Comprehensive study of SIR: Leading SUT repository for software testing

J L Min*, N Rajabi and A Rahmani
Department of Computer Engineering and Informatic, Politeknik Negeri Bandung, West Bandung, Indonesia

*joelianmin@jtk.polban.ac.id

Abstract. Software testing is performed by executing test cases against the system under test (SUT). For researchers, it is slightly challenging to find real case SUT. As an alternative, researchers utilize the existing SUT from repositories as their research objects. Software-artifact Infrastructure Repository (SIR) provides some reasonably comprehensive SUTs that had been used by over 50% of software testing literature. Nevertheless, it is still rare among researchers to conduct profound studies of SIR’s SUTs. This paper aims to fill the gap by explaining SIR’s SUTs specification and how it’s organized. Each of the C-language SUT was exposed and recorded its features comprising the size of SUT, lines of code (LOC), number of test cases, as well as the number and types of faults. SIR’s SUTs divided into two parts namely “older” and “newer” objects which both provided with various versions of the test program as well as predefined test sets along with its specification. These results indicate that SIR’s SUTs is carefully prepared to conduct software testing research.

1. Introduction
Software Testing is the process to bring on the latent defects into identifiable ones [1]. Software testing used for assuring the quality of the software that being tested or known as System Under Test (SUT) by executing various test cases against it [2]. The vast amount of cost and elapsed time that testing phase expends has made a lot of research done to find the most effective test automation technique. In the research world, obtaining real case SUT as a research object is difficult and time consuming [3]. As an alternative, researchers rely a lot on SUTs that are already available in the repository since researchers are no longer needs to create various versions of test program, place the errors, design the test suites and related fault information. The stages mentioned are very challenging to do, especially in designing and making test cases [4].

One of the well-known repositories among researchers is the Software-artifact Infrastructure Repository (SIR). In Yoo and Harman [3], it was stated that 60% of the total literature on regression testing was utilizing SIR’s SUTs. Also Singh et al. [5] stated that more than half of the SUTs used by researchers were obtained from SIR and no major repository was used other than SIR.

However, only tiny amount of literature that reveals the profound SIR’s object organization and specification whereas it is important for researchers to know the objects organization and specification as initial provisions before using SIR’s SUT. The detailed explanation of SIR infrastructure was discussed by Do et al. [6], a research paper that presented the SIR repository. Unfortunately, some of
the SUTs specification are no longer the same as what they were after the SUT objects was downloaded from the SIR webpage [7].

This paper aims to fill that gap by providing updated details on the organization and characteristics of each C-language SUTs in SIR. This would be beneficial for researchers who are new on using SIR’s SUTs as well as a consideration for researchers in choosing which SUTs to be used as research objects. In addition, we are interested why SIR is widely used by researchers to proof that SIR’s object organization and specification is one of the reasons behind it. In order to achieve those goals, we formulated some research questions as follows.

- RQ1: How is the organization and specification of SIR’s SUTs?
- RQ2: Why SIR is widely used by researchers?

2. Methods
The scope of this research area includes all the C-language SUTs from SIR and its related organization as well as specification. This limitation was taken due to time constraint. The C programming language was chosen as the language studied since a survey conducted by Yoo and Harman [3] indicates that C-language SUTs was primarily utilized by researchers. The organization meant is how the directory structure of the SUTs was arranged and what is stored. Meanwhile, specification is the details of the SUT comprising name, fault type, program versions, LOC, number of test cases, number of faults, number of procedures, and size of the SUT. These identities were taken since we assumed that it could influence the researcher when choosing which SUT to be used as a research object apart from how the object was organized.

The SUTs was obtained from SIR repository after registration phase. The objects documentation was read with intent to know how those objects organized and how to obtain the corresponding specification. Some of the specification can be seen from SIR download page. Yet, the number of test cases and the number of faults is not listed on the page so that examination on each SUT is inevitably done in order to expose those specification and obtain reliable data. The objects organization and specification that have been found are recorded to table as an answer for RQ1.

In order to answer RQ2, the results at RQ1 were analysed and linked with related literature to construct approximate answers. This answer is just an approximation based on the characteristics discussed. A definite answer is too difficult to find since it involves the preferences of each researcher.

