International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:4, 2010

Quantitative Evaluation of Frameworks for Web Applications

Thirumalai Selvi, N. V. Balasubramanian, and P. Sheik Abdul Khader

***Abstract*—**An empirical study of web applications that use software frameworks is presented here. The analysis is based on two approaches. In the first, developers using such frameworks are required, based on their experience, to assign weights to parameters such as database connection. In the second approach, a performance testing tool, *OpenSTA*, is used to compute start time and other such measures. From such an analysis, it is concluded that open source software is superior to proprietary software. The motivation behind this research is to examine ways in which a quantitative assessment can be made of software in general and frameworks in particular. Concepts such as metrics and architectural styles are discussed along with previously published research.

***Keywords*—**Metrics, Frameworks, Performance Testing, Web Applications, Open Source.

I. Introduction

MEASUREMENT is fundamental to any branch of engineering. Software engineering does provide techniques for this. However, standards have been slow to emerge. The main reason for this is the rapid evolution that is taking place in producing software, in terms of both technology and methodology. The *building* paradigm of yesteryear is based on creating customized code for each application. It has been replaced by the *assembly* paradigm. Existing *components* are reused in this approach where they are available, and new ones created if necessary. Measurement takes place at various stages in the software development life cycle. Terms such as *harvesting* time are used to denote this aspect of measurement. Reuse of architectural styles, such as data flow, call & return, repository, and layered approach, has always prevailed. In the new assembly paradigm, partial applications called *frameworks* are popular on account of reduced development effort and increased software quality. *Open Source* software has the same two advantages. The subsequent sections describe metrics, architectural styles & frameworks, performance testing, OpenSTA, MOODLE Framework, empirical study, and conclude.

Thirumalai Selvi is a PhD student in the Department of Computer Science, Mother Teresa University for Women, Kodaikkanal, Tamil Nadu, India.

N. V. Balasubramanian is Professor of Computer Science & Engineering, RMK Engineering College, Kavaraipettai, Tamil Nadu, India 601026 (phone: 91-44-28113956, 91-9841046961; email: laksbala@dataone.in).

P. Sheik Abdul Khader is Professor & Head of Computer Applications, B. S. Abdur Rehman University, Vandalur, Chennai, India.

II. Metrics

A. Problem and Solution Oriented Metrics

Requirements engineering precedes design, coding, and testing. If measurements can be done at this early stage of development, planning is greatly enhanced. Albrecht proposed *Function Point* in 1979[1], and there is an International Function Point User Group (IFPUG) to regulate metrics based on this approach. IFPUG holds conferences, workshops, and certifies professionals to carry out the measurement task. This approach uses the requirements document for computation. Inputs, outputs, inquiries, interfaces and files are weighted based on their complexity. An adjustment factor is then applied based on reuse, distribution, etc. to the raw values to arrive at the final numerical figure for the software. *Use Cases* are today's de-facto descriptions of customer' requirements. So, *Use Cases* too can be used to compute metrics at early stages of software development. As we harvest metrics at the time of describing the problem, they are called problem oriented metrics. Lines of Code (LOC) have, from the inception of software metrics, played an important role in measurement, particularly in the days when procedural programming languages dominated the software scenario. Halstead proposed Computer Science Metrics in 1972[2] based on *operands* and *operators* in programs. The disadvantage with such metrics is the harvesting time; numerical figures can not be derived till after the coding is complete. Nevertheless, software companies use them to reflect on the past in order to project into the future in a more professional manner.

B. Object Oriented Metrics

Most software these days follow the Object Oriented (**OO**) paradigm. Chidamber and Kemerer (**CK**) proposed metrics for **OO** in 1994[3], and they are still widely followed. For calculating complexity, they used Cyclomatic Complexity proposed by McCabe in 1976[4]. **CK** came out with a suite of six metrics:

- Weighted Methods per Class (**WMC**)
- Response for a Class (**RFC**)
- Lack of Cohesion (**LCOM**)
- Coupling Between Object Classes (**CBO**)
- Depth of Inheritance Tree (**DIT**)
- Number of Children (**NOC**)

For weighting, cyclomatic complexity is used.

