Summer 8-2018

Scaling up an Infrastructure for Controlled Experimentation with Testing Techniques

Wayne D. Motycka
University of Nebraska - Lincoln, wmotycka@cse.unl.edu

SCALING UP AN INFRASTRUCTURE FOR CONTROLLED EXPERIMENTATION WITH TESTING TECHNIQUES

by

Wayne D. Motycka

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Gregg Rothermel

Lincoln, Nebraska

August, 2018

SCALING UP AN INFRASTRUCTURE FOR CONTROLLED EXPERIMENTATION WITH TESTING TECHNIQUES

Wayne D. Motycka, M.S.

University of Nebraska, 2018

Adviser: Gregg Rothermel

Software testing research often involves reproducing previous experimental results. Previous work created a repository infrastructure for containment and dissemination of testable research subjects using a private centralized storage mechanism for hosting these test subject archives. While this is a good way to store these subjects it can be inefficient when the size of subjects increases or the number of versions of the subject's source code is large. The delivery of these large subjects from a centralized repository can be quite large and on occasion may not succeed requiring the user to repeat the download request. Coupled with the limited resources available to maintain the repository and package the subjects for wide distribution, an improved method for storing and delivering the subjects is desirable.

The research presented here explores alternative methods for storing and delivering the source code of research subjects that are currently being used. We describe a different approach to packaging testing research subjects within a standardized scheme that can significantly reduce the download time required for subjects with large source code compilations and multiple versions. The data presented shows that this storage of the source code using a secondary web service may introduce some additional cost to the installation time of multiple versions and at times improves it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software testing is an important part of the software development effort; it also is a large part of the overall cost of any software project [3]. Industry has a vested interest in minimizing this cost, thus providing impetus to investigate testing within the academic setting. Researchers on software testing have often relied on empirical evidence, thus performing experiments on software artifacts in order to derive new techniques and methods for approaching the testing effort. Hutchins et. al. [18] developed a standardized set of artifacts which formed a common basis of test subjects for use in research. These subjects, however, were not pervasively used, due in part to limited availability. Thus, researchers continued to develop their own subjects in isolation for use in evaluating techniques. While these empirical studies have resulted in many new approaches to testing activities, without the ability to compare techniques using common testing artifacts, a new technique could not be gauged relative to previous ones.

Elbaum [9] and Andrews [1] noted the need for standardized and widely available test subjects. Thus a push was made by the research community to develop code repositories containing faulty code, tests and oracles. Do et. al. [8] further

clarified this need for a repository, and to that end the Software Artifact Infrastructure Repository [29] (SIR) was created. The SIR's central location on the Internet provided researchers with several test subjects that included tests with oracles and known faults that could be enabled programmatically. The desire for multiple versions representing software evolution was also a driving factor; thus, many of the test artifacts of the SIR embody this valuable property also.

The initial SIR repository was quite successful in disseminating a core set of artifacts. However, the test subjects were frequently of small size, and thus provided a limited view of how effective a testing technique may be. Since the inception of the SIR the evolution of systems, and the software developed for them, have exhibited a general trend towards larger code bases. These larger size subjects are more representative and by implication more desirable for use in testing research. The availability of software to use in testing also improved with the use of public source repositories. These public web-hosted repositories are intended for sharing, co-authoring, and dissemination of source code and can be a fertile search space for experimental subjects. Hammoundi et. al. [17] leveraged these repositories to obtain research subjects. Their research demonstrated how the broad collection of software subjects provided by public repositories could be used in testing research studies. Public repositories also offer another useful aspect for software evolution testing research; namely, sequential versions. While prior subjects hosted by the SIR could provide distinct versions of evolving subjects as monolithic collections, the SIR method of packaging when applied on these larger code base artifacts and multiple versions became problematic. Specifically, when both code base size and numbers of versions increase. This results in massive test artifact archives. Eventually, a single archive becomes so large that the time required to download it is measured in hours, not seconds. This creates scalability problems for repositories.

This thesis presents a means for addressing this scalability issue. We propose a new packaging scheme for describing evolving subjects within a standardized testing artifact format. This format is adapted to leverage public repositories, while allowing researchers to contribute test artifacts resulting from their own work. The proposed packaging also provides enhanced maintainability and extensibility. Along with this packaging scheme, we describe scripting techniques that allow automation of the processes required to download and configure research subjects with an evolutionary perspective. We present results of an empirical study using both a traditional SIR packaging approach and our proposed packaging scheme. We present an analysis of the effect of our approach on download and installation time which shows that this manner of packaging improves the download time with either a small increase in total experiment replication time, and with some specific packaging methods providing improvement over the *traditional* packaging method.

The remainder of this thesis is structured as follows. First Chapter 2 examines the background and prior work related to how current repositories provide subjects with specifics on the SIR methodology. Chapter 3 specifies our proposed packaging approach along with the considerations that apply. Chapter 4 presents a study of our new method for packaging research subjects, and chapter 4.6 reports the results of measurements taken to determine the efficacy of the proposed packaging approach. Chapter 5 offers an analysis of the measurements and, finally, Chapter 6 presents conclusions derived from the study.

Chapter 2

Background and Related Work

In this chapter we discuss repositories that currently exist to enable testing research, and then describe the specific features and capabilities provided by the standardized SIR subject packaging format. This packaging format forms the basis for our proposed scaling adaptation.

2.1 Existing Repositories

As noted in Chapter 1, the SIR was created to facilitate replication of testing research results by making test artifacts freely available. The success of the SIR is evidenced by the thousands of users from hundreds of institutions and corporations that have downloaded subjects over the past decade. Owing to this ease of obtaining subjects, over 600 publications and many advances in testing research have been made using the SIR [30].

Since the inception of the SIR several other repositories have also been created. These repositories focus, however, on more specific aspects of testing. The subjects provided by the *Defects4J* [6] [22] repository focus on testing specific subjects with

"real-world" faults that can be activated selectively, with little or no evolutionary aspect considered. The subjects offered by the *SAMATE* [27] repository are intended for use in automated static analysis, again with no specific evolutionary concern. Research subjects provided by *SPL2Go* [33] do have some consideration of evolution in products, with less focus on faults and test suite experiments due to the intended target research interest. In a similar vein, the subjects provided by the *iBugs* [19] repository are intended to compliment those of the SIR by supplying real world faults that can be found programmatically, but the evolutionary aspects are of lesser concern.

The common theme in all these repositories is the same: providing subjects for research. Furthermore, a variety of means have been used to provide these subjects. Most repositories utilize the simplest form of web-hosting mechanisms which allows download of source code files either individually, or in archive collections. A few utilize more modern mechanisms and host subjects on web-hosted distributed version control systems. This latter strategy offloads the hosting effort, and avoids the problems of self-hosting. The limited number of these repositories is due in part to this problem.

The packaging scheme used by each of the aforementioned repositories also varies. SIR subjects which supply source code and faults as in-line compiler enabled blocks, with test replication and evolutionary aspects accommodated within the package specification. The iBugs repository employs a similar approach but with less focus on packaging for test replication and evolution. Defects4J provides source code and bugs as well, but again with little support for replication of experiments or evolutionary considerations. While these other repositories provide source materials, the disparate packaging styles are not as flexible or informative as those used by the SIR. With SIR, packaged downloads contain all relevant source code including alternate or subsequent versions, all tests, and installation scripts to facilitate replication.

2.2 SIR Artifacts

Using the SIR packaging method, a research subject's source code is stored along with requisite tests in a normalized directory tree structure as shown in Figure 2.1. The directory tree permits multiple copies of a subject arranged as versions to be provided along with tests specific to the version. Facilities within the directory tree can also be used to provide data resulting from experiments, along with test oracles and input data for tests. While most SIR subjects do not use all of these capacities, adherence to the structure allows users to have an idea where relevant aspects will be located. The existing set of SIR artifacts embody many characteristics of interest, and the packaging of them is documented and defined in a standardized format [31]. In the existing set of subjects, many are of small to moderate size; thus, even when providing multiple versions and test suites, the overall size of the artifacts provided by the subject is small. Any of these smaller subjects takes a small amount of time to download from the repository. Once downloaded, installing the subject for experimentation typically requires a minimal amount of time as well, often determinant upon the time required to perform a source code compilation. Thus, the ability to experiment with the subject is simple and the packaging allows quick initiation of test replication.

A few of the existing SIR subjects, however, reflect a trend towards a larger source code base. Subjects that currently exhibit this within the SIR are Derby DB [7] (a Java-based database server), and JBoss [21] (a Java-based Enterprise Application Platform (EAP)). These subjects possess multiple versions with injected faults applied, and once packaged into the SIR subject format they are between 1 and 2 Gigabytes (GB) in size. These subject are more interesting and relevant to testing research efforts but their size has led to download problems for users and the SIR

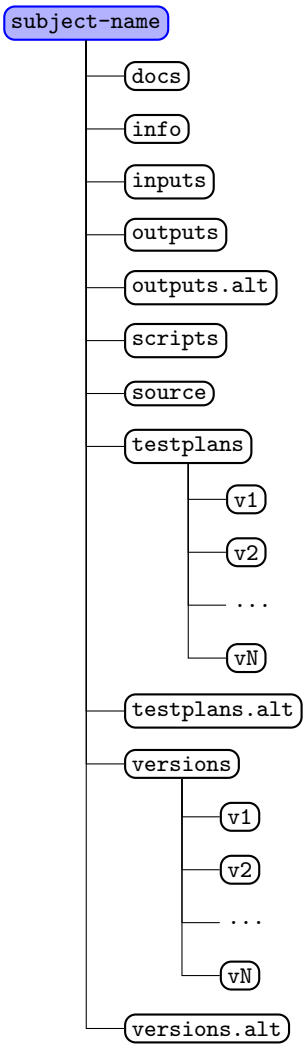


Figure 2.1: *Standard SIR Subject Directory Tree*

hosting server. On numerous occasions, researchers have been unable to download the large archives that comprise these subjects. This inability may be due to simple impatience or real network issues like congestion or imposed bandwidth limitations. Thus, these users will restart a download, resulting in additional SIR host resources being used, or the user will abandon the attempt. Consequently, large subject sizes cause download failure, real or perceived, and are problematic for the repository.

New subjects from our recent research into web testing evolution [17], when packaged using the traditional SIR packaging method, demonstrate a similar trend

toward excessively large size. These subjects are all examples of web site sources and several of them include quite large collections of source code. While the problem of downloading large code is not limited only to this type of subject, these web artifacts provide evidence that the overall trend in size increases described earlier is cause for concern and thus an impetus for deriving a scalable packaging paradigm.

Chapter 3

Proposed Approach

The aforementioned size issues form the impetus for the creation of a new subject packaging approach. Given that the general trend is towards larger code size collections for experimentation, how can the SIR, and presumably similar repositories, contend with download problems We propose a new way that test subjects can be provided within the artifact structure defined by the SIR. This new approach changes the manner in which artifact components are provided by leveraging modern web-host mechanisms. As a result, the net artifact sizes will be smaller by a factor of source code size multiplied by the number of versions, yet contain sufficient information to allow users to obtain the source code used in a research effort in order to replicate study results.

