Utilizing Operational Profile in Refactoring Large Scale Legacy Systems

A. Güneş Koru, Li Ma, and Zhao Li

Southern Methodist University
Computer Science and Engineering Department
Dallas, Texas, 75275-0122, USA
{gkoru,lima,lizhao}@engr.smu.edu

Abstract

Applying refactoring in a large-scale legacy system can potentially decrease future maintenance costs and prepare the system to accommodate future changes more easily. However, in such systems, it is desirable to identify the system components that will be the subject of the refactoring effort before making a significant investment. The operational profile of the system could be a useful input in focusing the effort, in addition to other guidance such as expertise, domain knowledge, static measurements of the system, and so on. In this position paper, we suggest using a markov chain model which could take operational profile into account and support engineers in identifying the system components of greatest interest for refactoring.
Keywords: Refactoring, large-scale systems, legacy systems, operational profile, markov chains.

1. Introduction

Software and information systems evolve over time. As also stressed by Lehman et. al. in [4], the evolution is not only because of our insufficient development techniques or processes, but also because of the nature of those systems. Therefore, as in the past, dealing with this intrinsic evolution and maintaining legacy systems will be parts of engineers' jobs.

Refactoring techniques enable engineers to restructure object oriented systems so that their several qualities, such as maintainability, changeability, and so on, could be improved following a disciplined approach. However, despite the new techniques, when the system in hand is a large-scale legacy one, applying refactoring to the whole system can be a daunting problem. Often, a common remedy is to focus on the system components that are more important so that the resources can be used in the most efficient manner. In the past, similar problems were encountered while deciding where to focus the quality or reliability improvement activities, such as testing and inspection, in large-scale systems. Utilizing operational profile [6] to guide these activities was reported to be successful (when used along with other risk identification techniques [8]).

Although a substantial amount of study about how to do refactoring has been performed in a short time recently, there is a lack of research on where to apply refactoring, as stressed in [5], especially in the case of large-scale systems.

In refactoring, decisions about which system components to focus on can generally be based on:

- Expertise: Engineers can recognize some system components that are better candidates for refactoring by looking at the system architecture and design.
- Domain Knowledge: An understanding of the problem domain and requirements can help prioritizing the candidates for refactoring.
- Static Measurements (metrics): The system under study can be measured using several complexity metrics and some threshold values can be used to identify the candidates for refactoring. Related research can be found in [7].
- Operational Profile: The system components with higher usage probabilities can be considered as better candidates.

Operational profile should be utilized in concert with the other means listed above in one of the two following ways: If it is used as the primary decision mean, the system components with highest usage probabilities can be picked for refactoring. If it is utilized as a supporting mean, it could be

used to identify the system components that closely collaborate with the components priorly identified by other means.

In this position paper, we suggest using a markov chain model that could enable engineers to utilize operational profile information in refactoring large scale systems. We start by describing the model. After that we explain our model giving a simple example. Then, we describe how we applied our ideas using the web logs of an open source development site. Finally, we conclude with a section about conclusions and future directions.

2. The Model

The model we suggest to support refactoring is basically a markov chain. A markov chain consists of a set of states and state transitions with assigned probabilities. An in depth discussion of markov chains can be found in [1]. In this model, a state represents a component of the system considered for refactoring and a state transition represents a usage relationship between two components. We also introduce two additional states, the start state s and the termination state t. All of the states in the model will be transient states, except t which is an absorbent state.

Let us represent the markov chain with a matrix P. As also discussed in [3], since we have a finite number of transient states, P can be written as:

$$P = \begin{bmatrix} K & 0 \\ L & Q \end{bmatrix}$$

by putting t to the first order in rows and columns. Q is obtained by deleting the row and column in P that correspond to the termination state t. L can be defined accordingly. For any $n \in \mathbb{N}$,

$$P^n = \begin{bmatrix} 1 & 0 \\ L_n & Q^n \end{bmatrix}$$

From here, we can calculate the matrix R, where $R(i,j)$ is the expected number of visits from i to j if the system is not changed and kept in operation forever.

$$R = \sum_{n=0}^{\infty} P^n = \begin{bmatrix} \infty & 0 \\ \sum L_n & \sum Q^n \end{bmatrix}$$

It can be written that

$$S = \sum_{n=0}^{\infty} Q^n = I + Q + Q^2 + ...$$

$$SQ = QS = Q + Q^2 + ... = S - I$$

$$(I - Q)S = I$$

and therefore,

$$S = (I - Q)^{-1}.$$

S provides us with $R(i,j)$s of our interest, which are the expected number of visits among transient states.

Once we obtain R, we can also calculate the matrix F, where $F(i,j)$ represents the probability of starting from i and ever reaching j, by using

$$F(i,j) = \frac{R(i,j)}{R(j,j)}$$

and

$$F(j,j) = 1 - \frac{1}{R(j,j)} \quad .$$

The above procedure is detailed in [3] and a slight variant of it was used by Cheung in a different context to predict system reliability from the reliability of its components [2].

It should be noted that $R(i,j)$ and $F(i,j)$ values take all possible paths from i to j into account.

3. Using the Model

Let us assume state (or system component) i is the initial focus of the refactoring effort. i can be the starting state s if the operational profile is the primary factor effecting the decisions about refactoring. Alternatively, i might be different from s and it might be decided as an initial point by other means, for example, expert decisions or static measurements.

We select n largest probability values among the $F(i,j)$ values in the ith row (outgoing probabilities from i) and the $F(j,i)$ values in the ith column (incoming probabilities to i) of the matrix F. For each selected $F(i,j)$ or $F(j,i)$, we include state j in our set C, which is the set of closely collaborating system components identified according to the operational profile. The value of n can be adjusted according to the desired value of $|C|$.