C. Component Based Metrics

Since components are basically a set of collaborating classes and a set of interfaces (those being classes themselves), it is justifiable to extend **CK** metrics for Component Based Software. A proposal to grade relationship between classes such as dependency, association, aggregation, composition, and generalization/specialization has been made. (This is akin to weightings given to inputs, outputs, inquiries, interfaces, and files in the *Function Point* Method.) However, some authors feel that additional metrics such as reuse, packing density, and criticality are needed to supplement the above.

Dolado[5] analyzed 46 projects and used Neural Networks for computing metrics. But the technology for software development at that time was fourth generation languages such as Application Language Liberators. Dolado used Mark II version of *Function Point*. Today's Component Based Software Development (**CBSD**) is far more sophisticated for using Dolado's approach. Often, *Frameworks* are used in conjunction with components. Cho and Kim[6] use a banking case study to illustrate how static and dynamic complexities of components can be computed. The values they use for dependency, association, generalization/specialization, aggregation, and composition are 2, 4, 6, 8, and 10 respectively. No explanation is given in their paper for arriving at these values. They also propose new measures for customizability and reusability. The Common Software Measurement International Consortium (**COSMIC**) has come out with a measurement method for functional size, with some assumptions. Firstly, a layered architectural style is the basis for component assembly; no component can straddle two layers. The metric is based on data movement, and ignores data manipulation. In addition to *Entries* and *Exits* of data to and from components, there are also *Reads* and *Writes* from and to persistent storage. We simply sum up the *Entries*, *Exits*, *Reads*, and *Writes* to arrive at the size. Event-driven paradigm is assumed for programming. An event triggering a functional process is considered an *Entry*, and may have only one data attribute (not a group). If input to a functional process comprises more than one data group, identify each data group as one *Entry*. Do likewise for *Exits*, *Reads*, and *Writes*. Any message from a functional process to the user retrieving data shall not be counted as an *Exit*. A requirement to delete a data group from persistent storage shall be measured as a single *Write*.

D. Web Metrics

Pioneering work, using empirical methods, has been done by Emilia Mendes, et al. [7][8][9] after analyzing several web hypermedia projects. They use three techniques, namely, Expert Judgment, Algorithmic Models, and Machine Learning. Essentially, the first technique has been used here for assigning weights to pages, links, database connections, multimedia contents, and so on. That is, experts make a subjective assignment of numeric values to these various factors in much the same manner as is done in the *Function Point* method. These were presented at the Second Functional Sizing Summit 2007[10] by the authors. This work was further developed, to include frameworks, and published in **IJWSP**[11]. However, performance issues were not included in this paper. While the above mentioned research work of all authors has focused on technological aspects, some authors [12] have taken a management oriented approach using essentially a questionnaire based survey for user satisfaction and such feedback. This will not be pursued in this paper.

III. Architectural Styles and Frameworks

A baseline architecture is an essential starting point for software development, once the requirements have been established. The *Call & Return* architecture was appropriate in the days when mainframe computers and procedural programming dominated the computing scene. With Unix, a new style came to be used, namely, the *Data Flow* architecture. *Filters* are smaller programs written in 'C', and they are put together using *Pipes* which are essentially Shell programs; this is how larger programs were built from a set of smaller programs. Even today, many image processing softwares use this architectural style. Certain applications are dominated by a *Repository*, and clients either retrieve or manipulate data in the *Repository*. **OSI** came out with a seven-*Layer* architecture, and soon such an approach became widespread, particularly in web applications. Typically, there is a back-end *Database Layer* that interfaces with the *Application Layer*. The customer uses a web browser (called a *Thin Client Layer*), and accesses the application via a *Web Server Layer*. The essential thing about all these architectural styles is that they are abstractions. Hence, it is tricky to incorporate measurements in them. *Frameworks*, on the other hand, are concrete; they do follow some architectural style and incorporate some design patterns. *Frameworks* are customized by a combination of *parameters* and *hook methods*. *Frameworks* allow *components* to be added as well as replaced. Thus the new software development paradigm is just like automobile *assembly*. Some consider even operating systems and database management systems to be frameworks. Microsoft's *Framework* uses *Active-X* components using the Distributed Component Object Model (**DCOM**). The fundamental component of **CORBA**, another *Framework*, is the Object Request Broker (**ORB**) whose task is to facilitate communication between objects. Given an Inter-operable Object Reference (**IOR**), the **ORB** is able to locate target objects and transmit data to and from remote method invocations. The interface to a **CORBA** object is specified using **CORBA**'s Interface Definition Language (**IDL**). An **IDL** compiler translates the **IDL** definition into an application programming language (C++, Java, Tcl/Tk) generating **IDL** stubs and skeletons that respectively provide client-side and server-side proxies. Microsoft also provides the Active Server Pages (**ASP**) *Framework* for web applications. **PHP** is a popular *Open Source* server-side scripting language for web