The specifics of the proposed approach are outlined here. First we discuss the constraints on the approach and the applicable requirements. Following this we describe the approach that we have formulated to facilitate the new source containment mechanism.

3.1 Requirements

To modify the structure of research artifacts, we need to be cognizant of the aspects that make an experiment replicable along with constraints imposed by the existing SIR artifact structure. We note in Section 2.1 that particular directory tree locations are used by the SIR to contain the source, tests, inputs, and scripting associated with a research artifact. Explicit definitions of how these are used will now be codified.

As mentioned in Section 2.2, the **versions** or the surrogate **versions.alt** directories are intended to contain the source code used for experiments. This must be maintained in the new approach, thus the following requirement:

Requirement 1. The new approach shall use the same directory tree hierarchy as a traditional SIR subject package.

It is also necessary to reflect the evolutionary or version aspect of artifacts under the **versions** or **versions.alt** directories as well. This characteristic is a defining feature of the SIR subjects and imbues a unique capability that users find quite useful. Hence the following requirement:

Requirement 2. The new approach must support version segmentation of artifacts.

To support the test suite and oracle components of the SIR subject specification, the new approach does not require any special adaptation. In the current SIR specification test suites are either applicable to all versions or are segmented in the same was as source code versions using subdirectories under either the **testplans** or **testplans.alt** directory.

Requirement 3. The new approach shall provide the same means of support for test suites and oracles.

Installation support for SIR subjects is a defining aspect of the SIR specification. The reliance of the new approach on external source code availability therefore necessitates the following:

Requirement 4: The new approach shall provide adequate scripting and documentation to inform users on how to use the provided source code location information to facilitate installation.

Requirement 5: Installation of a subject using the new approach must result in a source code image mirroring that used in the primary research experiment.

Requirement 6: Any remaining specifications pertaining to the SIR subject shall still apply to the new approach.

In addition to the requirements imposed by the SIR specification, we need to consider the new mechanism and its implications. To leverage the web-hosting mechanisms, the following requirements apply:

Requirement 7: The new approach must support file transfer using the the Internet *HTTP GET* protocol.

Requirement 8: The new approach must be adaptable to support protocols of distributed version control systems.

Satisfying these two requirements will enable the new packaging approach to furnish source from a variety of web-hosting locations.

Ultimately the approach also needs to provide usability comparable to existing subjects. Formalizing this need we state the following requirement:

Requirement 9: The new approach shall result in an installation time per version that is comparable to the install time required by the traditional packaging approach.

Comparable installation time in this context will naturally need to accommodate the need to download but this additional amount of time will still be less than the user effort to manually reproduce the version without any subject packaging. The overall time needed to download a traditionally packaged subject and install a version should always be larger than the time needed to install a single version in the new approach.

The new approach must also address the underlying problem, large subject packages.

Requirement 10: The new approach will support subjects of any source code collection size.

The task of package preparation should also be considered by the new approach. At a minimum, the effort required to create a new subject package should not increase. Ideally, easing the burden of packaging subjects is desirable.

Requirement 11: The new approach should not impose any additional effort on preparing new subjects than the amount required for *traditional* packaging.

Requirement 12: The new approach shall make subject packaging easier when possible.

The existing SIR specification allows for enhancement of subjects by adding versions and tests. This capability needs to be provided by the new approach.

Requirement 13: The new approach shall allow the versions and tests provided with subjects to easily be extended.

3.2 Implementation

In the interest of clarity we introduce two terms that will be used in the remainder of this text. We use the term of *traditional* packaging to indicate how a traditional SIR subject would be constructed. This packaging method requires source code copies for all versions to be provided within a single download archive. The term *concise* packaging as stated earlier refers to our new proposed packaging method. Further, the directory structure defined by the SIR [31] and depicted in Figure 2.1 is used as the basis of the logical arrangement discussed.

To fully appreciate how the the proposed approach differs from the traditional packaging method, we offer a short description of how subjects are packaged using the traditional approach currently used by the SIR.

3.2.1 Traditional Packaging

In *traditional* packaging subject source is placed under either the directory **versions** or **versions.alt** segment with subdirectories labeled using the convention of *v1* to *vN* delineating each version. Similarly, the tests applicable to each version are stored under the **testplans** or **testplans.alt** directory tree within each subject using the same subdirectory convention. This manner of packaging encapsulates all versions of the source code along with the tests for all versions in a logical manner. This *traditional* method however can suffer from excessive size as discussed earlier.

Traditional packaging obviously satisfies all the requirements with the exception of Requirements 7 through 12, which apply only to the new approach.

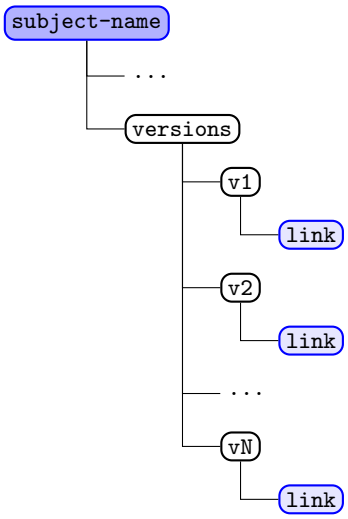


Figure 3.1: *SIR Subject Concise Versions Directory Contents for URL Download*

3.2.2 Concise Packaging

To meet our requirements, the new *concise* approach needs to provide source code to users while avoiding the large size file delivery problem. To do this, a solution is to use ordinary web download capabilities or modern web-hosting mechanisms. Modern web-hosting mechanisms such as GitHub [13], BitBucket [2], SourceForge [32] and the open source GitLab [14] repositories offer a solution that was not available when the SIR was conceived. The artifacts contained there may not be specifically intended for testing research studies but can be adapted.

To satisfy Requirement 1, the *concise* packaging approach must maintain compliance with the required directory tree structure. With *concise* packaging the root SIR artifact directory tree layout is the same, thus satisfying Requirement 1, but instead of placing the subject source code in subdirectories of the **versions** or **versions.alt** directory, each subdirectory *v1* to *vN* contains a file consisting of the URL [36] to the web-host server. In this way, the *concise* approach satisfies the version segmentation of Requirement 2. The activity of creating reference files also relieves the burden of collecting source code under each subdirectory; thus, Requirements

11 and 12 are addressed by the *concise* packaging. This referential style, however, requires accommodating the different protocols.

For *concise* packaging of subjects stored as archives of source trees and downloaded using the *HTTP GET* mechanism, a file named *link*, as shown in Figure 3.1, offers a standardized means of defining the web-host location. In this way, the *concise* approach satisfies Requirement 7. Each *link* file refers to an archive of the entire source code of that version, thus satisfying Requirement 5. This reference can then be used to download the appropriate version archive from the command line using tools such as **curl** [5] or **wget** [38] and via appropriate scripting can satisfy Requirement 4.

For *concise package* subjects to satisfy Requirement 8, we developed the following standardized method. A reference to a Git [12] repository web-host requires two elements to obtain a particular version of the source. We define a pair of files containing the Git URL and Git commit identifier, named *giturl* and *gitcommit* respectively, that are placed under each version subdirectory as depicted in Figure 3.2. The URL provided by the *giturl* file allows downloading of the source files as a project from the specified Git web-host system. The *gitcommit* file is used to configure the Git project to the specific source version of the testing subject.

Satisfaction of Requirements 3 and 6 is accomplished by applying the *traditional* approach for test suite inclusion to the new *concise* packaging approach. This consistency allows users to easily locate the appropriate test suites and oracles that apply. The test suites and oracles historically represent only a small fraction of the total size of the SIR subject; thus, adopting this strategy is the optimal choice. Our empirical study, however, reveals that incorrect test suite preparation can adversely affect the improvements offered by the *concise* approach.

Both the *traditional* and *concise* approaches are ultimately provided to users using

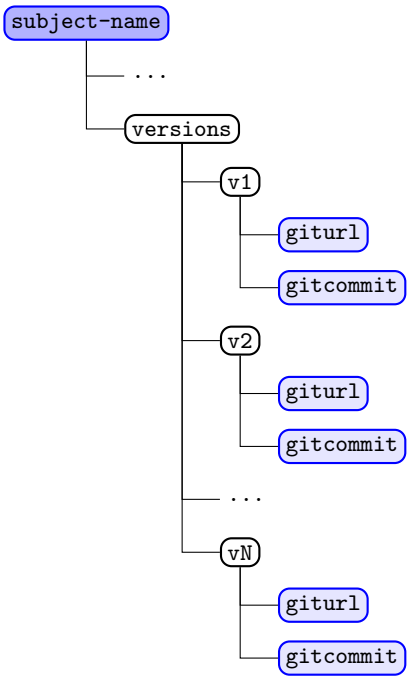


Figure 3.2: *SIR Subject Concise Versions Directory Contents for Git Download*

the same encapsulation. Each packaging method is placed into an archive, using the **tar** [34] utility and compressed using the **gzip** [16] tool. The resulting archive file forms the downloaded subject users will receive from the repository. Net reduction in subject size and how the two approaches intend to provide source code collections is depicted in Figure 3.3. The expected reduction provided by the *concise* approach directly addresses Requirement 10 through this subject size reduction.

The *concise* approach addresses requirements 9 through the segmentation provided by the versions. It is expected that the size of one version will be a fraction of the *traditional* packaging. Specifically, it will be approximately:

$$\frac{1}{\#\ of\ versions} \times traditional\ package\ download\ time$$

This expectation applies to both *HTTP GET* and Git storage methods. The support for versions allows extension of a subject with new versions, thus satisfying Require-