3. Results and discussion
We divide the results and discussion into the following subsections to make it easier for answering the RQs.

3.1. SIR’s objects organization in general
The following description of the object organization is answer for RQ1. The SUTs written in C programming language its function shown in table 1. The fifteen SUTs are the result of a merger from Siemens Corporate Research programs [8], the ORACOLO2 program [9], and programs made by the SIR project itself—the SUT created in Do et al. [6]. The "Siemens" programs are the first seven programs in table 1 (printtokens, printtokens2, replace, schedule, schedule2, tcas, and totinfo) and the "Space" program is the ORACOLO2 program—some SUT name may appear different from the original paper since it was edited by SIR. And the remaining seven (bash, flex, grep, gzip, make, sed, and vim) are Unix Utilities made by SIR.
Table 1. SIR’s SUTs written in C and its function.

| Name         | Description                                      |
|--------------|--------------------------------------------------|
| printtokens  | lexical analyzers                                |
| printtokens2 | lexical analyzers                                |
| replace      | pattern replace                                  |
| schedule     | priority scheduler                               |
| schedule2    | priority scheduler                               |
| tcas         | altitude separation                              |
| totinfo      | information measure                              |
| space        | Array Definition Language (ADL) interpreter       |
| bash         | shell script interpreter                         |
| flex         | pattern matching                                 |
| grep         | pattern matching                                 |
| gzip         | file compressor/expander                         |
| make         | makefile commands executor                       |
| sed          | stream editor for filtering and transforming text|
| vim          | text editor                                      |

SIR’s SUTs categorization was previously done by Khatibsyarbini et al. [10] who implicitly stated that C-language SIR’s SUTs are divided into three parts, namely Siemens, Space, and TSL. The TSL programs meant are flex, grep, gzip, and make. Nevertheless, the name “Unix Utilities” is more appropriate since the SUT object is a unix command and taken from the GNU site by SIR [7].

SIR divides the object organization into two parts namely "older" object organizations (for the Siemens and Space program) and "newer" object organizations (other than Siemens and Space) which will be further explained. The organization described below may be different from the real one at SUT (after extracted), so there will be a CONTENTS file in the directory or subdirectory that is unclear or different from the naming schemes to provide some additional information.

3.1.1. Older SIR’s object. The older SIR’s object refers to objects that are single version program where there is only one original version (v0) as the base program and some faulted versions (v1, v2, … vk). Objects are organized into subdirectories shown in table 2 (sorted as in the documentation).

3.1.2. Newer SIR’s object. Newer SIR’s objects refer to multi-version program objects that have a sequential version in the original (fault-free) version of the SUT. Table 3 shows the organization of the new SIR object and how it differs from the previous organization. Same as Above (SAA) refer to the description that is same with the previous organization in table 2.

The generic structure of the newer C SUTs object are mainly the same with older object. However, the files contained in it and the interpretation of the files may differ. Further information about this can be found in the C Object Handbook on the SIR webpage at Software-artifact Infrastructure Repository [7].

3.2. SIR’s objects specification

Table 4 provides a summary of the SIR’s SUTs specification as the answer for RQ1. As mentioned in Section 2, these eight identities (as columns in the table) can be expected to be considered when using SIR objects as research objects.
### Table 2. Object directory information for older C objects.

| Directory Name | Description |
|----------------|-------------|
| source.alt     | Contains “variants” of the source code of the non-faulted base version. Since Siemens and Space only have one base version, this directory contain “source.orig” which store the .h, .c, and makefile if needed. |
| source         | Empty directory for storing a version of a program that used in the experiment. Contains “variants” of the source code of the program. Siemens and Space program only have “version.orig” subfolder that contain faulted version of the program that derives from source.alt. |
| versions.alt   | An empty directory, used during some experiments to hold either a version, or a directory of versions, used in the experiment. |
| versions       | Testing information for the objects or known as test suites (which ones of the test cases should be called). Also contain “universe” file. |
| testplans.alt  | An empty directory, used during some experiments to hold test suites needed by that experiment. |
| testplans       | Files containing inputs or startup directories used by programs in various tests. These could be grouped into subdirectories or distinguished by naming conventions if helpful, in which case a CONTENTS file would help. |
| traces         | Place to permanently store the outputs of test runs. This is useful for regression testing where new outputs are compared (‘diffed’) against previous outputs. This directory could contain subdirectories v0, v1, ... vK to differentiate between versions. |
| inputs         | An empty directory, used during some experiments to hold test outputs. |
| outputs.alt    | Documentation about the object. |
| outputs        | Original materials obtained on the objects/real program before added with file to support testing research. |
| docs           | The directory from which experiments are run. This directory may also contain saved scripts, such as runall.sh and gettraces.sh scripts that run all tests in the universe. |
| originals      | Other information about the program, especially information gathered by analysis tools and requiring saving for experiments. Most likely containing a fault matrix, fault index information, versions matrix, a sensitivity matrix for that version. |

