitate the collection of a representative set of artifacts.

4.1.2 Object organization

We organize objects and associated artifacts into a directory structure that supports experimentation. Each object we create has its own "object" directory, as shown in Figure 1. An object directory is organized into specific subdirectories (which in turn may contain subdirectories), as follows.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE

IEEE COMPUTER SOCIETY

Table 4. Objects in our Infrastructure

Subjects	Size (LOC)	No. of Versions	No. of Tests	No. of Faults	Release Status
tcas	173	1	1608	41	released
schedule2	374	1	2710	10	released
schedule	412	1	2650	9	released
replace	564	1	5542	32	released
tot_info	565	1	1052	23	released
print_tokens2	570	1	4115	10	released
print_tokens	726	1	4130	7	released
space	9564	1	13585	35	released
gzip	6582	6	217	15	ready
sed	11148	5	1293	40	ready
flex	15297	6	567	81	ready
grep	15633	6	809	75	ready
make	27879	5	1043	17	ready
bash	48171	10	1168	69	near release
emp-server	64396	10	1985	90	near release
pine	156037	4	288	24	near release
vim	224751	9	975	7	near release
nanoxml	7646	6	217	33	ready
siena	6035	8	567	3	near release

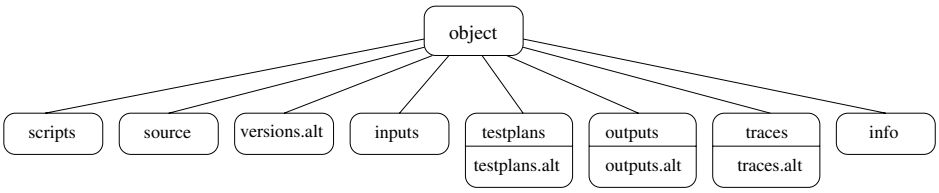

Figure 1. Object directory structure (top level)

- The scripts directory is the "staging platform" from which experiments are run; it may also contain saved scripts that perform object-related tasks.
- The source directory is a working directory in which, during experiments, the program version being worked with is temporarily placed.
- The versions.alt directory contains various variants of the source for building program versions; these include (among others) original source code for each version, and fault-seeded variants of that code. Each variant is itself organized as a subdirectory; that subdirectory contains subdirectories v0, v1, . . ., vk corresponding to different versions.
- The inputs directory contains files containing inputs, or directories of inputs used in various tests.
- The testplans.alt directory contains subdirectories v0, v1, . . ., vk, each of which contains testing information for a system version; this information typically includes a "universe" file containing a pool of tests, and various test suites drawn from that pool.
- The traces.alt directory contains subdirectories v0, v1, . . ., vk, each holding trace information for a version of the system, in the form of individual test traces or summaries of coverage information.
- The outputs.alt directory permanently stores the outputs of test runs, especially useful when experimenting with regression testing where outputs are compared against previous outputs.
- The testplans, outputs, and traces directories serve as "staging platforms" during specific experiments. Data from a specific "testplans.alt" subdirectory is placed into the testplans directory prior to experimentation; data from outputs and traces directories is placed into subdirectories in their corresponding ".alt" directories following experimentation.
- The info directory contains additional information about the program, especially information gathered by analysis tools and requiring saving for experiments, such as fault-matrix information (which describe the faults that various test cases reveal).

Our object organization supports consistent experimentation conditions and environments, allowing us to write generic tools for experimentation that know where to find things, and that function across all of our ob-

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 **IEEE**

jects. This in turn helps reduce the costs of executing and replicating controlled experiments, and aggregating results across experiments. The use of this structure can potentially limit external validity by restricting the types of objects that can be accommodated, and the transformation of objects to fit the infrastructure can create some internal validity threats. However, the continued use of this structure and the generic tools it supports ultimately reduces a large class of potential threats to internal validity arising from errors in automation, by facilitating cross-checks on tools, and leveraging previous tool validation efforts. The structure also accommodates objects with various types and classes of artifacts, such as multiple versions, fault types, and test suites, enabling us to control for and isolate individual effects in conducting experimentation.