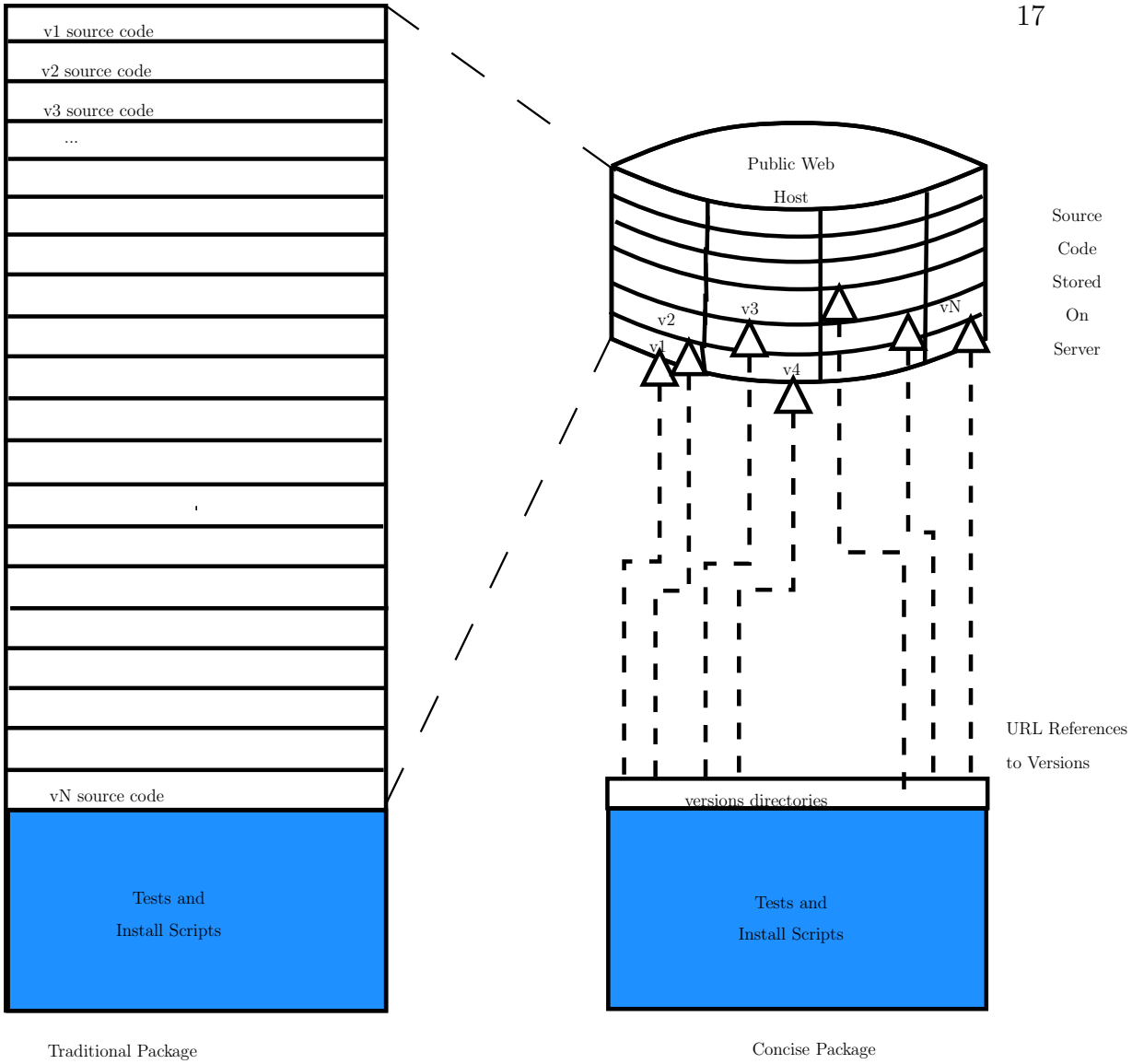


Figure 3.3: *Traditional and Concise Packaging*

ment 13. Any subject without versions would typically not benefit from using the *concise* approach.

3.2.3 Advantages of using Distributed Version Control Systems

Git and other distributed version control (DVC) systems allows for some useful capabilities that the *HTTP GET* download mechanism of source acquisition does not.

The Git project packaging provides information that allows selection of a particular version using the **git checkout** [10] command on a local copy of the Git project. This eliminates the need to download a full copy of each subject version that is done using the *HTTP GET* method.

Other distributed version control systems, e.g. Mercurial [25], SVN [4], and CVS [15], can also be supported using similar file naming conventions. For the purposes of this work we did not utilize any subjects hosted on these systems, so we defer definition of the file name conventions to accommodate these systems to future work.

The use of files to contain web-host location information also allows for another feature offered by the *concise* packaging approach. Due to the use of simple text files, multiple URL locations may be provided within them separated by line-feed characters. For subjects where the source code is available from multiple web-host services, these files can thus enumerate these locations. Scripting can then leverage this to sequentially visit each web-host URL location until the download succeeds. This feature adds a level of robustness via redundancy to the *concise* approach.

3.3 Subject Scripting

Scripting provided with each subject automates the installation tasks required for both the *traditional* and *concise* packaging methods. Each packaging method requires a slightly different set of steps to properly provision the subject and host system for test experiment replication.

The activities involved in installation using the *traditional* packaging method are depicted in Figure 3.4. In this packaging method, the subject source is included so it is simply a matter of selecting the source from the **versions** or **versions.alt** directory, un-archiving if needed, and performing the subject configuration and test

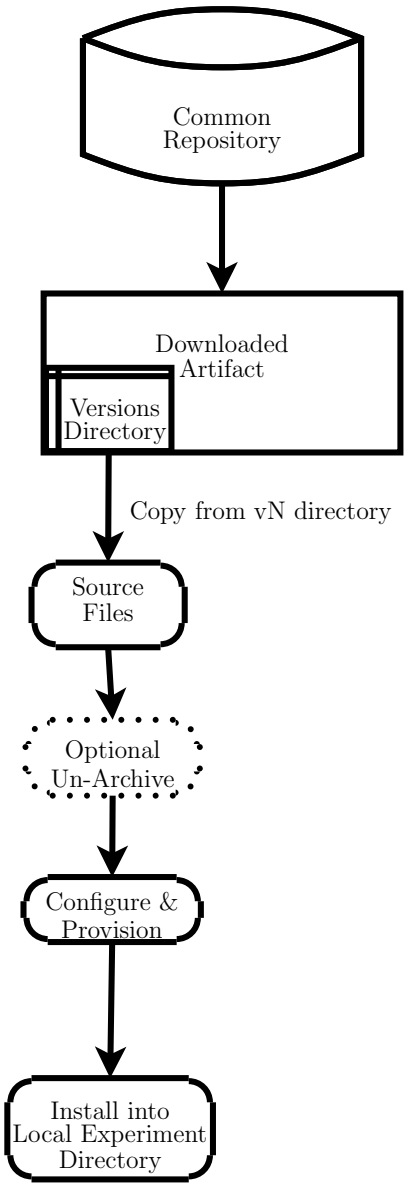


Figure 3.4: *Traditional Source Installation Process*

system provisioning. The configure and system provisioning steps, that may also contain a compilation activity, is the same for the *traditional* and *concise* packaging methods, and becomes a constant cost of installation borne by each. This step is somewhat unique to the type of SIR subject, some subjects may require compilation

while web testing subjects may need configuration file initialization and hosting server provisioning. The last common task for both packaging approaches performs the installation step to relocate the configured subject source to the location on the system where experiments can be executed. For our web testing subjects this was a directory on the system where an installed Apache web server would provide them to a web browser.

As noted in Section 3.2.2, the *concise* packaging method requires additional steps owing to the source reference mechanism employed. Figure 3.5 describes the steps needed to perform *concise* packaging installation. The installation process for these subjects uses the reference file contents located within the **versions** or **versions.alt** directory. When using reference files, the storage method determines how the installation will be performed. The process shown in Figure 3.5 reflects how the install scripts handles each of the storage methods used in our study.

The Link style reference, which uses a *link* file to indicate the web URL Zip storage location, uses the *HTTP GET* protocol to download the source code archive. In the Bash installation scripts this activity is done programmatically via the **curl** or **wget** Linux command. Once the download has been successful, the Zip archive is unbundled into the **source** location within the SIR directory tree.

The Git style reference utilizes the *giturl* file to indicate the web URL location of a Git repository. This web URL is then used by the **git clone** [11] command to download of the source tree package via the Git protocol. For maximum efficiency, the installation script checks for the presence of an existing Git project prior to performing the **git clone** activity as shown. If the Git project is present, the source tree is set to the desired version by applying the value from the *gitcommit* file for that version using the **git checkout** [10] command.

In both the *traditional* and *concise* packaging methods, the installation process

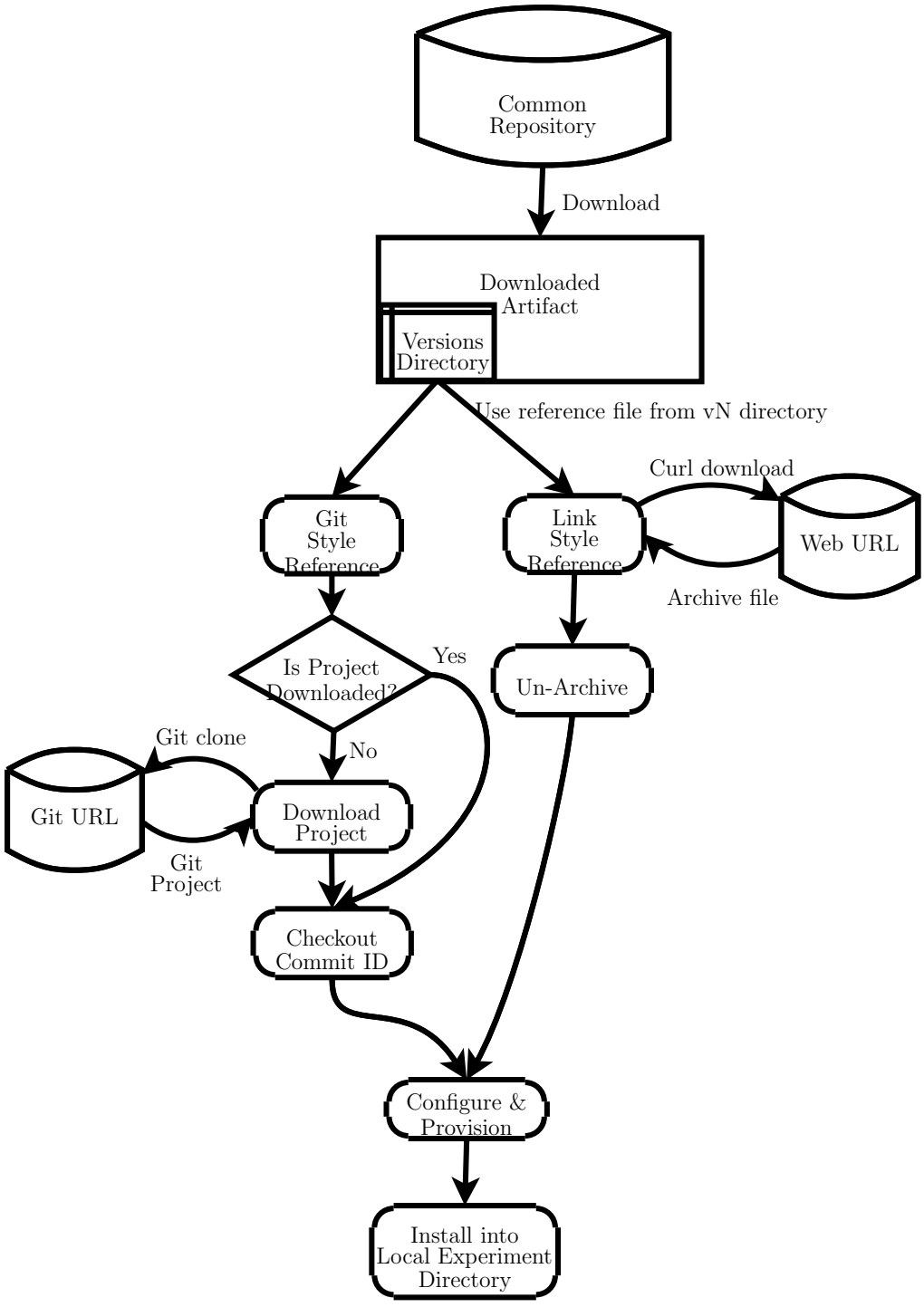


Figure 3.5: *Git and Link URL Source Installation Process*

ends with the same two steps: configuration and provisioning of the subject and the final installation into the local experiment directory. This supports the desired outcome expressed in Section 3.1 where installation results in source code identical to that used in the experiments performed with the *traditional* packaged subject.

