International Journal Of Engineering Research & Management Technology ISSN: 2348-4039

Email: editor@ijermt.org July- 2014 Volume 1, Issue-4 www.ijermt.org

Performance Analysis to Changeability of Measuring Software Components

Anuradha Panjeta, Prof.Ajay Kumar
Computer Science Engineering
Kurukshetra University
Kurukshetra

Abstract—
Software may be defined as asset of programmes which contains set of instructions to form software. Software maintenance is that part of software which maintenance every part of it and reduce problems. Maintainability has four components, namely, analysability, testability, stability, and changeability. We identify the quantum of indexes which belongs to particular project.

Keywords— Object-oriented (OO); Line of code (LOC); Source Line of Code (SLOC).

I INTRODUCTION

The software maintenance is a major cost concern. The maintainability of a system seems to have much influence on the ease or difficulty to implement changes. A consensus has emerged that the maintainability of a software system is dependent on its design in the procedural paradigm as well as in the object-oriented (OO) paradigm. Maintainability has four components, namely, analysability, testability, stability, and changeability. In application areas like telecommunications, software systems are evolving constantly. There are organizations, which do not develop the software they operate, but purchase it. These organizations are not directly interested in testability or diagnosis, but in the software's ability to sustain an on-going flow of changes. In this research, the focus will be on that single aspect of maintainability, i.e., changeability. One way of assessing changeability is to assess the impact of changes earlier; systems were developed by using structured approach, which was very successful, but only for simple applications. Then came the object-oriented (OO) approach, which is based upon polymorphism, encapsulation and inheritance.

The use of object-oriented (OO) technology for developing software has become quite widespread. Researchers assert that OO practice assures good quality software, that is, particularly software that is easy to maintain, reuse, and extend. The assessment of the changeability of software systems is of major concern for buyers of large systems found in fast moving domains. Designing and maintaining systems in a dynamic contemporary environment requires a rethinking of how systems provide value to stakeholders over time. Developing either classically or changeable robust systems are approaches to promoting value sustainment. But in definition ambiguity across system domains has resulted in an inability to specify, design, and verify to utilities that promote value sustainment.

Changeability, in contrast, means the ability of an operation system to alter autonomously the configuration to meet new, previously unknown demands e. g. from the market. Changeability is then the ability to realize new states of the in, out and throughput. Additionally, the system’s reconfiguration has to be realized as quickly as the environmental changes. Therefore, to be changeable, the speed of adaptation is important. Software applications are not only single components systems, they are made up by collection of components. These components should be developed in such a way that the code written once can be reused many times by doing some changes in the code , which can reduce the development time ,cost and effort. Sometimes changes introduce a new fault that degrades the functionality of the system. If proposed a large change increment is, it will introduce many new faults that will limit the useful change delivered in the new version of the system. The rate of change of the system is therefore governed by organization decision making process.

The claim these 'laws' are invariant and widely applicable. Lehman and Beady examined the growth and evolution of a number of large software systems. The **first law** states that system maintenance is an inevitable process .As the systems environment changes, new requirements emerges and the system must be modified When the modified system is re-introduced into the environment ,this promotes more environmental changes so the evolution process recycles.

The **second law** states that, as a system is changed its structure is degraded. The only way to avoid this happening is to invest in preventative maintenance where you spend time improving the software structure without adding to its functionality. The changes to the source code follow a well defined process. New software releases or software updates are customer's deliveries that contain new functionality (features) and fixes or improvements to the code. The basic properties of change that are recorded by a simplest change management system include ownership (who made change), object that is changed (product, subsystems, module, file, set of lines) and time when the change was performed.

II Need to measure change

Measuring change is important for many reasons:

- It can help managers understand the direction that the software product is taking and help evaluate the work done by different members of development team.
- Metises can also help developers to improve their programming and design practices and to forecast where change and testing needs to occur. Also it can help researchers trying to understand how software evolves.

In order to measure changes it is not uncommon to measure the systems at two points: in time before and after event or time interval and then compute the measurements.