4.1.3 Object setup

Test suites

Systems we have selected for our repository have only occasionally arrived equipped with anything more than rudimentary test suites. When suites are provided, we incorporate them into our infrastructure because they are useful for case studies. For controlled experiments, however, we typically prefer to have test suites created by uniform processes. Such test suites can also be created in ways that render them partitionable, facilitating studies that isolate factors such as test size, as mentioned in Section 3 (Challenge 5).

To construct test suites that represent those that might be constructed in practice for particular programs, we have relied primarily on two general processes, following the approach used by Hutchins et al. [17] in their initial construction of the Siemens programs.

The first process involves specification-based testing using the category-partition method, based on a test specification language, TSL, described in [27]. A TSL specification is written for an initial version of an object, based on its documentation, by a person who has become familiar with that documentation and the functionality of the object. Subsequent versions of the object inherit this specification, or most of it, and may need additional tests to exercise new functionality, which can be encoded in an additional specification added to that version, or in a refined TSL specification. TSL specifications are processed by a tool, provided with our infrastructure, into test frames, which describe the requirements for specific test cases. Each test case is created and encoded in proper places within the object directory.

The second test process we have used involves coverage-based testing, in which we instrument the object program, measure the code coverage achieved by specification-based tests, and then create tests that exercise code not covered by those tests.

Employing these processes using multiple testers helps reduce threats to validity involving specific tests that are created. Creating larger pools of test cases in this fashion and sampling them to obtain various test suites, such as test suites that achieve branch coverage or test suites of specific sizes, provides further assistance with generalization. We store such suites with the objects along with their pools of tests.

At present, not all of our objects possess equivalent types of tests and test suites, but one goal in extending our infrastructure is to ensure that specific types of tests and test suites are available across all objects, to aid with the aggregation of findings. A further goal, of course, is to provide multiple instances and types of tests suites per object, a goal that has been achieved for the Siemens and space programs allowing the completion of several comparative studies. Meeting this goal will be further facilitated through sharing of the infrastructure, and collaboration with other researchers.

Faults

For studies of fault detection, we provide processes for two cases: the case in which naturally occurring faults can be identified, and the case in which faults must be seeded. Either possibility presents advantages and disadvantages: naturally occurring faults are costly to locate and typically cannot be found in large numbers, but they represent actual events. Seeded faults are costly to place, but can be provided in larger numbers, allowing more data to be gathered than would otherwise be possible, but with less external validity.

To help with the fault seeding process, and increase the potential external validity of results obtained on seeded faults, we insert faults by following fault localization guidelines, which provide direction on places that are likely to contain faults. We also provide fault classifications based on published fault data, so that faults will correspond, to the extent possible, to faults found in practice. To further reduce the potential for bias, fault seeding is performed independently of experimentation, by multiple persons with at least 3 years of programming experience, and without knowledge of any specific experiment plans.

Another motivation for seeding faults occurs when experimentation concerned with regression testing is the goal. For regression testing, we wish to investigate errors caused by code change (regression faults). With the assistance of a differencing tool, fault seeders locate code changes, and place faults within those.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

4.2 Documentation and supporting tools

Documentation and guidelines supplied with our infrastructure provide detailed procedures for object selection and organization, test generation, fault localization, automatic tool usage, and current object descriptions (our descriptions in this paper have summarized the far more extensive information available on our infrastructure site.) As suggested in Section 3, such guidelines support sharing (and thus cost reduction), as well as facilitating replication and aggregation across experiments. Documentation and guidelines are thus as important as objects and associated artifacts.

Depending on the research questions being investigated, testing experiment designs and processes can be very complex and require multiple executions, so automation is important. Our infrastructure provides a set of automated testing tools that build scripts executing tests automatically, gather traces for tests, generate test frames based on TSL specifications, and generate fault matrices (tables relating faults to the tests that expose them) for objects. These tools make experiments simpler to execute, and reduce the possibility of human errors, such as typing errors, supporting replicability as well. The automated testing tools function across all objects, given the uniform directory structure for objects; thus, we can reuse these tools on new objects as they are completed, reducing the costs of preparing such objects.