The installation scripts are further customized for each subject due to their individual needs. Bash functions for subject-specific tasks were created to facilitate the configuration tasks each subject needs within the user's target system environment. Thus the scripts facilitate the mirroring of source code desired.

The tasks required to configure portions of the supporting packaging these web testing subjects use, e.g. configuring the relational database server MySQL/MariaDB [26] [24], have been implemented using Bash functions. Thus, these functions are reusable among these and future web testing subjects. The use of Bash functions also allows for simplification of the main installation scripts.

Chapter 4

Study

To assess the effectiveness of the proposed approach we performed a study on eight test subjects. The questions we want to answer in this evaluation are:

RQ1: How do the sizes of subjects compare when using the *traditional* and *concise* packagings?

RQ2: What is the effect on download time when using the *traditional* and *concise* packagings?

RQ3: What is the effect on installation time when using the *traditional* and *concise* packagings?

RQ4: How do the combined effects of both download and installation compare between the two approaches with regard to the time needed to install all versions of the subject?

4.1 Subjects of Analysis

The subjects we use in this study derive from our research into web test evolution. Each subject was chosen because it embodies aspects of interest in evolutionary testing studies, possesses a number of chronological versions, and is of overall non-trivial complexity. The subjects also exhibit changing and occasional faulty behaviors as well. These properties make them suitable for use in studying many types of testing research questions. A few of these subjects also embody qualities that are problematic for the current SIR repository to accommodate. Some subjects have very large source collections, and when combined with the need for multiple versions and the evolved test suites that were developed for them in our research, become massive archives.

Table 4.1 catalogs the subjects used for this work. The salient aspects highlighted here are the aggregate sizes of each subject in each packaging method, the number of consecutive versions provided within each subject packaging, and the web-host storage method. The sizes, reported in bytes, are the sizes of the compressed archive files in the SIR package format as described in Section 3.2.

Table 4.1: *Test Experiment Package Subjects*

Subject Name	**Traditional Size** (bytes)	**Concise Size** (bytes)	**# Versions**	**Storage Method**
Dolibarr	469,944,805	1,082,388	30	Git Repo
JavaMyCollab	3,655,633,690	785,802	22	Zip Files
PHP Agenda	8,802,166	324,945	29	Zip Files
PHP AddressBook	383,525,105	271,419,615	109	Zip Files
Joomla	441,780,013	743,111	83	Zip Files
YourContacts	97,130,072	2,732,474	47	Git Repo
PHP Fusion	108,556,013	2,065,919	41	Zip Files
MyMovieLibrary	140,931,105	126,267,045	20	Git Repo

Notable when considering the subjects is that some have large differences in sizes across the *traditional* and *concise* packaging methods.

The different storage methods listed in Table 4.1, Git Repository and Zip Files, indicate the two forms in which subject source is provided in a web-host location for use by the *concise* packaging method. The subjects used in this study consist of two types, those that utilize the *HTTP GET* file download capability and those that are hosted as Git repositories. For the purposes of this study, we used the University of Nebraska GitLab [35] system as the web-host server for both types of these subjects. The UNL GitLab instance can be used to source both *HTTP GET* downloads of the Zip files and the Git protocol, by uploading the Zip files into a repository instance on the UNL GitLab server and downloading these individually instead of as a single repository project download. Git subjects hosted on the UNL GitLab server are downloaded as projects using the Git clone [11] protocol.

4.2 Study Scope

In this study we want to simulate the actions of a user when replicating a testing experiment. The proposed changes introduced by our approach affect only two parts of this user activity, however: the download and installation phases. Thus, the study is limited to taking measurements of these two phases and ignores the actual execution of the testing experimentation (which would be the same in either the *traditional* or proposed *concise* approach.)

4.3 Variables and Measures

4.3.1 Independent Variables

The primary independent variable considered for this study is the subject size. The sizes of *traditional* and *concise* packagings reflected in Table 4.1 exhibit a range

from a minimum of 326 KB to a maximum of 9.6 GB which effectively represents three orders of magnitude variation in overall download sizes. Since the individual sizes of each version are aggregated within the *traditional* packaging, the downloads performed for the *concise* packaging should generally equate to the same amount of data being transferred with only a difference in when that transfer occurs during the two parts of the experiment.

A second independent variable in this study is formed by the number of versions included in the subject packaging. While this is reflected in the aggregate size of the *traditional* packaging subjects, it has a direct impact on the total time required for installation in either packaging form. Subjects having a large number of version archives will exhibit a greater installation time overall as the number of versions increases.

4.3.2 Dependent Variables

The dependent variables measured for this study are the download time for each packaging, *traditional* and *concise*, as well as installation time for the versions contained within each of these packagings. We also wish to evaluate the overall time needed to perform both downloads and installs of all versions for both packaging methods, in order to establish whether the *concise* packaging method obtains the desired time-agnostic outcome.

A significant factor affecting these dependent variables is related to network performance. This cannot be fully controlled without a dedicated network. Thus, some portion of these variables will reflect this dependence. The methodology we use to generate time values attempts to mitigate this non-deterministic effect as discussed below.

4.4 Study Approach

Each of the eight subjects of study was encapsulated using both *traditional* and *concise* packaging methods. prior to the study. This packaging included bundling each subject as an archive file using **tar** and **gzip**. As mentioned in Section 3.2.2, this is the same methodology used by all existing SIR subjects.

Once packaged, each of these subject versions was downloaded from the server using a PHP script that closely emulates the behavior of the actual download mechanism used by the SIR. In this way, a fair estimation of download time that would be required if the subject was actually hosted on the SIR can be obtained. To determine statistical relevance, each subject in both packaging methods was downloaded 10 times in an alternating order of *traditional* then *concise* packaging methods. The intent of this is to reduce bias in the download times introduced by the network bandwidth variability present in the supporting infrastructure. The download host server used for this part of the experiment was the UNL CSE system (https://cse.unl.edu). This system is composed of 64 Intel Xeon E7 4820 CPU cores running at 2.00 GHz and 132 GB of main memory. For a client system, a Dell Lattitude E5500 laptop was used, equipped with an Intel Core 2 Duo CPU @ 2.4GHz and 4GB of main memory, and a BroadCom BCM4322 801.11a/b/g/n wireless LAN card. All network transfers were done via the wireless interface of the client system.

For the second part of the study every version of each subject is installed sequentially using the installation scripts provided with the subject. These installs were again performed 10 times for all versions of each packaging method to obtain a statistical perspective on the performance of each with respect to time.

To obtain the download time measurements for the experiment, we used the PHP script on the server to compute the time required to initiate and complete the file

transfer using the previously mentioned SIR mechanism. For the installation time measurements we used the Linux **time** [23] command to collect wall clock time for each subject's install script execution.

We do not attempt to execute the actual tests on the subjects used in this study. The reason for this is that both packaging approaches result in identical subject source code being made manifest on the client system, thus test execution time would be equivalent.

4.5 Threats to Validity

The results of these experiments could potentially suffer from a few of the choices made. The choice of using the host server, CSE, may introduce a level of difference in download time results because it is not the exact same server used to host the SIR website. The SIR web site is hosted by a similar but distinct server within the Computer Science Department infrastructure, and typically is not as heavily used. The underlying differences between these systems, however, should not adversely affect the overall analysis since all experiments using the subjects were conducted using the same server.

A second threat is that of network congestion and overall available bandwidth during the experiment. Given that the time statistics being used in this study rely on network performance, which was not controlled, it can be argued that this more accurately represents the real world case instead of a laboratory setting. Thus, time measurements should align better with the expected actual conditions in which these subjects would be accessed and used.

A tertiary threat is introduced due to web-host availability. If a web-host is off-line, the subject source will not be available when requested using the proposed

concise packaging approach. We believe that this threat can be reduced via the aforementioned redundancy feature provided by multiple URL links within the *link* and *giturl* files. By hosting the Zip files or Git package on separate web-host servers, it can be argued that it is unlikely that both locations will be off-line at the same time.

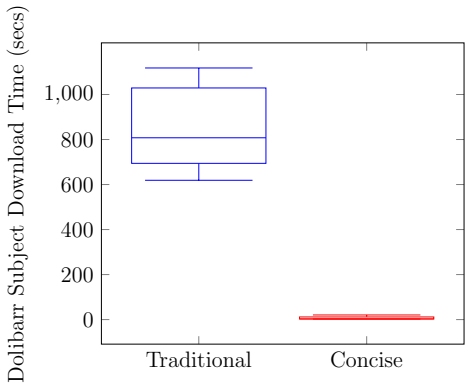
4.6 Experiment Results

Conducting the experiment described in Section 4.4 resulted in the data presented here. We present the measures of the dependent variable, time, for each object listed in Table 4.1, for both download and installation activities.

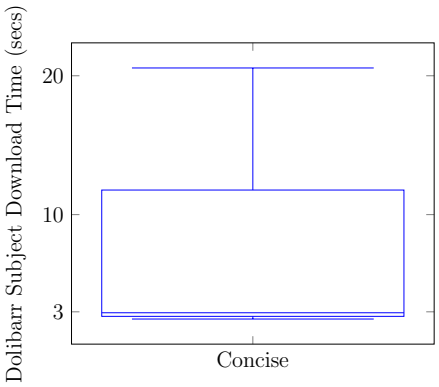
4.6.1 Download Comparisons

4.6.1.1 Dolibarr Download

A plot of time comparisons for Dolibarr is given in Figure 4.1a. From this data we observe that the amount of time required to transfer all of the source and tests from the repository is more than two orders of magnitude higher with the *traditional* packaging method than with the *concise* packaging. The times required to transfer the *concise* form of Dolibarr are expanded in Figure 4.1b. A mean time of 2.93 seconds is needed by the *concise* packaging method compared to a mean time of 807.71 seconds for the *traditional* packaging. These measures indicate a two order of magnitude improvement.