Metrics are needed that can measure change object and that can yield meaningful results. We can define change metrics as metric that can be used to measure how much a software system has been modified between two versions. The set of change requests from system users, management or customers are assessed to see how much of the system is affected by the change and how much it might cost to implement the change. If accepted the proposed changes, the system is planned for a new release. During release planning are proposed changes (new functionality, fault repair and adaptation are considered. A decision is then made on which the changes to implement in the next version of the system. The changes are implemented, validated and a new version of the system is released. The process then proposed for a release iterates with a new set of changes. To automate the change management we have to build a collection of tools which we refer to as a "soft change". Soft change defines constructs and present essential measure of software changes size, complexity and developer expertise. Also it provides tools for inputting the purpose and effort required for a change and generates predictions of the quality of the change. The change measure provides a new information infrastructure for managers as well as for future research on large software systems. The new insights generated by using measures of software change underline the importance of studying changes to source code.

Measures on the change can be of 4 classes:

(i) Size measures
(ii) Duration measure
(iii) Complexity measure
(iv) Expertise measure.

- **Size measures**-Include LOC added and deleted and LOC in the files touched by the change .The change size is also measured by number of sub changes.
- **Duration measures-**Indicate the temporal interval spanned by the change.
- **Complexity measures**-change complexity or interaction includes total number of files, modules subsystems touched by a change or the number of developers or organization involved.
- **Expertise measures**-change expertise measures are based on average expertise of developers performing the change.

III Clustering

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some re-designated criteria or criterion , while observations drawn from different clusters are dissimilar.Using with different clustering techniques make different assumptions on the structure of the data. Defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Different methods are based on estimated density and graph connectivity. Clustering is method of unsupervised learning, for statistical data analysis a common technique is used .Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of explorative data mining, a a common technique for statistical data analysis used in many fields, including machine, pattern recognition, image analysis, bioinformatics and information retrieval.

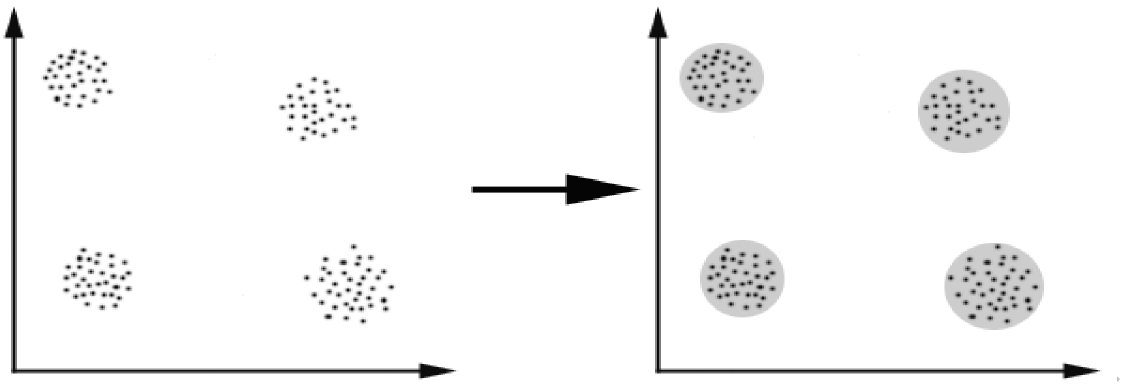

Figure: 1.1 cluster formation

IV LITERATURE REVIEW

[1] Kabaili, H.; Keller, R.K.; Lustman, F., "Cohesion as changeability indicator in object-oriented systems," Software Maintenance and Reengineering, 2001. Fifth European Conference on, vol., no., pp.39,46, 2001.The assessment of the changeability of software systems is of major concern for buyers of large systems found in fast-moving domains such as telecommunications. One way of approaching this problem is to investigate the dependency between the changeability of the software and its design, with the goal of finding design properties that can be used as changeability indicators. In the realm of object oriented systems, experiments have been conducted showing that coupling between classes is such an indicator. However, class cohesion has not been quantitatively studied in respect to changeability.

