Volume 5, Issue 4, 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Survey on Software Sizing for Project Estimation

Eric Githua Ng'ang'a*
Department of Information Technology
Mount Kenya University, Kenya

Isaac Kiprotich Tonui
Head of Information Technology Department
Mount Kenya University, Kenya

Abstract— Software sizing is an activity in software engineering that is used to estimate the size of a software project in order to be able to apply other software project activities. Accurate software project estimation is determined by the degree to which the software managers have correctly estimated the size of the software. Accurate sizing estimation is an important measurement in the calculation of estimated project costs, effort, schedules and duration which provides important information for software project development. Estimation in software projects can be carried out by first measuring the size of the product to be developed. This paper analyses and provides a details overview of two most common software sizing metrics; Lines of codes (LOC) and Function Point Analysis (FPA). Their strengths and weaknesses are also examined; the 'second generation' software sizing methods are also discussed. The paper also presents remarks on the findings and future research of the software sizing methods.

Keywords— software sizing, software project estimation, software size estimation, COSMIC, software effort estimation, Lines of Code, Function Point Analysis, Use cases.

I. INTRODUCTION

Software sizing is an activity in software engineering that is used to estimate the size of a software component in order to be able to implement other software project management activities [1]. To attain reliable software projects success in software development, software planners need achievable plans including viable estimates of schedules, sizing, resources, and risks [2]. Software project failure has become one of the main challenges and norm in software engineering industry in fact many are cancelled before completion or not implemented at all. Addressing some of the key reasons why software projects fail is the best starting point. According to [3] the main reasons behind project failures and software problems occur because of the following;

- Inability to accurately size a software project.
- Inability to accurately size a software project,
- Inability to accurately specify an appropriate software development and support environment,
- Improper assessment of staffing levels and skills.
- Lack of well defined requirements for the software activity estimated.

Mainly software projects fail because of the inaccuracy of software estimation.
Software sizing remains a demanding area in the field of software engineering, according to [4] the main reasons for measuring software size are;

- Software size is a key measure that is used as an input in estimating cost and effort.
- Planned size and actual size are compared to monitor the project achievements.
- Contracts are also formed by using software size measurement.
- Software size is also used in normalization of other measures.

The software size should be estimated in the requirements stage of the software development life cycle. Software sizing is one of the main inputs to software development effort, if size is estimated correctly; the effort estimate will be realistic and will translate to a realistic cost estimate [2]. Without a concept of function size software managers may find it challenging to confer practical schedules based on their established ability to deliver software products. Several software sizing methods and techniques exist. Once software size is determined business estimate the required effort to construct the required software product.

II. SOFTWARE SIZING METHODS AND TECHNIQUES

Many software sizing approaches and techniques have been proposed over the years by engineering research community. Software project planners are using these approaches to estimate the exact project cost for developing software products. These software sizing techniques can be broadly categorized into two, code based sizing metrics and functional size measurement (FSM). Code based sizing metrics measure the size of software using the program source code. Examples of code based sizing metrics include LOC (Lines of Code), Halstead's software length equation, McCabe's Cyclometic Complexity. FSM is an alternative to code based sizing method which include Function Points Analysis (FPA) which takes into account both static and dynamic aspects of the system [5]. Reference [6] identifies

LOC and FPA as most commonly used software sizing techniques. In this paper we describe in details these two most popular sizing metrics and techniques that have been proposed and applied in practice. During software project estimation LOC and FPA are used as estimation variables to size every element of the software and as a baseline metrics collected from past software projects and estimation variables to develop cost effective projects [7].

A. Lines of code

LOC is the most commonly used and simplest metric to estimate project size. The measure was first proposed when programs were typed on cards with one line per card, where project size is estimated by counting the number of source instructions in the finished software project. LOC is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments on the line. This specifically includes all lines containing program header declarations, executable and non executable statements. LOC is normally used to calculate the amount of effort that will be required to build a software program and estimate programming productivity and maintainability [8].

Accurate estimate in LOC can only be obtained after the completion of software project and there is no accepted standard definition exists for LOC [9]. Counting LOC is very simple method but difficult to estimate LOC from problem description and as a result LOC is not very useful for project planning.

B. Function Point Analysis

Function points were developed by Albrecht (1979) at IBM as a way to measure the amount of functionality in a system [10]. Later in 1983 Albrecht and Gaffney extended and published the method [11]. This metric overcomes many of the limitations of the LOC sizing metric. FP measure the amount of functionality in a system by counting inputs, outputs, queries, and logical files. Function points capture the size of an actual product and does not relate to something physical but rather to something logical that can be assessed quantitatively. FPA can be used to easily estimate the size of a software product directly from the problem specification and it measure the functionality of a program from what the user requests to system and receives in return from the system as output. The FPA is divided into following functional units [12]; input or set of inputs to the system, outputs from the system, request for access data or information, internal logical information, and external interface required. Once requirements are categorized, then their respective complexity is associated with each functional requirement based on subjective judgement of the organization.