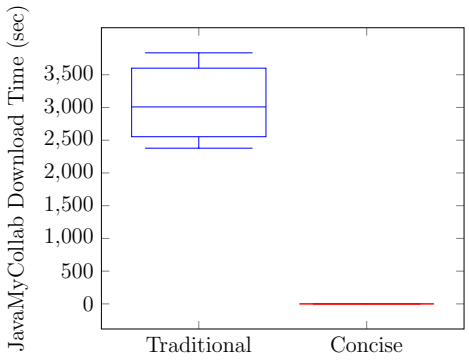


(a) Comparison of Both Methods

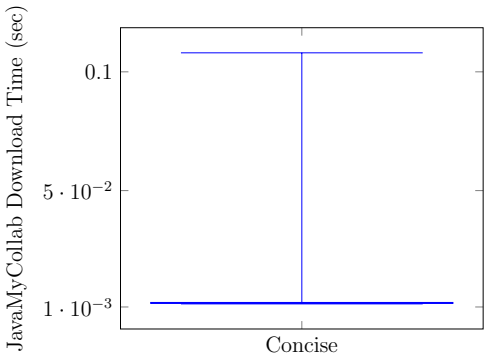


(b) Download Time for Concise

Figure 4.1: *Comparing Download Times for Traditional vs. Concise (Dolibarr)*



(a) Comparison of Both Methods



(b) Download Time for Concise

Figure 4.2: *Comparing Download Times for Traditional vs. Concise (JavaMyCollab)*

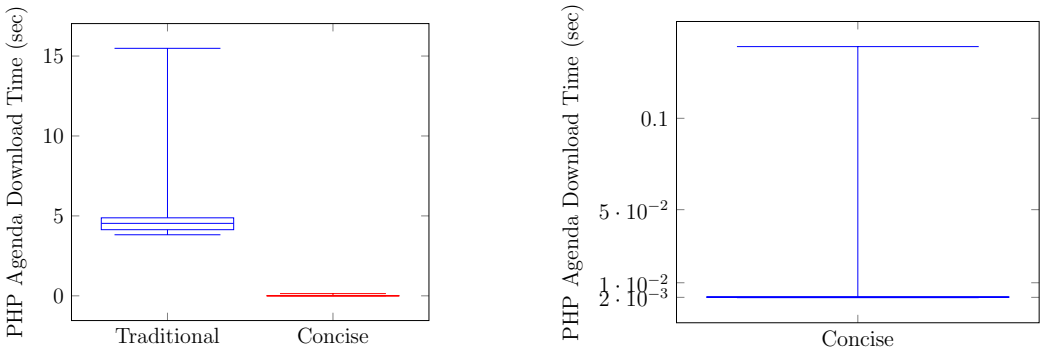
4.6.1.2 JavaMyCollab Download

The download time comparison for JavaMyCollab is presented in Figure 4.2. For this subject the download time of the *concise* packaging method is six order of magnitude faster than the *traditional* packaging with a median time for the *concise* packaging being 0.00263 second compared to 3009 seconds for the *traditional* packaging.

4.6.1.3 PHP Agenda Download

The download time measured for the *concise* packaging form of PHP Agenda is similarly much smaller than for the *traditional* packaging, as seen in Figure 4.3. Here the

median download time for the *concise* packaging method is 0.0029 seconds compared to 4.53 seconds for the *traditional* packaging.



(a) *Comparison of Both Methods* ***(b)*** *Download Time for Concise*

Figure 4.3: *Comparing Download Times for Traditional vs. Concise (PHP Agenda)*

4.6.1.4 PHP AddressBook Download

The level of improvement in the measured download time for the PHP AddressBook subject is less dramatic than in the foregoing cases, as shown in Figure 4.4. The median download time for the *concise* packaging is 218.96 seconds compared to 286.81 seconds for the *traditional* packaging. An examination of the constituent parts of the *traditional* packaging contents of PHP AddressBook reveals that the collection of all its source code versions contributes 79 MB to the size of the package. The test suites contained in both the *traditional* and *concise* packages contribute 350 MB. The size of the tests suites therefore dominate the size of both package methods. A deeper dive into what contributes to the test suite size considers how the tests were prepared. In this case, the tests were contained in zip archives that had the Selenium [28] testing Java archive (**jar** [20]) files enclosed with each version, thus containing multiple copies of the same static contents. This can be considered a mis-prepared test suite because it contains redundant code. A repackaging of these tests to have only one copy of the

requisite Selenium "jar" files along with the test sources and build scripts to generate tests would presumably make this subject display the same level of improvement seen on the previous source-size-dominant subjects.

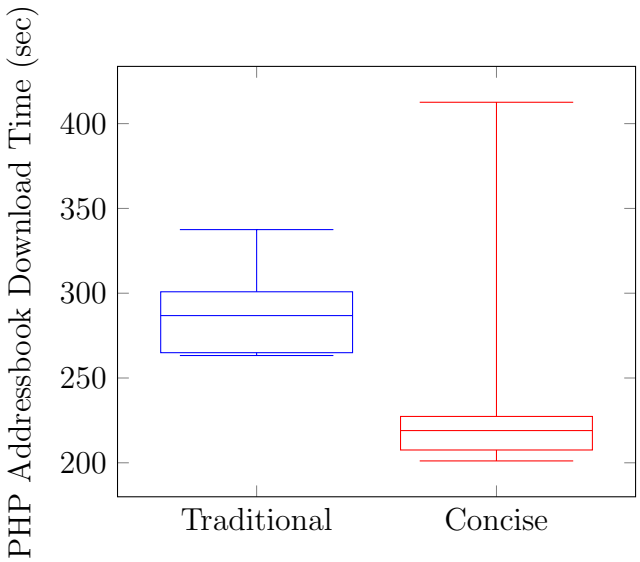
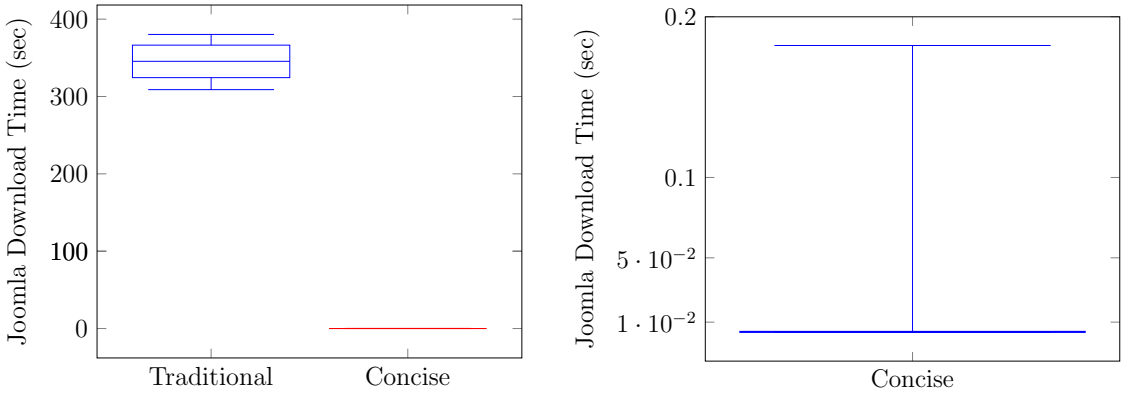


Figure 4.4: *Comparing Download Times for Traditional vs. Concise* (*PHP AddressBook*)

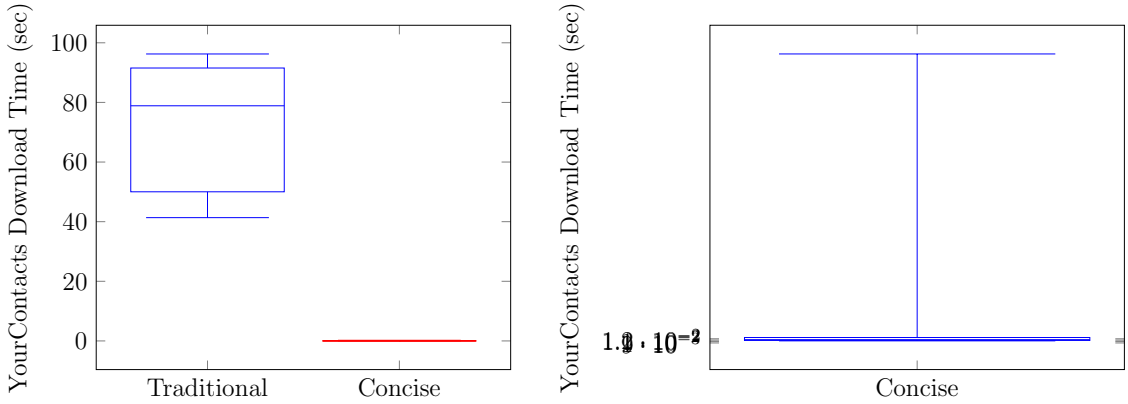
4.6.1.5 Joomla Download

The Joomla subject illustrates an intermediate sized example of the improvement possible using the *concise* packaging form. Again, in the comparison graph shown in Figure 4.5a, the difference achieved by removing the source in the *concise* packaging method is evident. The median download time for the *concise* packaging required 0.0038 seconds versus the median download time of 345.5 seconds needed by the *traditional* packaging. The high cost incurred in the *traditional* packaging in this case is directly related to the 83 versions, along with the fact that each version is a compressed archive ranging from 3.6 Megabytes (MB) to 7.8 MB in size.



(a) *Comparison of Both Methods* ***(b)*** *Download Time for Concise*

Figure 4.5: *Comparing Download Times for Traditional vs. Concise (Joomla)*



(a) *Comparison of Both Methods* ***(b)*** *Download Time for Concise*

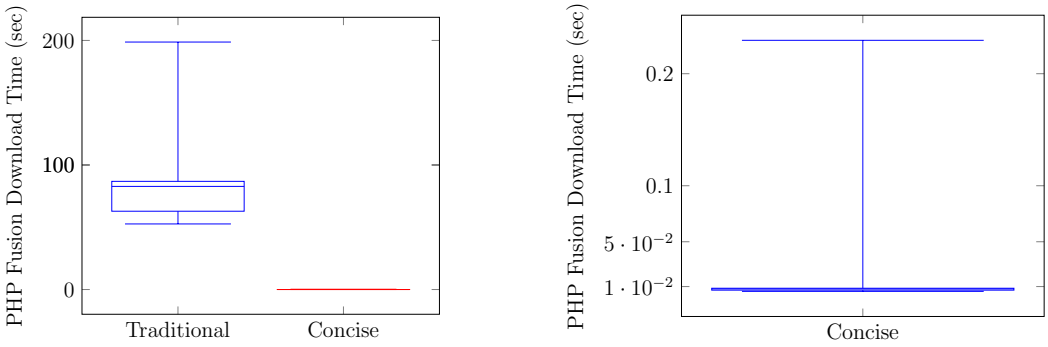
Figure 4.6: *Comparing Download Times for Traditional vs. Concise (YourContacts)*

4.6.1.6 YourContacts Download

The YourContacts subject comparison in Figure 4.6a also exhibits a large difference in download time. The median time needed to download the *concise* packaged subject is only 0.11 seconds and the median download time needed for the *traditional* packaging is 78.87 seconds.

4.6.1.7 PHP Fusion Download

PHP Fusion download data also shows a significant download time difference. The median times for downloading the subject are 82.8 seconds for the *traditional* packaging versus 0.0079 seconds for the *concise* packaging. While subjects like PHP Fusion would not typically benefit from *concise* packaging, the median download time improvement demonstrates how effective the approach can be.