4.3 Sharing and extending the infrastructure

Our standard object organization and tool support help our infrastructure be extensible; objects that meet our requirements can be assembled using the required formats and tools. This is still an expensive process, but in the long run such extension will help us achieve sample representativeness, and help with problems in replicability and aggregation as discussed in Section 3.

In our initial infrastructure construction, we have focused on gathering objects and artifacts for regression testing study, and on facilitating this with faults, multiple versions and tests. Such materials can also be used, however, for experimentation with testing techniques generally, and with other program analysis techniques. (Section 5 discusses cases in which this is already occurring.) Still, we intend that our infrastructure be extended through addition of objects with other types of associated artifacts, such as may be useful for different types of controlled experiments. For example, one of our Java objects, nanoxml, is provided with UML statechart diagrams, and this would facilitate experimentation with UML-based testing techniques.

Extending our infrastructure can be accomplished in two ways: by our research group, and by collaboration with other research groups. To date we have proceeded primarily through the first approach, but the second has many benefits. First, it is cost effective, mutually leveraging the efforts of others. Second, through this approach we can achieve greater diversity among objects and associated artifacts, which will be important in helping to increase sample size and achieve representativeness. Third, sharing implies more researchers inspecting the artifacts setup, tools, and documentation reducing threats to internal validity. Ultimately, collaboration in constructing and sharing infrastructure can help us contribute to the growth in the ability of researchers to perform controlled experimentation on testing in general.

As mentioned earlier, we have been making our Siemens and space infrastructure available, on request, for several years. We have recently created web pages that provide this infrastructure, together with all more recently created infrastructure described in this article, and all of the programs listed in Table 4 with the exception of those listed as "near release". We have made this web page available to researchers at several institutions for initial Beta testing, and we will make it available to any other researchers who request the address from us by email, provided they are willing to report to us any experiences that will help us to improve the infrastructure. Following this Beta shakedown, and correction of problems found during this period, we intend to make our web site openly available.

5 Conclusion

We have presented our infrastructure for supporting controlled experimentation with testing techniques, and we have described several of the ways in which it can potentially help address many of the challenges faced by researchers wishing to conduct controlled experiments on testing. We close this article by providing additional discussion of the impact, both demonstrated and potential, of this infrastructure.

First, we remark on the impact of our infrastructure to date. Many of the infrastructure objects described in the previous section are only now being made available to other researchers. The Siemens and space programs, however, in the format extended and organized by ourselves, have been available to other researchers since 1999, and have seen widespread use. In addition to our own papers describing experimentation using these artifacts (over twenty such papers have appeared, see http://www.cs.orst.edu/~grother) we have identified seven other papers not involving creators of this initial infrastructure that describe controlled experiments

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

involving testing techniques using the Siemens and/or space programs [7, 14, 18, 24, 25, 42]. The artifacts have also been used in [12] for a study of dynamic invariant detection (attesting to the feasibility of using the infrastructure in areas beyond those limited to testing).

In our review of the literature, we have found no similar usage of other artifacts *for controlled experimentation in software testing*; the willingness of other researchers to use the Siemens and space artifacts thus attests to the potential for infrastructure, once made available, to have an impact on research. This same willingness, however, also illustrates the need for improvements to infrastructure, given that the Siemens and space artifacts present only a small sample of the population of programs, versions, tests, and faults. It seems reasonable, then, to expect our extended infrastructure to be used for experimentation by others, and to help extend the validity of experimental results through widened scope. Indeed, we ourselves have been able to use several of the newer infrastructure objects that are about to be released in controlled experiments described in recent publications [9, 10, 22, 30], as well as in three publications currently under review.