applications, competing with Perl. Using **PHP**, several *Open Source Frameworks* (**JOOMLA**, **TYPO3**, **MOODLE**) have been developed and continually upgraded for web based applications. These cover Content Management, Course Management, and the like. A very useful software is **XAMPP** which bundles Apache, MySQL, PHP, and Perl. Measurement is facilitated by using existing features (such as **DBCheck** in **TYPO3**) and writing additional **PHP** code to gather static data (number of pages) and dynamic data (response time). For a variety of reasons, *Open Source* has penetrated every aspect of computing, and the trend is expected to continue. Since source code is available for *Open Source* software, *Glass Box Components* are available for extension and substitution. Additional code can be incorporated for measurement purposes in these *Glass Box Components*.

IV. Performance Testing

Web applications depend on quick response to *visitor's* requests. Often times, the software has to be loaded in the server, before a service can begin. Since there are repeat requests for the same service, *cache* and other techniques (such as *proxy*) are used to speed up the process. Rarely, we find that measurements, such as response times, are provided by available *Frameworks*. We therefore need to use measurement tools to intercept service requests, and obtain the relevant information. An *Open Source* tool for performance measurement is *OpenSTA*. After web applications were developed, they were run along with *OpenSTA*. It was thus feasible to compare various projects on performance factors. *OpenSTA* creates virtual users to load the system, and thus simulate a live environment. This tool has the following features:

- *Test Commander* – The central control application for testing using *OpenSTA*,
- *Name Server* – **CORBA** background process to let *OpenSTA* components find each other and communicate,
- *Script Modeler* – Applications where scripts are recorded and developed,
- *HTTP Gateway* – Proxy like background process that performs recording,
- *Test Executer* – Background process that actually executes the test,
- *Web Relay Demon* – Uses **XML RPC** to get over **CORBA** limitations on the Internet,
- *Repository* – Where all test scripts, configurations, and results are maintained,
- *Test Manager* – Background process that manages *Test Executer*,
- *Task Group Executer* – Process that runs other tasks.

Whilst testing web applications, the starting time for *Open Source* was found to be smaller as compared to *Proprietary* software. These results are compared in a later section in the paper. (See Appendix for GUI Interfaces.)

V. Moodle Framework

A. PHP (Personal Hypertext Processor)

As **MOODLE** is developed in **PHP**, an introduction is presented here. **PHP** is competing with Perl for building high performance dynamic web sites. It is a server side scripting language, and uses a *Parser* for dynamically interpreting scripts containing both **HTML** and **PHP** as shown in the Fig.1 below. The *Zend* engine enhances the performance of **PHP** based web sites. Software can be developed using Object Oriented (**OO**) paradigm, including **SOAP**. Exceptions handling is also available in **PHP** for managing error conditions during operation. Excellent support is provided for **MySQL** database management system, as well as **SQLite**. Increasingly, a template based approach is being used to create web sites quickly with **PHP**. Since the source code is available, one can customize a specific web application with relative ease. Interactive Development Environments (**IDE**) are the norm for coding and scripting these days. Builders of **TYPO3** have created one such called **FLOW3**. This adds an extra layer to the Framework for customization purposes. **FLOW3** supports *Aspect Oriented* Programming as well as *Agile* Software Process.

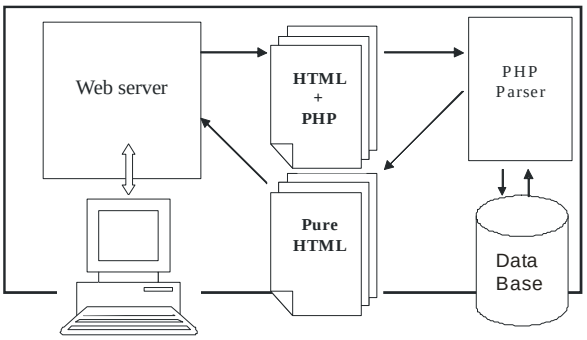

Fig. 1 Interaction between Web Server and PHP Parser

B. MOODLE

MOODLE stands for Modular Object Oriented Dynamic Learning Environment. It is an *Open Source Framework* for course management. It has an excellent database organization, supported by **ADODB** library. The components of MOODLE are called *activity modules*, and are useful in the expansion of the **ELF** (which is currently not activity based). Since the source code is at our disposal, new functionality can be added to **MOODLE**.