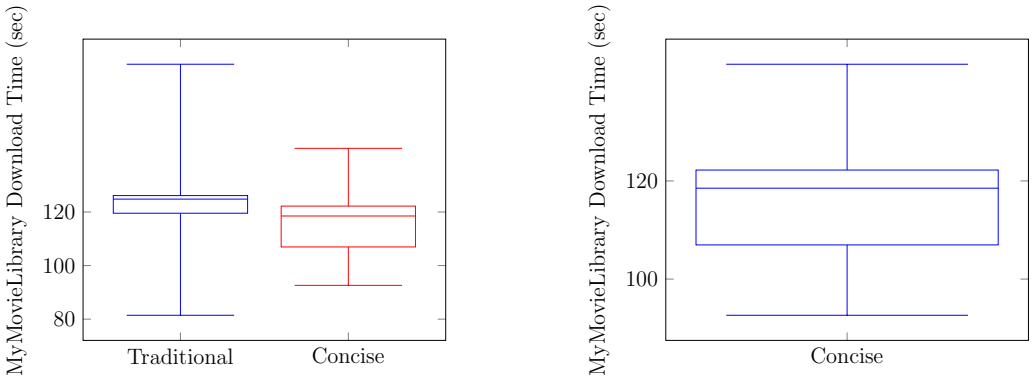


(a) *Comparison of Both Methods* ***(b)*** *Download Time for Concise*

Figure 4.7: *Comparing Download Times for Traditional vs. Concise (PHP Fusion)*

4.6.1.8 MyMovieLibrary Download

The final subject, MyMovieLibrary, also utilizes the Git repository mechanism. Comparing the download time characteristics for each packaging method, the median time for the *concise* packaging was 118.52 seconds and the median download time for the *traditional* packaging was 124.82 seconds. Examination of this subject reveals that again the tests dominate subject size. This subject also suffers from the mis-prepared test suite issues seen earlier in the PHP AddressBook subject with the source code collection for all versions requiring 16 MB and the tests contributing 50 MB to the package size.



***(a)** Comparison of Both Methods* ***(b)** Download Time for Concise*

***Figure 4.8:** Comparing Download Times for Traditional vs. Concise (MyMovieLibrary)*

4.6.2 Installation Comparisons

The underlying improvement offered by our proposed technique centers on deferring downloading of source code. This trade off of "when" source code is downloaded necessitates comparing the amount of time required to install subjects for the two methods. The *traditional* packaging method provides all source code in one large archive file whereas the *concise* packaging method downloads each version when required. Due to this difference in how installation obtains the source code, copying it from within the package or downloading when required, we expect installation times to be longer for the *concise* packaging method. Subjects using Zip storage, and using the *HTTP GET* download mechanism, will incur an additional cost per version over the *traditional* packaging method due to the download on installation approach used by the *concise* packaging. Subjects using the Git repository storage mechanism, however, should need to incur the cost of download only once when cloned [11] due to the use of Git checkout to select the version to install.

The data presented in this section reflects the median installation time for installing any one version of a subject using the *traditional* and *concise* packaging method. This perspective allows us to compare the two methods on a per-version ba-

sis and offers insight into the user experience. An examination of the user experience for installation of all versions is presented in Section 4.6.3.

4.6.2.1 Dolibarr Installation

The installation times for a single version of the Dolibarr subject are shown in Figure 4.9. Here we observe that the times required for installing any one version of the subject show almost no difference between the two packaging methods despite the added cost incurred by the Git download and version checkout/selection activities required by the *concise* packaging. The median time for installing one version under the *concise* packaging is 374 seconds whereas the median time under the *traditional* packaging is 367 seconds. The longer upper extreme exhibited by the *concise* packaging method reflects the additional time incurred by Git to download the source code when it is not already present. This is a one-time cost of using the Git storage method. Once a Git source code project is available, the version checkout/selection operation time is comparable to that needed to un-archive and install a subject using the *traditional* packaging.

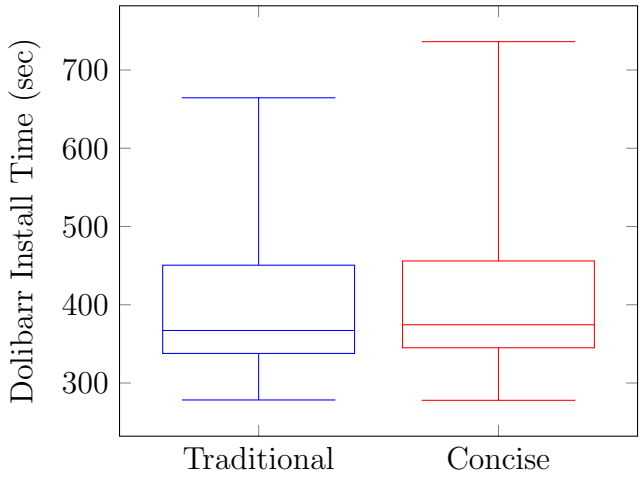


Figure 4.9: *Comparing Install Times for Traditional vs. Concise (Dolibarr)*

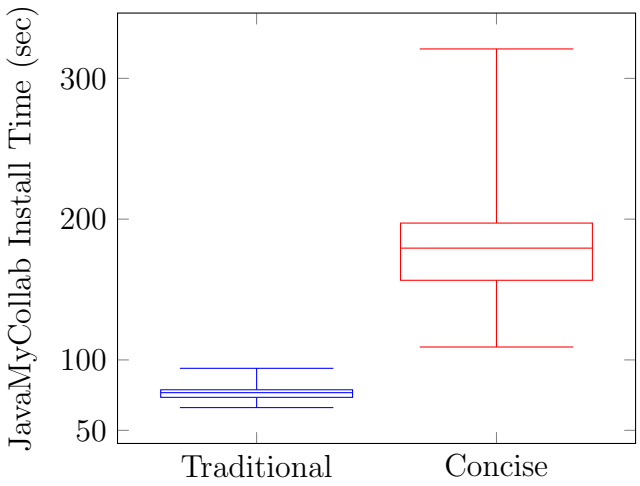


Figure 4.10: *Comparing Install Times for Traditional vs. Concise (JavaMyCollab)*

4.6.2.2 JavaMyCollab Installation

A comparison of the installation times of the JavaMyCollab subject shows a different time cost between the two approaches. In the case of JavaMyCollab, each version is stored as an archive on the web-hosting server and requires a download each time. The median installation time for the *traditional* packaging is 76.735 seconds whereas to download and install the subject using *concise* packaging requires 179.41 seconds. As expected, the time required to download one source code archive adds significantly to the single version install time a user would experience. JavaMyCollab source code archives range in size from 82 to 139 MB, which could also be a factor in the extremes present in Figure 4.2a.

4.6.2.3 PHP Agenda Installation

When viewing the installation times for PHP Agenda, download cost is negligible. Figure 4.11 shows that the median time required to install one version using the *concise* packaging was 5.695 seconds and median time for the *traditional* packaging was 3.59 seconds. Considering the several orders of magnitude difference in download

times seen in Figure 4.3, the small cost for installation provided by the *concise* packaging appears to be worthwhile for this subject.

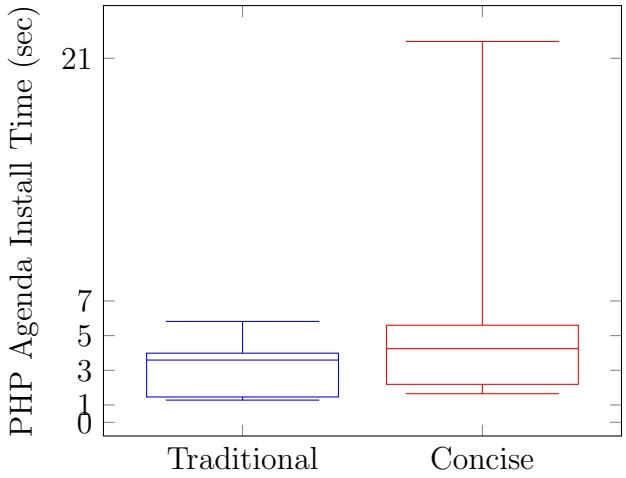


Figure 4.11: *Comparing Install Times for Traditional vs. Concise (PHP Agenda)*

4.6.2.4 PHP AddressBook Installation

PHP AddressBook has less improvement when viewing the installation time data. Figure 4.12 shows that the amount of installation time needed is between 2 and 3 times more for the *concise* packaging method, (a median of 3.28 seconds for the *traditional* packaging method versus a median of 6.15 seconds for the *concise* packaging.) While the difference in real time is small, the cost of download is evident with this subject. Of note is the fact that PHP Addressbook uses the Zip storage mechanism and each source code archive ranges from 68 KB to 3.6 MB. This may be a factor in the observed per-version installation time.

4.6.2.5 Joomla Installation

Joomla install times (Figure 4.13) follow the majority pattern as well. The median installation times differ by nearly 10 seconds (35.795 versus 45.745 seconds) which

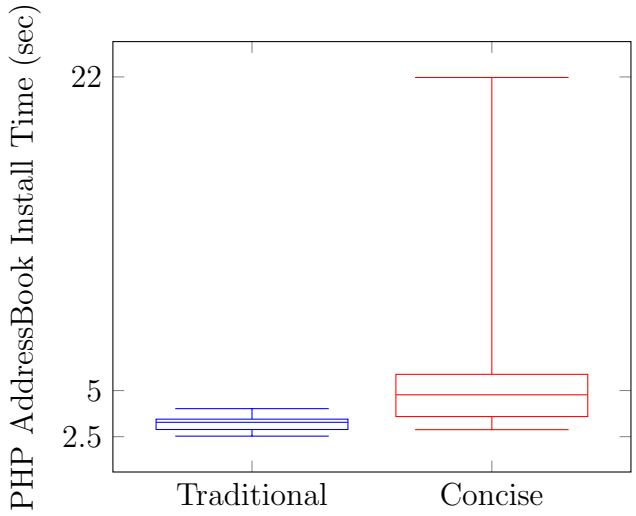


Figure 4.12: *Comparing Install Times for Traditional vs. Concise (PHP AddressBook)*

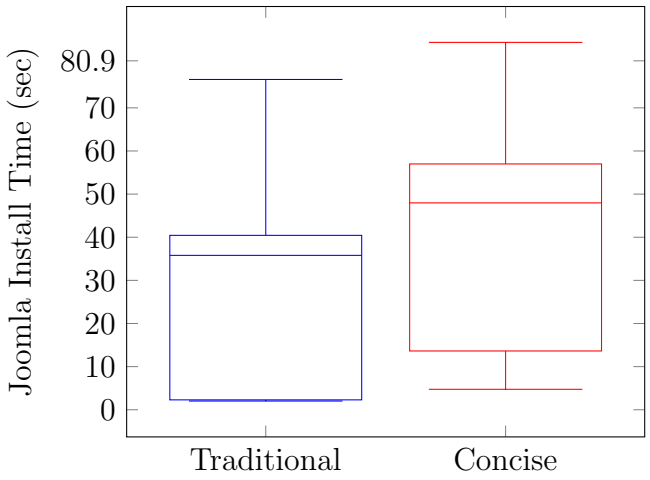


Figure 4.13: *Comparing Install Times for Traditional vs. Concise (Joomla)*

is not onerous considering the remainder of configuration and provisioning time required, as noted in Section 3.3. With a single version download size ranging from 3.5 to 7.7MB, the additional time seems to be in line with what we should expect.