In terms of impact, it is also worthwhile to discuss the costs involved in preparing infrastructure; it is these costs that we *save* when we re-use infrastructure. For example, the emp-server and bash objects required between 80 and 300 person-hours per version to prepare; two faculty and five graduate research assistants have been involved in this preparation. The flex, grep, make, sed and gzip programs involved two faculty, three graduate students, and five undergraduate students; these students worked 10-20 hours per week on these programs for between 20 and 30 weeks. These costs are not costs typically affordable by researchers; it is only by amortizing the costs over the potential controlled experiments that can follow that we render the costs acceptable.

Finally, there are several additional potential benefits to be realized through sharing of infrastructure in terms of challenges addressed; these translate into a reduction of threats to validity that would exist were the infrastructure not shared. By sharing our infrastructure with others, we can expect to receive feedback that will improve it. User feedback will allow us to improve the robustness of our tools and the clarity and completeness of our documentation, enhancing the opportunities for replication of experiments, aggregation of findings, and manipulation of individual factors.

We are in the process of setting up mechanisms for encouraging researchers who use our infrastructure to contribute additions to it in the form of new fault data, new test suites, and variants of programs and versions that function on other operational platforms. Ultimately, we expect the community of researchers to assemble additional artifacts using the formats and tools prescribed, and contribute them to the infrastructure, which will increase the range and representativeness of artifacts available to support experimentation.

Through this effort we hope to aid the entire testing research community in pursuing controlled experimentation with testing techniques, increasing our understanding of these techniques and the factors that affect them in ways that can be achieved only through such experimentation.

Acknowledgements

We thank the current and former members of the Galileo and MapsText research groups for their contributions to our experiment infrastructure. We also thank the Siemens researchers, and especially Tom Ostrand, for providing the Siemens programs and much of the initial impetus for this work, and Phyllis Frankl, Filip Vokolos, and Alberto Pasquini for providing most of the materials used to assemble the space program infrastructure. Finally, this work has been supported by NSF awards CCR-0080898 and CCR-0347518 to the University of Nebraska - Lincoln, and by NSF awards CCR-9703108, CCR-9707792, CCR-0080900, and CCR-0306023 to Oregon State University.

References

[1] V. Basili, R. Selby, E. Heinz, and D. Hutchens. Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, 12(7):733–743, July 1986.

[2] V. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, 1999.

[3] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.

[4] J. Bible, G. Rothermel, and D. Rosenblum. Coarse- and fine-grained safe regression test selection. *ACM Trans. Softw. Eng. Meth.*, 10(2):149–183, Apr. 2001.

[5] D. Binkley. Semantics guided regression test cost reduction. *IEEE Trans. Softw. Eng.*, 23(8):498–516, Aug. 1997.

[6] Y. Chen, D. Rosenblum, and K. Vo. TestTube: A system for selective regression testing. In *Int'l. Conf. Softw. Eng.*, pages 211–220, May 1994.

[7] A. Coen-Porisini, G. Denaro, C. Ghezzi, and P. Pezze. Using symbolic execution for verifying safety-critical systems. In *Proc. ESEC/FSE*, 2001.

[8] H. Do, G. Rothermel, and S. Elbaum. Infrastructure support for controlled experimentation with testing and regression testing techniques. Technical Report 04-60-01, Oregon State University, Jan. 2004.

[9] S. Elbaum, D. Gable, and G. Rothermel. The Impact of Software Evolution on Code Coverage. In *Int'l. Conf. Softw. Maint.*, pages 169–179, Nov. 2001.

[10] S. Elbaum, P. Kallakuri, A. Malishevsky, R. Rothermel, and S. Kanduri. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. *J. Softw. Testing, Verif., and Rel.*, 12(2), 2003.

[11] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Int'l. Conf. Softw. Eng.*, pages 329–338, May 2001.

[12] M. Ernst, A. Czeisler, W. Griswold, and D. Notkin. Quickly detecting relevant program invariants. In *Int'l. Conf. Softw. Eng.*, June 2000.

[13] P. G. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Trans. Softw. Eng.*, 19(8):774–787, Aug. 1993.