It supports several operating systems like Linux, Windows, and Mac OS-X. Several **MOODLE** sites can be interlinked to each other. **MySQL** provides database backup and recovery facilities. A software house, *Tenth Planet Technologies Limited*, Chennai, India, has specialized in using this *Framework* to support school administration, and willingly supplied us data for our empirical study. Software architecture of **MOODLE** and database architecture follows in Fig. 2, 3.

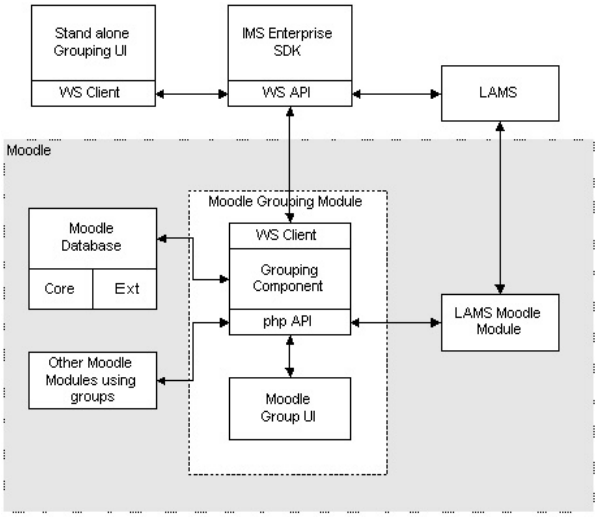

Fig. 2 Architecture Diagram for MOODLE

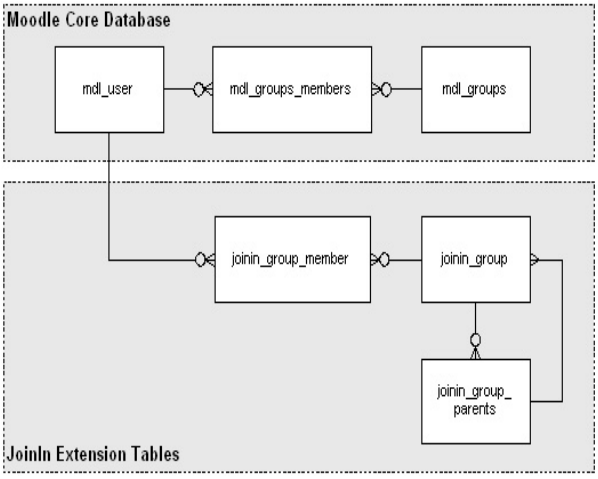

Fig. 3 MOODLE Core Database

VI. Empirical Study

A. Scope of Empirical Study

In addition to data provided by *Tenth Planet Technologies Limited* on substantial projects using **MOODLE**, additional projects for the empirical study were carried out by our graduate students, many of whom had good familiarity with Microsoft **ASP** *Framework*. They were given training in **PHP** so that they can carry out the same project in both **ASP** and **PHP**. The analysis is based on two approaches. In the first, developers were required to assign weights to parameters such as database connection based on their experience. In the second approach, *OpenSTA* was used to compute start time and other such measures.

B. Web Metrics

For each project, the developers were asked to give weights to the following factors:

- *Platform Neutral*
- *Creating Record Set*
- *Database Connection*
- *Email Objects*
- *Cascading Style Sheet*
- *Content (Multimedia)*
- *Scripting Language*
- *Audio and Video Files*

ASP developers could not assign weights to certain factors like *Platform Neutral*. For each project, its size was computed by multiplying, for each item, the number of occurrences and the weight assigned for the item, and summating individual item values. The weights assigned by the experts are given in Fig. 4 below for **ASP**, **PHP**, **MOODLE**.