4.6.2.6 YourContacts Installation

With YourContacts, the installation times (Figure 4.14) indicate a moderate increase in the median times observed for the two packaging methods of 1.3 seconds (2.965

versus 1.66 seconds). Since YourContacts uses the Git repository source storage, only the initial version causes the download, via a **git clone** operation, meaning the time required to select a version using the **git checkout** command adds only a modest additional cost.

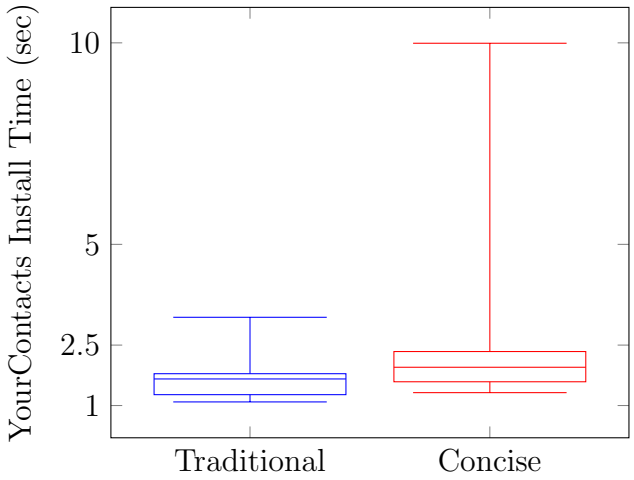


Figure 4.14: *Comparing Install Times for Traditional vs. Concise* (*YourContacts*)

4.6.2.7 PHP Fusion Installation

The PHP Fusion subject continues the pattern of small increases in median installation time observed in most of the prior subjects. As Figure 4.15 shows, the median install time for the *concise* packaged subject exhibits only a 1.23 second increase (4.385 versus 5.62 seconds). Considering that PHP Fusion users must download each version due to the Zip archive storage method, this result implies that the proposed *concise* packaging approach has a limited effect when subject source size is moderate.

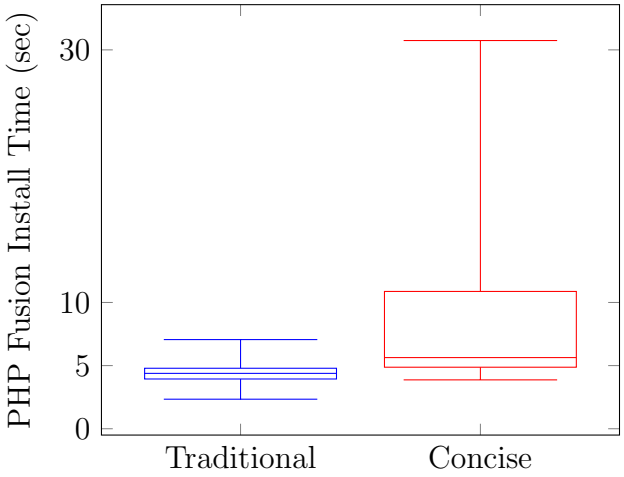


Figure 4.15: *Comparing Install Times for Traditional vs. Concise (PHP Fusion)*

4.6.2.8 MyMovieLibrary Installation

The data for the MyMovieLibrary installation (Figure 4.16) again reflects little change in installation time between the *traditional* packaging and *concise* packaging methods. Given that MyMovieLibrary uses a Git repository and has an small aggregate source size with each version, the time needed to acquire the subject is small, as is the requisite time for version changes via the Git checkout mechanism. The initial retrieval time cost is reflected in the upper bound of the concise plot, whereas the median time for each packaging method is nearly the same: 7.77 seconds versus 7.67 seconds. In fact, the majority of time required for the installation of MyMovieLibrary seems to be dependent on the common tasks needed to install the subjects, create a Venv [37] virtual environment, and include required Python packages.

4.6.3 Combined Time Comparisons

While both time metrics evaluated up to now show a decrease in download time and an increase in installation time, evaluating both together is necessary given the way these subjects are typically used. Users will experience both of these aspects when

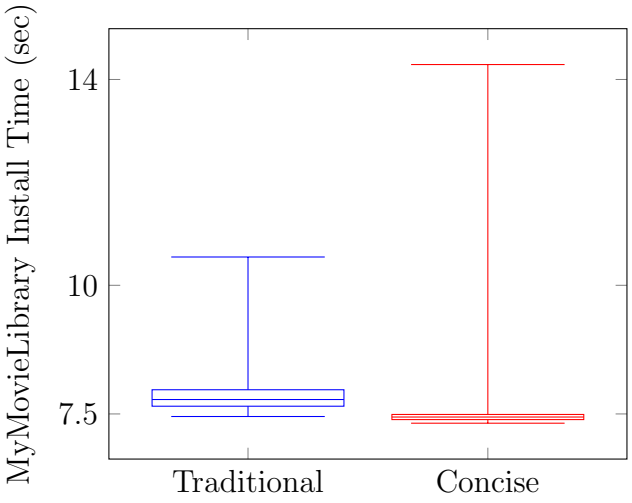


Figure 4.16: *Comparing Install Times for Traditional vs. Concise (MyMovieLibrary)*

utilizing these subjects, and thus an evaluation combining these measures can provide insights into the impact of the proposed packaging approach on the user's experiment workflow.

Determining how to evaluate this combined workflow effect, however, is not straight forward. Considering that both download and installation times are impacted by the *concise* packaging, and form an intrinsically additive relationship, then a simple addition of values may achieve this assessment. The additive relationship necessitates taking all aspects of these time measures into account, thus adding the median download time to the median accumulated time to install all versions is necessary if the comparison is to have sufficient similarity. This summation of installation times is not as simple, however, as it seems. The download and installation information in the previous sections provide only statistical range data and median values. Thus, a combination is not easily derived from the two range sets.

A proxy for both of these statistical sets is needed to perform the desired combined comparison. To that end, the use of median download and installation times may offer the intended comparative metric. In an attempt to address the variable nature

Table 4.2: *Combined Times for Subjects*

Subject Name	**Packaging Method**	**Median Dwnld Time**	**Median Install Time** (for All Versions)	**Total Time** (secs)	Dwnd S.D.	Install S.D.
Dolibarr	Traditional	807.71	11598.16	12405.87	153.48	147.61
	Concise	2.93	12188.82	12191.75	5.69	101.57
JavaMyCollab	Traditional	3008.65	1675.56	4684.21	506.14	29.32
	Concise	0.00263	3914.46	3914.46	0.029	248.56
PHP Agenda	Traditional	4.53	85.73	90.26	4.09	3.05
	Concise	0.00209	121.23	121.23	0.04	9.37
PHP AddressBook	Traditional	286.81	349.26	636.07	23.56	1.59
	Concise	218.96	582.44	801.40	59.86	15.06
Joomla	Traditional	345.50	2123.5	2469.0	22.84	188.49
	Concise	0.003769	3323.78	3323.78	0.054	57.19
YourContacts	Traditional	78.87	73.45	152.32	19.87	2.98
	Concise	0.011369	99.66	99.67	0.061	6.75
PHP Fusion	Traditional	82.80	182.69	265.49	39.47	10.62
	Concise	0.007897	329.30	329.31	0.067	22.10
MyMovieLibrary	Traditional	124.82	157.76	282.58	23.05	1.42
	Concise	118.52	152.58	271.10	16.02	1.83

evidenced within the statistical data, the standard deviation values are also presented in Table 4.2. In viewing the combined times, we reflect on the storage methods listed in Table 4.1. Dolibarr, YourContacts and MyMovieLibrary use the Git repository method. Owing to this, we observe that median installation times for the *concise* packaging of Git subjects differ by only a small amount from those of the times for the *traditional* packaged subjects in Table 4.2. The data indicates that Git repository subjects show small reductions in time. We deduce that this is due to the way source is provided in the *traditional* subjects using Zip archives. Thus, the un-archive step requisite during installation is effectively equal to the time required for the Git checkout activity used by the *concise* installer.

Table 4.3: *Git Subject Version Installation Statistics*

Subject Name	Median Git Clone Time	SD	Median Git Checkout Time	SD
Dolibarr	725.76	0.96	372.36	80.90
YourContacts	4.38	2.96	1.94	0.73
MyMovieLibrary	10.21	0.24	7.44	0.55

Another facet that we explored within the data was the time required for the initial Git clone activity. We offer some analysis of this in Table 4.3, where the median time required to install the first version is compared to the median time required to install any one of the remaining versions via the Git checkout command.The table data indicates that the Git clone activity time cost is relatively high compared to the time cost of changing versions using Git checkout. This initial Git clone cost, borne on the first installation attempt, becomes amortized across all versions.

A similar evaluation of the subjects in Table 4.2 employing Zip archive storage seems to indicate that installation costs result in an overall increase in total time.

When the standard deviation is considered for all of these total time assessments, we note that most of these total times reflect equivalent total time between the two approaches.

Chapter 5

Discussion

We now provide further discussion of our results; we begin by reiterating the research questions we stated in Chapter 4.

RQ1: How do the sizes of subjects compare when using the *traditional* and *concise* packagings?

RQ2: What is the effect on download time when using the *traditional* and *concise* packagings?

RQ3: What is the effect on installation time when using the *traditional* and *concise* packagings?

RQ4: How do the combined effects of both download and installation compare between the two approaches with regard to the time needed to install all versions of the subject?

In response to RQ1 we can see in Table 4.1 the sizes of subjects provided by the two packaging approaches. With this set of subjects, six of the eight evince a sizable reduction in size when packaged in the proposed *concise* method. In the case of the

remaining two subjects, PHP AddressBook and MyMovieLibrary, we noted in Section 4.6.1 that these subjects were mis-prepared, having test suites containing duplicated testing tool libraries. A percentage comparison of the difference in size between the two packaging methods is presented in Table 5.1. The percentages we show indicate the relative size reduction provided by *concise* packaging when compared to *traditional* packaging using the difference calculation of:

$$\frac{Traditional\ size - Concise\ size}{Traditional\ size} \times 100$$

Considering the percentages presented we conclude that our response to RQ1 is that a significant reduction in the size of subjects can be garnered by using the *concise* packaging.