4. Preliminary Studies

Before directly applying these ideas to restructuring object oriented systems, we conducted two preliminary studies. We constructed markov chains that represent the operational profile of web sites whose web logs were already available to us. The web logs were kept by Apache web server. We represented each hit detected in the web log as a state transition from the referrer web page to the target web page. The referrer and target web pages were represented as states. When the number of in-links to a state i was smaller than the number of out-links from it, we introduced a state transition from the start state s to i. When the number of out-links from state i was less than the number of in-links to it, we introduced a state transition from i to termination state t. This also alleviated the limitations in the logging capabilities of Apache server, for example not recording page

	t	s	/cse/index.html	/cse/program.html	/cse/programs/bscs.html	/cse/coursePlan.html	/index.html	/cse/faculty.html	/cse/programs/mscs.html	/cse/courseDesc.html	/cse/8364/index.html	/cse/programs/msse.html	/cse/3353/index.html	/cse/programs/msce.html	/cse/Pfi ster/index.html	/cse/8371/index.html	/cse/7350/index.html	/cse/7312/index.html
t	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
s	1.00	.00	.70	.14	.02	.09	.42	.25	.03	.11	.07	.03	.02	.02	.06	.04	.06	.02
/cse/index.html	1.00	.00	.00	.20	.03	.12	.00	.33	.05	.15	.00	.04	.00	.03	.00	.00	.00	.00
/cse/program.html	1.00	.00	.00	.00	.16	.00	.00	.00	.25	.00	.00	.20	.00	.15	.00	.00	.00	.00
/cse/programs/bscs.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/coursePlan.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/index.html	1.00	.00	1.00	.20	.03	.12	.00	.33	.05	.15	.00	.04	.00	.03	.00	.00	.00	.00
/cse/faculty.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/programs/mscs.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/courseDesc.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/8364/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/programs/msse.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/3353/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/programs/msce.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/Pfi ster/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/8371/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/7350/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
/cse/7312/index.html	1.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00

Figure 1. F matrix for SMU CSE department web logs.

accesses made via browser's back button or not recording terminations in the case of exiting from the browser.

In the first study, we analyzed the web logs that belong to the Computer Science and Engineering department at SMU. The data was collected over a period of one month and included 4802 hits (web page accesses). In creating P matrix from web hits, we only considered the hit types that took place more than 50 times. Therefore, our state transitions were created according to the frequency of those hit types. This abstraction also enabled us to show our resulting F matrix that can be seen in Figure 1 more easily because it only included 18 states.

The second line in Figure 1 shows that the web pages that have a higher probability of being accessed from the start state compared to others are /cse/index.html, /index.html (engineering school's index page), /cse/faculty.html, and /cse/program.html in decreasing probability order.

If one decided to make changes on /cse/program.html, the matrix would show that the highly collaborating four web pages would be /cse/programs/mscs.html, /cse/8364/index.html, /cse/index.html, and /index.html.

As the second study, we analyzed the web logs made available to us by the developers of KDE which is a desktop for Linux systems (http://www.kde.org). The KDE web log was collected in one day, and it included 600,000 hits. When creating the markov chain, we followed the same approach followed for SMU/CSE logs, except this time we did not set any threshold value on the number of hits while determining hit types to be considered as state transitions. Therefore, we obtained a large F matrix with 1211 states. By extending our programs with additional functionality, we automated the job of finding closely collaborating pages of any web page.

5. Future Directions

When applying the concepts presented here to refactoring large scale legacy systems, the classes in the system could be considered as states and method calls could be considered as state transitions in the markov models.

The first important task is to obtain execution logs kept over a representative amount of running time. An open source Apache sub-project called Log4J has already developed capabilities to keep run time logs (http://jakarta.apache.org/log4j). Therefore, it can be useful for our purposes.

Secondly, in the case of very large sparse matrices, using conventional spreadsheet and statistical tools can make it difficult to import the matrices or to perform matrix operations such as finding the inverse matrix. Therefore, alter-

native and more efficient ways to represent sparse matrices and perform matrix operations should be employed.

Acknowledgments

This research is supported in part by NSF grants 9733588 and 0204345, and THECB/ATP grant 003613-0030-2001.

References

[1] U. N. Bhat and G. K. Miller. *Elements of Applied Stochastic Processes, Third Edition*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2002.

[2] R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, SE-6(2):118–125, March 1980.

[3] E. Cinlar. *Introduction to Stochastic Processes*. Prentice Hall, New Jersey, 1975.

[4] M. Lehman and L. A. Belady. *Program Evolution: Processes of Software Change*. Academic PRESS, INC.

[5] T. Mens, S. Demeyer, B. D. Bois, H. Stenten, and P. V. Gorp. Refactoring: Current research and future trends. In *Proceedings of Third Workshop on Language Descriptions, Tools and Applications*, pages 120–130, Warsaw, Poland, 2003.

[6] J. D. Musa. Operational profiles in software reliability engineering. *IEEE Software*, 10(2):14–32, Mar. 1993.

[7] F. Simon, F. Steinbruckner, and C. Lewerentz. Metrics based refactoring. In *Proceedings of 5th European Conference on Software Maintenance and Reengineering*, pages 30–38, Lisbon, Portugal, 2001.

[8] J. Tian. Quality assurance alternatives and techniques: A defect-based survey and analysis. *Software Quality Professional*, 3(3):6–18, June 2001.