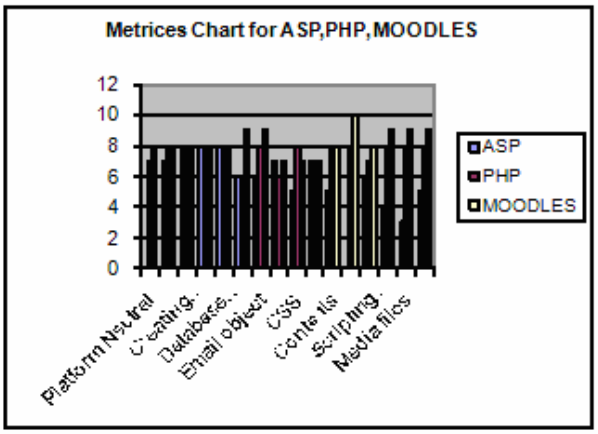

Fig. 4 Metrics Chart for ASP, PHP, MOODLE

C. Measuring Size Metrics

The complexity and length metrics, except reuse, were measured for the various projects using the **COSMIC** method. The counting rules corresponding to each *Entry*, *Exit*, *Read*, and *Write* are as follows: each **HREF** tag counted as one *Entry* plus one *Read* plus one *Exit*. By pressing a link, the user sends an *Entry* to an application that *Reads* the data from the web server, and shows the contents to the user (i.e., *Exit*). The following measurements were taken:

- *Length* – Page counts, program counts, total page allocation, total embedded code length,
- *Complexity* – Connectivity (internal links), density of connectivity (connectivity / pages), total page complexity, number of different types of media, media density (media / pages).

A simple figure for comparison is the ratio of *Lines Of Code* (**LOC**) divided by *Function Points* (**FP**). Here is a ratio that uses a solution oriented metric (**LOC**) and normalizes it using a problem oriented metric (**FP**). **PHP** programs have a consistently smaller ratio as compared to **ASP** (*Proprietary*

Framework) for the same project, although the margin is too small to be tabulated.

D. Performance Metrics

Two screen shots of the tool *OpenSTA* are given in the appendix (Fig. 6, 7). These give a flavor for the tool. Using *OpenSTA* software, starter times were measured for the same project using different implementation methods, and are tabulated below in Fig. 5.

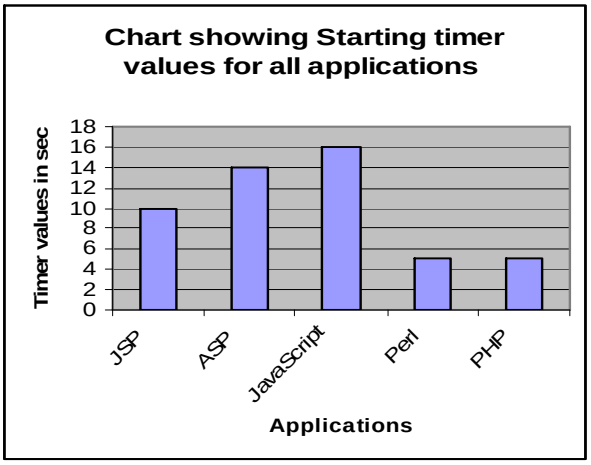

Fig. 5 Comparison of Starter Time Performance

It is seen that proprietary software takes longer to start as compared with *Open Source* software.

VII. Conclusion

This paper has presented the studies of researchers in arriving at a quantitative method of evaluating software in general and web engineering in particular. While **COSMIC** method is expected to replace the **FP** method for problem oriented metrics in web engineering, it is the *Use Case* based method that will form the basis for general software. For solution oriented metrics, a *Framework* based analysis is advocated by the authors. In fact, the proposal is to build measurement instrumentation into such *Frameworks*. A Rich Internet Application (**RIA**) *Framework* is being designed with this in mind, using **PHP**. For implementation, this new *Framework* will use **FLOW3** and event-driven programming paradigm. This new *Framework* will also interface with *OpenSTA*. The objective is to integrate measurement as part of software development, and not as an after thought.

Acknowledgment

We thank the company *Tenth Planet Technologies Limited*, Chennai, and its staff for providing us with the data on software projects, including those that were not readily available but were prescribed by us.