Table 5.1: *Test Experiment Package Subjects*

Subject Name	**# Versions**	**Percentage Reduction**
Dolibarr	30	99.76968%
JavaMyCollab	22	99.97850%
PHP Agenda	29	96.30835%
PHP AddressBook	109	29.23010%
Joomla	83	99.83179%
YourContacts	47	97.18679%
PHP Fusion	41	80.96807%
MyMovieLibrary	20	10.40513%

Inspection of the download time reduction provided by the *concise* packaging in Table 5.1 indicates that for six of the eight subjects an 80% or greater reduction is attained (again, the remaining two subjects with lesser improvement exhibit the effects of a mis-prepared test suite.) These two subjects also have smaller source code collections in each version that also contributes to the minimal reduction, and with

small source code size the number of versions has less of an effect. If these results generalize, we can conclude that downloading subjects packaged in the proposed *concise* form will be much faster if the download time is dominated by source code and number of versions. This would seem to lead us to an answer to RQ2. If the source size and number of versions dominate the overall size of a subject then the *concise* packaging leads to a reduction in download time.

Installation time metrics are where the true cost of using the *concise* packaging method are expected to be seen. The figures in Section 4.6.2 indicate that the median installation time of *concise* packaging is almost always longer when median single version installation time is measured. The exception to this is MyMovieLibrary where median install time for the *concise* packaging is a few seconds faster than the time for *traditional* packaging. In general, we expected installation time to be longer for *concise* packaging due to the download-when-needed approach, thus the results for MyMovieLibrary are surprising. We suspect that this improvement in *concise* packaging installation over the *traditional* packaging for this subject is due to the small source code size (Table 5.2) and the efficiencies provided by Git checkout. The three *concise* packaging subjects using Git storage all show nearly equal median installation times compared to the *traditional* packaging. This empirical evidence seems to indicate that using Git provides advantages as noted in Section 3.2.3. Given these observations our answer to RQ 3 is that in general the per-version installation time is greater when using the *concise* packaging method.

When considering the total time for both download and installation the data in Table 4.2 shows that the two packaging methods are nearly equal. Subjects using the Git storage method, Dolibarr, YourContacts and MyMovieLibrary, all show an overall shorter total time required for the *concise* packaging than for the *traditional* packaging. Subjects with Zip storage generally evidence a longer total time for the

Table 5.2: *Source Code Collection Size Ranges*

Subject Name	Number of Versions	Source Code Collection Size Range
Dolibarr	30	25 MB → 48 MB
JavaMyCollab	22	82 MB → 139 MB
PHP Agenda	29	101 KB → 197 KB
PHP AddressBook	109	68 KB → 3.6 MB
Joomla	83	3.5 MB → 7.7 MB
YourContacts	47	1.9 MB → 2.1 MB
PHP Fusion	41	2.0 MB → 5.3 MB
MyMovieLibrary	20	687 KB → 804 KB

concise packaging than the *traditional* packaging. The exception to this is the largest of the Zip storage subjects, JavaMyCollab, which shows a shorter total time. This assessment of total time does not consider the statistical nature of these values. If we include standard deviations for both download and installation into the assessment, the total times for both *concise* and *traditional* overlap for the Git storage subjects, and for the PHP Fusion subject using Zip storage. The remaining Zip storage subjects are all generally longer in regard to total time required. Therefore we conclude that the answer to RQ 4 is that the two approaches are effectively equivalent when using Git storage and often require a small amount of additional time when a Zip storage method is used. The extra time required per version install is assessed in Table 5.3. The installation time cost per version is largest for the JavaMyCollab subject, which is directly related to the per-version size of this artifact shown in Table 5.2. When viewed from the perspective of total time cost, JavaMyCollab shows a small per-version improvement. While this one subject shows an improved total time cost per version, the other subjects show small increases in both installation and total time. Subjects with larger source code and numbers of versions appear to exhibit the greatest cost. Thus, for Zip storage subjects the number of versions appears to have

Table 5.3: *Median Installation Time Cost per Version of Concise Packaging*

Subject Name	Install Time Difference (secs)	Total Time Difference (secs)	Number of Versions	Install Time Cost (secs) per version	Total Time Cost (secs) per version
Dolibarr	591	-214	30	19.7	-7.1
JavaMyCollab	2239	-770	22	102	-35
PHP Agenda	36	31	29	1.24	1.07
PHP AddressBook	233	165	109	2.14	1.51
Joomla	1200	855	82	14.63	10.43
YourContacts	26	-53	47	0.55	-1.13
PHP Fusion	147	64	41	3.59	1.56
MyMovieLibrary	-5	-11	20	-0.25	-0.57

a sizable effect.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this research we studied the process of using a referential approach to the source code of a testing research subject. Downloading large subjects is difficult for users due to network factors and a means for improving the user's download experience is desirable. We developed the *concise* packaging approach that replaces explicit source code collections with web URL references to the source code providing a download-when-needed method. We empirically studied the approach and found that it can significantly reduce the overall package size, and likewise improve the subject download time that a user observes. Use of both Zip file and Git repository structures for storage of testing subjects was conducted. In all cases, the *concise* packaging method exhibits longer per-version installation time, but not to the degree where user experience is unacceptably longer. Considering the total download and install time that a user observes, the *concise* packaging method demonstrates an equivalent or slightly increased per-version installation time.

Given the results of this research, a case can be made to justify use the proposed

concise packaging method. Because downloads of large subjects have historically been problematic, the distribution-of-download mechanism offered by the *concise* approach provides a means to significantly reduce the size of an individual subject downloaded from the repository. This reduction allows researchers to obtain the basic information needed to reproduce an experiment faster. The added expense, in terms of installation time, that the approach can sometimes incur may be less of an issue to users than that encountered by the initial download time.

The *concise* approach also streamlines subject packaging efforts. In the *traditional* approach source code must be maintained and the potential exists for misplaced source code leading to out-of-order versions. Using the proposed *concise* approach the versions can be rearranged by simply changing the file contents of the *link* or *gitcommit* files.

The Git repository storage method also offers extensibility to test subjects. By referencing a public web-host repository, new versions could be easily referenced allowing research to be extended as the subject evolves. Using the *traditional* approach this is manually intensive and has rarely been done.

Ultimately, no packaging mechanism is optimal for every situation. If source code size is small or the number of versions few, the *concise* packaging method has little merit compared to the *traditional* method. Only when full package size is in excess of 1GB or if the subject is provided by a distributed version control mechanism are we able to fully leverage the power of the *concise* approach. However, smaller subjects would not exhibit a large installation performance loss due to using the *concise* method.

6.2 Future Work

This research limited itself to considering only two storage mechanisms, Git repositories and Zip archive file stores. Owing to the similarities between Git and Mercurial, it would be interesting to craft *concise* subjects using this alternative distributed version control system. It would also be interesting to package subjects using SVN using this approach. Both of these technologies are actively used by projects and offer capabilities similar to Git, thus the prospect of similar improvements in total experiment time may be achieved using these as well.

Considering the effectiveness of Git repositories shown by the study, a hybrid packaging approach could be created. In this approach the subject package would be prepared with a Git clone copy of the source code, and the versions would provide the *gitcommit* needed to select the desired version. This approach would yield a large reduction in package size, but would require little or no additional network activity to change versions. This would require defining new directories or redefining the purpose of one or more directories within the SIR shown in Section 2.1 to store the Git clone source code along with usage documentation. This method would also lack the extensibility offered by a true Git repository.

It is also conceivable that the *concise* packaging could be applied to facilitate the download of the test suites. Doing so would be a simple matter of hosting the suites on a server using one of the storage techniques we have discussed. The reference mechanism could then be employed within the **testplans** or **testplans.alt** directory sub-trees along with the appropriate scripting to automate retrieval and provisioning. For subjects where test scripts dominated package sizes, this packaging change would create smaller test artifact packages.

Bibliography

[1] James H. Andrews. Relevant empirical testing research: Challenges and responses. *SIGSOFT Softw. Eng. Notes*, 29(5):1–4, September 2004.

[2] Atlassian. BitBucket a web-based version control repository hosting service owned by atlassian, for source code and development projects that use either mercurial or git revision control systems. `https://www.bitbucket.com`.

[3] Boris Beizer. *Software Testing Techniques (2Nd Ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[4] CollabNet. Apache subversion revision control system. `https://subversion.apache.org`.

[5] curl command line tool and library for transferring data with URLs. `https://curl.haxx.se`.

[6] Defects4j: A database of real faults and experimental infrastructure to enable controlled experiments in software engineering. `https://github.com/rjust/defects4j`.

[7] Apache Derby: an open source relational database implemented in Java. `https://db.apache.org/derby`.

[8] Hyunsook Do, Sebastian G. Elbaum, and Gregg Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[9] Sebastian Elbaum. An experimental infrastructure for evaluating failure analysis techniques for released software, 2001. `http://www.cse.unl.edu/~elbaum/papers/workshops/wess01.pdf`.

[10] git-checkout: Switch branches or restore working tree files. `https://git-scm.com/docs/git-checkout`.

[11] git-clone: Clone a repository into a new directory. `https://git-scm.com/docs/git-clone`.

[12] Git: a distributed version control system. `https://git-scm.com/`.

[13] GitHub: A web-based hosting service for version control using git. `https://github.com`.

[14] GitLab: An open-source hosting service for version control using git. `https://about.gitlab.com`.

[15] Dick Grune. Concurrent versioning system. `http://www.nongnu.org/cvs`.

[16] Gzip: a compression utility designed to be a replacement for compress. `https://gzip.org`.

[17] M. Hammoudi, G. Rothermel, and P. Tonella. Why do record/replay tests of web applications break? In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 180–190, April 2016.

[18] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 191–200, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[19] ibugs: Bug repositories extracted from project history. `https://www.st.cs.uni-saarland.de/ibugs/`.

[20] Java Archive utility. `https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jar.html`.

[21] Redhat jboss enterprise application platform. `https://developers.redhat.com/products/eap/overview`.

[22] René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 437–440, New York, NY, USA, 2014. ACM.

[23] time - a simple time command, a utility to display execution time statistics of a linux command. `http://pubs.opengroup.org/onlinepubs/9699919799/utilities/time.html`.

[24] MariaDB open source relational database system. `https://mariadb.org/`.

[25] Mercurial source control management system. `https://mecurial-scm.org`.

[26] MySQL relational database system. `https://mysql.com`.

[27] National institute of standards SAMATE research subjects. `http://samate.nist.gov/SRD`.

[28] Selenium automated web testing library. https://www.seleniumhq.org.

[29] Software Artifact Infrastructure Repository. http://sir.unl.edu.

[30] List of documents citing the SIR. http://sir.unl.edu/portal/usage.php.

[31] Java Object Handbook. http://sir.unl.edu/content/java-overall.php.

[32] SourceForge an Open Source community resource dedicated to helping open source projects be as successful as possible. https://sourceforge.net.

[33] SPL2Go software product lines testing. http://spl2go.cs.ovgu.de/projects.

[34] GNU tar: an archive tool. https://www.gnu.org/software/tar/manual/tar.html.

[35] UNL Git: a GitLabs private repository hosted by the University of Nebraska - Information Technology Services. https://git.unl.edu.

[36] Uniform Resource Locator. https://www.w3.org/TR/url.

[37] Virtual environments and packages. https://docs.python.org/3/tutorial/venv.html.

[38] wget - a free utility for non-interactive download of files from the Web. https://www.gnu.org/software/wget/manual/wget.html.