### Table 3. Object directory information for newer C objects.

| Directory Name | Description |
|----------------|-------------|
| Source, versions, testplans, traces, inputs, outputs, docs, originals, scripts | Contains “variants” of the source code of the program (.c and .h extension), presented in subdirectory v0, v1, ... vK. In the newer C SUTs, this directory contain the non-faulted program (in “versions.orig”) and the faulted program (in “versions.seeded”). |
| versions.alt   | Contains subdirectories v0, v1, ... vK, each of which contains testing information for that version. General files on the vJ directory are TSL specification (.tsl), frames generated from .tsl files (.frame), STImPL files (.tsl.universe or .cov.universe). |
| testplans.alt  | This directory contains subdirectories v0, v1, ... vK for the versions of the object; these subdirectories may hold trace information in the form of individual test traces. |
| traces.alt     | SAA. Although, fault matrices content presented differently than the older objects. Subdirectory of versions may appear to inform fault matrices of the corresponding versions. |
| info           | SAA |

The information and specification of SUTs listed in table 4 will be slightly different from those published in Do et al. [6] as well as in research papers that utilizing SIR’s SUTs. This may happen because several
SIR’s SUTs have been updated to support research implementation and the use of certain tools while the archives of old objects (before being updated) still accessible and usable.

Table 4. C-language SUTs specification.

| Name      | Fault Types | No. of Original Versions | No. of Faulted Versions | LOC | No. of Test Cases | No. of Fault | No. of Procedures | Size (Mb) |
|-----------|-------------|--------------------------|-------------------------|-----|------------------|--------------|-------------------|-----------|
| printtokens | Seeded      | 1                        | 7                       | 726 | 4130             | 7            | 18                | 122       |
| printtokens2 | Seeded      | 1                        | 10                      | 570 | 4115             | 10           | 19                | 95        |
| replace    | Seeded      | 1                        | 32                      | 564 | 5542             | 32           | 21                | 203       |
| schedule   | Seeded      | 1                        | 9                       | 412 | 2650             | 9            | 18                | 50        |
| schedule2  | Seeded      | 1                        | 10                      | 374 | 2710             | 10           | 16                | 55        |
| tcas       | Seeded      | 1                        | 41                      | 173 | 1608             | 41           | 9                 | 24        |
| totinfo    | Seeded      | 1                        | 23                      | 565 | 1052             | 23           | 7                 | 33        |
| space      | Real        | 1                        | 38                      | 6199| 13585            | 38           | 136               | 1454      |
| bash       | Seeded      | 7                        | 6                       | 59846| 1061            | 32           | 1061              | 79        |
| flex       | Seeded      | 6                        | 5                       | 10459| 570            | 81           | 162               | 14        |
| grep       | Seeded      | 6                        | 5                       | 10068| 809             | 57           | 146               | 13        |
| gzip       | Seeded      | 6                        | 5                       | 5680 | 217            | 59           | 104               | 18        |
| make       | Seeded      | 5                        | 4                       | 35545| 1043            | 35           | 268               | 261       |
| sed        | Real        | 8                        | 7                       | 14427| 370            | 32           | 255               | 10        |
| vim        | Seeded      | 8                        | 7                       | 122169| 975            | 22           | 1999              | 163       |

The columns in table 4 are described as follows.