References

[1] A. J. Albrecht and J. E. Gaffney Jr., "Software Function, Source Lines of Code, and Development Effort Prediction," *IEEE Transactions on Software Engineering*, vol. 9, no. 6, November 1983.
[2] M. H. Halstead, "Elements of Software Science," *Elsevier*, New York, 1977.
[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, June 1994.
[4] T. McCabe, "A Software Complexity Measure," *IEEE Transactions on Software Engineering*, vol.2, no.12, December 1976.
[5] J. J. Dolado, "A Validation of the Component Based Method for Software Size Estimation," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, October 2000.
[6] E. S. Cho, M. S. Kim, and S. D. Kim, "Component Metrics to Measure Component Quality," *The 8th Asia-Pacific Software Engineering Conference (Macau)*, 2001.
[7] E. Mendes, N. Mosley, and S. Counsell, "Web Metrics – Estimating Design and Authoring Effort," *IEEE Multimedia's Special Issue on Web Engineering*, 2001.
[8] E. Mendes, N. Mosley, and I. Watson, "A Comparison of Case-based Reasoning Approach to Web Hypermedia Project Cost Estimation," *Proceedings of the 11th International World Wide Web Conference (Hawaii)*, 2002.
[9] E. Mendes, N. Mosley, and S. Counsell, "Investigating Early Web Size Measures for Web Costimation," *Proceedings of EASE Conference (Keele University)*, 2003.
[10] R. Thirumalai Selvi, "Metrics in Component Based Software Engineering," *Second International Functional Sizing Summit (IFPUG)*, Vancouver, Canada, April 2007.
[11] R. Thirumalai Selvi, N. V. Balasubramanian, and P. Sheik Abdul Khader, "Framework and Architectural Style Metrics for Component Based Software Engineering," *International Journal of Web Services Practices*, vol. 3, no. 1-2, 2008.
[12] Aaron Don M. Africa, "Quantitative Evaluation of Open Source Content Management Systems," *IEEE Multidisciplinary Engineering Education Magazine*, vol. 3, no. 2, June 2008.
[13] meetinguniverse.com
[14] nambco.com
[15] showcase.rhytha.org/vyabr
[16] cdmainteractive.com
[17] thejo-engg.com
[18] denvik.in
[19] chipkidz.com
[20] typo3.org
[21] flow3.typo3.org

Thirumalai Selvi completed her MS from Madurai Kamaraj University and did her MPhil from M. S. University. She has more than 13 years of teaching experience, and is currently Head of the Department of Computer Science at Government Arts and Science College, Tiruttani, India. She has several publications in the area of Software Engineering.

N. V. Balasubramanian completed his doctorate at Liverpool University, U.K., and has 14 years of industrial and 30 years of academic experience, including 14 years as the Founding Head of the Department of Computer Science at City University of Hong Kong. He was active in Hong Kong in professional bodies such as IEEE, Hong Kong Institution of Engineers, Hong Kong Association of Computer Education. Between 1987 and 1989, he served as Region 10 Director, IEEE Computer Society, covering Australia, China, India and Japan. He has published several papers in the area of Software Engineering and has been invited speaker at conferences.

P. Sheik Abdul Khader completed his doctorate at Anna University, Chennai, India, and has been working in several institutions as an academic before joining B. S. Abdur Rehman University, Vandalur, Chennai, India, as Professor and Head of Department of Computer Applications. He has more than 25 years of academic experience. His areas of interest include Computer Networks, Software Engineering, and Image Processing. He is an approved PhD supervisor at several universities, including Sathyabama University and Mother Teresa Women's University. He has published several papers in international journals and conference proceedings.