[14] M. Harder and M. Ernst. Improving test suites via operational abstraction. In *Int'l. Conf. Softw. Eng.*, May 2003.

[15] J. Hartmann and D. Robson. Techniques for selective revalidation. *IEEE Software*, 16(1):31–38, Jan. 1990.

[16] D. Hoffman and C. Brealey. Module test case generation. In *3rd Workshop on Softw. Testing, Analysis, and Verif.*, pages 97–102, Dec. 1989.

[17] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Int'l. Conf. Softw. Eng.*, pages 191–200, May 1994.

[18] J. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Int'l. Conf. Softw. Eng.*, May 2002.

[19] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.*, 28(8):721–734, Aug. 2002.

[20] B. Kitchenham, L. Pickard, and S. Pfleeger. Case studies for method and tool evaluation. *IEEE Software*, pages 52–62, July 1995.

[21] H. Leung and L. White. Insights into regression testing. In *Conf. Softw. Maint.*, pages 60–69, Oct. 1989.

[22] A. G. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. In *Int'l. Conf. Softw. Maint.*, pages 230–240, Oct. 2002.

[23] B. Marick. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley, Sept. 1999.

[24] M. Marre and A. Bertolino. Using spanning sets for coverage testing. *IEEE Trans. Softw. Eng.*, 29(11), Nov. 2003.

[25] V. Okun, P. Black, and Y. Yesha. Testing with model checkers: insuring fault visibility. *WSEAS Trans. Sys.*, 2(1):77–82, Jan. 2003.

[26] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. *Comm. ACM*, 41(5):81–86, May 1988.

[27] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Comm. ACM*, 31(6), June 1988.

[28] L. Pickard and B. Kitchenham. Combining empirical results in software engineering. *Inf. Softw. Tech.*, 40(14):811–821, Aug. 1998.

[29] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Trans. Softw. Eng.*, SE-11(4):367–375, Apr. 1985.

[30] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *Int'l. Conf. Softw. Eng.*, May 2002.

[31] G. Rothermel and M. Harrold. Selecting tests and identifying test coverage requirements for modified software. In *Int'l. Symp. Softw. Testing Anal.*, Aug. 1994.

[32] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.*, 22(8):529–551, Aug. 1996.

[33] G. Rothermel and M. Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Meth.*, 6(2):173–210, Apr. 1997.

[34] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Test case prioritization. *IEEE Trans. Softw. Eng.*, 27(10):929–948, Oct. 2001.

[35] W. Tichy, P. Lukowicz, E. Heinz, and L. Prechelt. Experimental evaluation in computer science: a quantitative study. *J. Sys. Softw.*, 28(1):9–18, Jan. 1995.

[36] W. Trochim. *The Research Methods Knowledge Base*. Atomic Dog, Cincinnati, OH, 2nd edition, 2000.

[37] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Int'l. Conf. Softw. Maint.*, pages 44–53, Nov. 1998.

[38] E. Weyuker. The evaluation of program-based software test data adequacy criteria. *Comm. ACM*, 31(6), June 1988.

[39] C. Wohlin, P. Runeson, M. Host, M. Ohlsoon, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[40] W. Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Eighth Intl. Symp. Softw. Rel. Engr.*, pages 230–238, Nov. 1997.

[41] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini. Test set size minimization and fault detection effectiveness: A case study in a space application. In *Comp. Softw. Appl. Conf.*, pages 522–528, Aug. 1997.

[42] T. Xie and D. Notkin. Macro and micro perspectives on strategic software quality assurance in resource constrained environments. In *Proc. EDSER-4*, May 2002.

[43] R. K. Yin. *Case Study Research : Design and Methods (Applied Social Research Methods, Vol. 5)*. Sage Publications, London, UK, 1994.

[44] M. Zelkowitz and D. Wallace. Experimental models for validating technology. *IEEE Computer*, pages 23–31, May 1998.

Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)
0-7695-2165-7/04 $20.00 © 2004 **IEEE**

IEEE COMPUTER SOCIETY