- The "Name" column contains the name of the SUT object.
- The "Fault Types" column contains the fault types attached to the SUT. Fault types are divided into two, namely Seeded and Real. Seeded indicates that the fault is planted in the program source code on purpose and given the fault-id. Real indicates that the SUT has a real fault and was found during the testing process.
- The “No. of Original Versions” column contains the number of original versions of the SUT object. The original version is version of a program that has not been implanted with a fault (or fault-free).
- The “No. Faulted Versions” column contains the number of program versions that already embedded with faults.
- The “LOC” column shows the sum of the LOC in each source code file. The LOC meant is the whole LOC (comment included), not the number of executable LOC.
- The “No. of Test Cases” column contains the number of test cases that available to test against the program. The number of test cases was obtained from the fault matrices that contained in the SUT.
- The “No. of Fault” column contains the number of faults planted on the SUT. In the multi-version program, the number of faults meant is the sum of faults available in each version. The sum of faults is obtained from the “FaultSeeds.h” files which is available in each version. For the Siemens and Space fault objects, each faulted version of the SUT contains only one fault so that the number of faults is equal to the number of faulted versions.
- The “No. of Procedures” column contains the number of functions/procedures (also known as method) defined in the SUT.
- The “Size” column shows the size of the SUT after extracted in Megabytes.

3.3. SIR utilized among researchers
From the SUT object organization that has been described in the previous section, it can be concluded that SIR is indeed prepared to conduct research with the existence of a "staging folder" as a place when running the SIR’s objects. The specification of the SUT objects also may influence researchers on why
they tend to use SIR’s SUT in their research considering that the fault-embedded version of the programs, test cases, and test suites has been carefully prepared so that researchers only need to design testing techniques without the need to think about test programs and test cases.

An SLR conducted by Khatibsyarbini et al. [10] showed that the C-language SIR’s SUT that mostly used in research papers is the Siemens program, Unix Utilities, followed by Space. This can be due to the Siemens program has a relatively smaller LOC (also lesser source files) than other objects and provided with more test cases. This is advantageous for researchers focused on regression testing who seek to reduce the number of test cases executed while performing software testing. Another factor that may influenced the tend to select Siemens programs is because Siemens programs earlier presented to the research world than other SUTs, namely in 1994 by Hutchins et al. [8].

4. Conclusion
This section describes the research conclusion and future works for researchers who will conduct the same research.

4.1. Overall summary
From the findings that have been described, SIR’s SUT has an object organization that is specially made and organized for conducting software testing research. This is indicated from the existence of “staging folder” and some files such fault-embedded program versions, test cases, test suites, fault matrices etc. This organization and specification of SIR’s SUT that carefully prepared for conducting research are the reason why software testing researchers tend to use SIR’s SUT since they only need to design test execution techniques without the need to design test objects and test cases.

4.2. Future works
Further research investigating the other SIRs SUTs will help to complete the profound discussion of SIRs SUTs.

References
[1] Kumar D and Mishra K K 2016 The Impacts of Test Automation on Software’s Cost, Quality and Time to Market Procedia Comput. Sci. 79 8–15
[2] Huang R, Sun W, Xu Y, Chen H, Towey D and Xia X 2020 A Survey on Adaptive Random Testing arXiv XX Xx
[3] Yoo S and Harman M 2012 Regression testing minimization, selection and prioritization: A survey Softw. Test. Verif. Reliab. 22(2) 67–120
[4] Solanki K and Singh Y 2014 Novel Classification of Test Case Prioritization Techniques Int. J. Comput. Appl. 100(12) 36–42
[5] Singh Y, Kaur A, Suri B and Singhal S 2012 Systematic literature review on regression test prioritization techniques Inform. 36(4) 379–408
[6] Do H, Elbaum S and Rothermel G 2005 Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact Empir. Softw. Eng. 10(4) 405–435
[7] Software-artifact Infrastructure Repository: Home [Online] Retrieved from: https://sir.csc.ncsu.edu/content/sir.php [Accessed on: 30-Nov-2020]
[8] Hutchins M, Foster H, Goradia T and Ostrand T 1994 Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria Proc. - Int. Conf. Softw. Eng. pp 191–200
[9] Vokolos F I and Frankl P G 1998 Empirical evaluation of the textual differencing regression testing technique Conf. Softw. Maint. 1 44–53
[10] Khatibsyarbini M, Isa M A, Jawawi D N A and Tumeng R 2018 Test case prioritization approaches in regression testing: A systematic literature review Inf. Softw. Technol. 93 74–93