APPENDIX

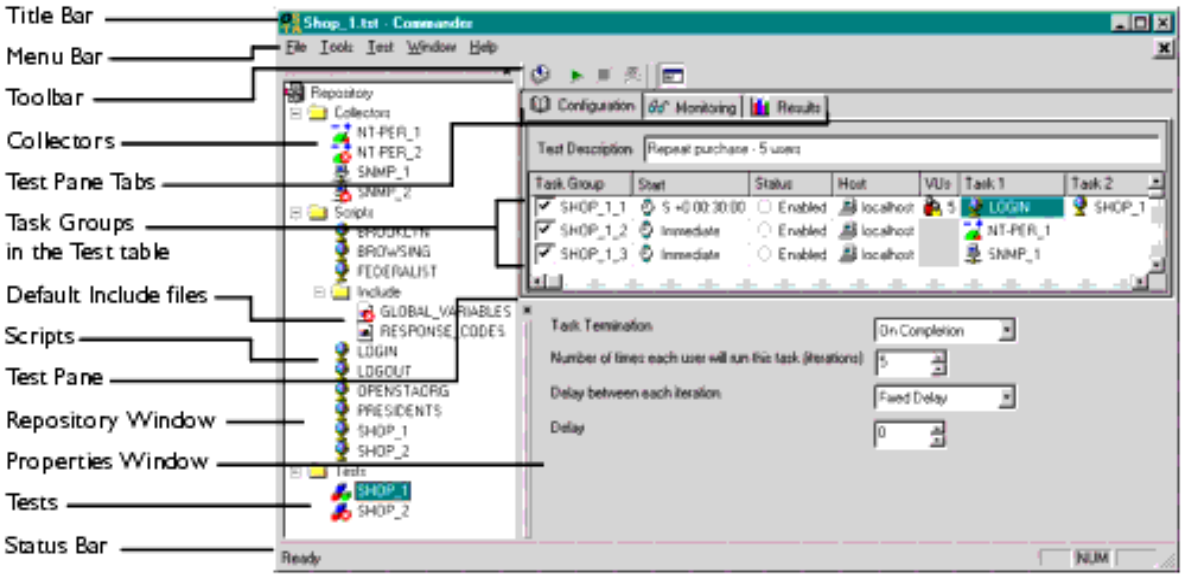

Fig. 6 Test Commander Interface of OpenSTA

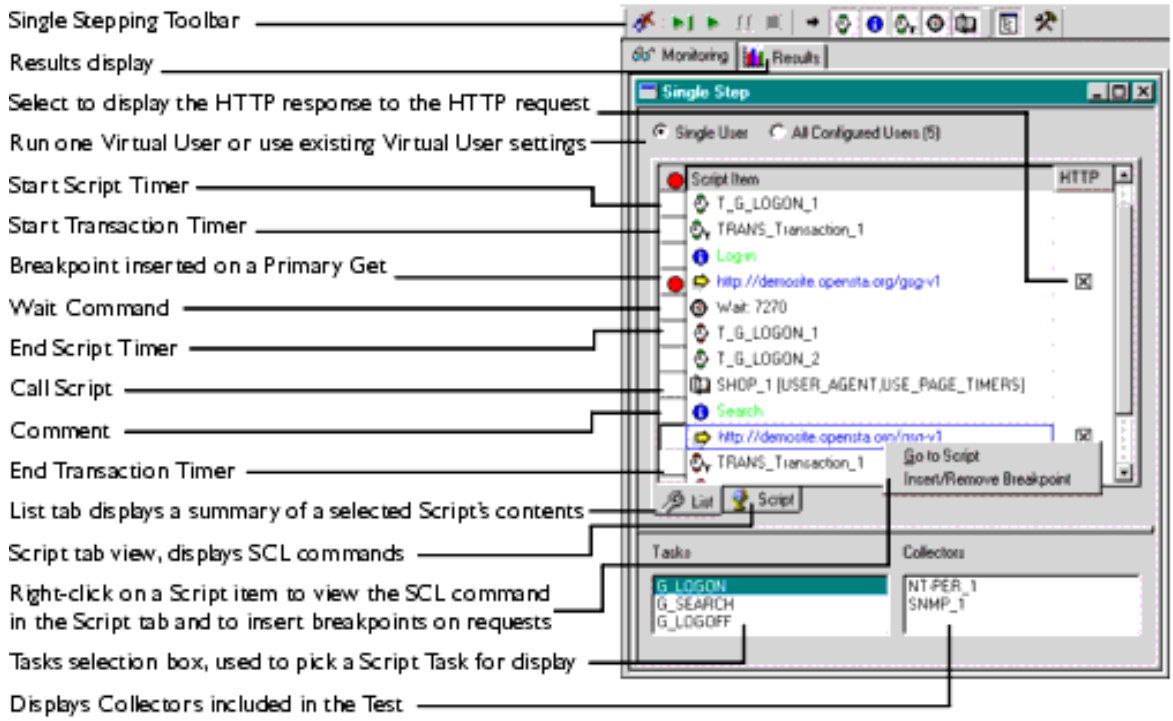

Fig. 7 Single Stepping Interface of OpenSTA