[2] Hebig, R.; Gabrysiak, G.; Giese, H., "Towards patterns for MDE-related processes to detect and handle changeability risks," Software and System Process (ICSSP), 2012 International Conference on , vol., no., pp.38,47, 2-3 June 2012. One of the multiple technical factors which affect changeability of software is model-driven engineering (MDE), where often several models and a multitude of manual as well as automated development activities have to be mastered to derive the final software product. The ability to change software with only reasonable costs, however, is of uppermost importance for the iterative and incremental development of software as well as agile development in general. Thus, the effective applicability of agile processes is influenced by the used MDE activities.

[3] Ajrnal Chaumun, M.; Kabaili, H.; Keller, R.K.; Lustman, F.; Saint-Denis, G., "Design properties and object-oriented software changeability," Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European , vol., no., pp.45,54, Feb 2000. The assessment of the changeability of software systems is of major concern for buyers of the large systems found in fast-moving domains such as telecommunications. One way of approaching this problem is to investigate the dependency between the changeability of the software and its design, with the goal of finding design properties that can be used as changeability indicators. In our research, we defined a model of software changes and change impacts, and implemented it for the C++ language. Furthermore, we identified a set of nine object-oriented (OO) design metrics, four of which are specifically geared towards changeability detection. The model and the metrics were applied to three test systems of industrial size. The experiment showed a high correlation, across systems and across changes, between changeability and the access to a class by other classes through method invocation or variable access. On the other hand, no result could support the hypothesis that the depth of the inheritance tree has some influence on changeability.

V Objective

- Develop representative dataset of projects(object-oriented)
- Identify parameters which impact the changeability of software components
- Develop a framework and scale to measure the changeability metrices.
- Using machine learning algorithm develop automated assessment of changeability
- Evaluate the model using recall and precision

VI Methodology

Selecting projects for developing representative datasets:

For any software analysis the basic step is to create the database. At this step we develop the representative datasets so that we could evaluate metrics and then changeability could be measured.

Downloading the projects from git repository

In this step we are getting open source GIT local repository and the projects after downloading getting stored in Net Beans.
Step1: Explore the git hub repository to find different java projects.
Step2: Select the project and then clone it to Net Beans.
Step3 Select the class repository
Step 4 Now save that project at particular path.
Step5.All the files of that project from git repository getting saved as datasets in Net beans.
Step6 Creating project from the cloned files in Net beans
Step7 Choose the project
Step8: Create java applications.

In this step all the metrics are calculated manually with help of source code These are the following total metrics we have selected) Instability(I)-This metric was proposed by Robert C. Martin and measures the instability of packages, where stability is measured by calculating the effort to change a package without impacting other packages within the application. Robert C. Martin's proposed Stable Dependencies Principle states that the dependencies between packages in a design should be in the direction of the stability of the packages; i.e. a package should depend only on packages that are more stable that itself. The number of incoming and outgoing dependencies is one indicator determining the stability and instability of a package. Packages that contain multiple outgoing but few incoming dependencies are less stable because of the consequences of changes in these packages. On the other hand, packages containing more incoming dependencies are more stable because they are more difficult to change. Stability can be calculated by comparing the incoming and outgoing package dependencies (afferent coupling Ca and efferent coupling Ce): Instability $I = Ce / (Ca + Ce)$.If I falls in the range between 0.0 and 1.0 where 0.0 indicates a maximally stable package and 1.0 indicates a maximally unstable package. However, do not design all packages to be completely stable or completely unstable. Instead, each package should intentionally be made as stable (0.0 to 0.3) or unstable (0.7 to 1.0) as possible.[25]
LOC-Total Lines of Code / Source Lines of Code / Effective Lines of Code: The Total Lines of Code (often labelled LOC, SLOC, and ELOC) metric counts all lines regardless of whether they contain code, comments, or

white space. The lower limit for LOC is default 5 lines and the upper limit is default 1000 lines for a file or a class. It is advisable to split up the class and use delegation, If a class exceeds 1000 lines. It calculates LOC as follows: Method body lines are only counted for a method; class and interface body lines are only counted for a class or interface; lines of all source files belonging to a package are counted for the package. The implications are: Method declarations and Java doc comments are not counted on the method level. They are counted on the class / interface and package levels. Class or interface declarations and Java doc comments are not counted on interface level / the class. They are counted on the package level.

WMC-Weighted method class The WMC metric is the sum of the complexities of all class methods. It is a sign of how much effort is required to develop and maintain a particular class. It sums the V (G) of all declared methods and constructors of class to calculate the WMC. A class with a low WMC points to greater polymorphism. A class with a high WMC shows that the class is complex (application specific) and therefore harder to reuse and maintain. For WMC the lower limit is default 1 because a class should consist of at least one function and the upper default limit is 50.

IV CONCLUSION

Software changeability is associated with refactoring which makes code simpler and easier to maintain. In this we identify the quantum of indexes which belongs to particular project. We want to run the algorithm, we find the expected values called index corresponding to our actual values for each observations n every projects.

REFERENCES

1. INGRAM, C.; RIDDLE, S., "USING EARLY STAGE PROJECT DATA TO PREDICT CHANGE-PRONENESS," EMERGING TRENDS IN SOFTWARE METRICS (WETSOM), 2012 3RD INTERNATIONAL WORKSHOP ON , VOL., NO., PP.42,48, 3-3 JUNE 2012
2. Sebastian G. Elbaum John C. Munson,; “Code Churn: A Measure for Estimating the Impact of Code Change”.
3. The Challenges Of Software Change Management In Today’s Siloed IT Organizations, A Commissioned Study Conducted By Forrester Consulting On Behalf Of Serena Software, November 2006
4. N. Fenton, S. L. Pfleeger and R. Glass, "Science and Substance, A Challenge to Software Engineers", IEEE Software, July 1994, pp.86-95.
5. D. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softChange. In Proc. of the 16th Internation Conference on Software Engineering and Knowledge Engineering (SEKE 2004), pages 336–341, 2004.
6. Nary Subramanian, Lawrence Chung;, “Metrics for Software Adaptability” 2Department of Computer Science University of Texas, Dallas Richardson, TX, USA,
7. Gentzane Aldekoa1, Salvador Trujillo2, Goiuria Sagardui1, Oscar Díaz2”Experience measuring maintainability in software product lines”; XV Jornadas de Ingeniería del Software y B.ases de Datos JISBD 2006
8. Chidamber, S. R. and Kemerer, C. K. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, Vol. 20 (June 1994), pp.476-493.
9. Ajrnal Chaumun, M.; Kabaili, H.; Keller, R.K.; Lustman, F., "A change impact model for changeability assessment in object-oriented software systems," Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on , vol., no., pp.130,138, 1999
10. J. C. Munson, S. G. Elbaum, and R. M. Karcich, "Software Risk Assessment Through Software Measurement and Modeling", will appear Proceedings of the 1998 IEEE Aerospace Applications Conference, IEEE Computer Society Press
11. en.wikipedia.org/wiki/Cluster_analysis
12. Jukka Kainulainen “Clustering Algorithms: Basics and Visualization”HELSINKI UNIVERSITY OF TECHNOLOGY Laboratory of Computer and Information Science T-61.195 Special Assignment 1
13. Jon Kleinberg;” A Theorem for Clustering”;, Department of Computer Science Cornell University
14. MA Chaumun, RK Keller, F Lustman - Advances in software engineering, 2002 - dl.acm.org
15. Bieman, J.M.; Andrews, A.A.; Yang, H.J., "Understanding change-proneness in OO software through visualization," Program Comprehension, 2003. 11th IEEE International Workshop on , vol., no., pp.44,53, 10-11 May 2003
16. Kabaili, H.; Keller, R.K.; Lustman, F., "Cohesion as changeability indicator in object-oriented systems," Software Maintenance and Reengineering, 2001. Fifth European Conference on , vol., no., pp.39,46, 2001 Adam M. Ross, Donna H. Rhodes, Daniel E. Hastings, Volume 11, Issue 3, pages 246–262, Autumn (Fall) 2008
17. 2007Changeability in operations: A critical strategic resource for European manufacturing " Universities Engineering Conference, 2007. UPEC 2007. 42nd International , vol., no., pp.930,937, 4-6 Sept.