III. ADVANTAGES AND DISADVANTAGES OF SIZING METHODS

Software size is a key input to all software cost estimation models. In the past two decades LOC and FP have been dominating software sizing methods. Unfortunately, there are significant drawbacks in LOC and FP for software sizing estimation. LOC is simplest method (ease of counting automation) but can only be accurately counted when the software product is complete, while the most critical software estimations need to be performed before construction. Software sizing using LOC is specific to the technology platform on which the application was developed, the LOC count will vary for different technology platforms. Disadvantages include difficulty of counting early in the development process, dependence on programming style, need for inflexibility in applying counting rules, and unpredictability of methods across different programming languages [13].

FP can only be manually counted, and the estimator has to have experience to do so. Moreover, FP counting involves a degree of subjectivity. The FP counting rules are satisfactorily well defined and codified to enable experienced FP planners to be constant in FP counts, unfortunately manual FP counting and recounting processes are unavoidably time consuming and expensive relative to automated counting. The software size measured through FP approach is independent of the technology on which the application is developed. Another advantage includes ease of generation from a clear specification and persistence across intermediate software products. With FP there is inconsistency as analysts interpret the notional constructs and the difficulty of assessing the size of embedded systems [13].

IV. 'SECOND GENERATION' SOFTWARE SIZING METHODS

Because of the above challenges with 'first generation' software sizing methods, software engineering researchers' community are looking for faster, cheaper, and more effective methods for measuring software size. Other software sizing methods include use case based software sizing, which relies on counting the number and characteristics of uses cases found in a software and COSMIC which addresses sizing software that has a very limited amount of stored data such as real time systems [1]. A benefit of using the COSMIC method includes [14]:

- Easy to learn and stable due to the principles based approach, hence a future proof and cost effective method.
- Accepted by software project staff due to the ease of mapping of the methods concepts.
- Improves estimating accuracy, especially for larger software projects.
- Possible to size requirements automatically that is held in CASE tools.
- COSMIC reveals real performance improvement where using FP has not indicated any improvement due to their inability to recognise how software processes have increased in size over time.
- Sizing with COSMIC is an excellent way of controlling the quality of the requirements at all stages as they evolve.

Use cases provide software developers with insight into software requirements [7]. According to [15] the following are problems of estimating software size using use case;

- There is no standard form for describing use cases; it has many different formats and styles.
- Use case represents an external view of the software and can therefore be written at many different levels of abstraction.

- Use cases do not address the complexity of the functions and features that are described
- Use cases can be described using complex behaviour that involves many functions and features.

V. CONCLUSION AND FUTURE WORK

In this paper an overview of traditional and 'second generation' software sizing techniques, their strengths and weaknesses have been discussed. Various software sizing approaches have been proposed over the years and each of them has taken the solution of estimating software size a step further. Software size estimation is critical to providing a credible software cost estimates, thus choosing the appropriate method by which to estimate size is important. None of these software sizing methods gives hundred percent accurate software size estimation. Software engineering research community must continue to research on reliable, accurate, and effective software sizing techniques. Improving the existing software sizing methods and introducing new methods for inputs for software cost estimation will be our future work.

ACKNOWLEDGMENT

We would like to express our gratitude and special thanks to the Mount Kenya University, Mombasa Campus Management for providing us with a conducive environment to do this research.

REFERENCES

[1] (2015) Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Software_sizing.
[2] Daniel D. Galorath and Michael W. Evans, *Software Sizing, Estimation, and Risk Management: when Performance is measured Performance Improves*, 1st ed., Auerbach Publications, 2006.
[3] *Parametric Estimating Handbook*, International Society of Parametric Analysts, 2008.
[4] Gencel C. and Demirors O., Measuring the Software Functional Size as a Vector of Measures, 2nd ed., Boston PWS Publishing, 2008.
[5] Vu Nguyen, "Improved Size and Effort Estimation Models for Software Maintenance," Doctor of Philosophy, Computer Science, University of Southern California, Dec. 2010.
[6] Hee B. Kuan, Yuan Zhao and Honguy Zhang, "Estimating LOC for information systems from their conceptual models," ACM/*IEEE International Conference on Software engineering*, pp. 321–330, May. 2006.
[7] Roger S. Pressman, Software Engineering a Practitioner's Approach, 7th ed., McGraw-Hill, 2010.
[8] (2015) Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Source_lines_of_codes.
[9] Chander Diwaker, Astha Dhiman, "Estimating Size and Effort Estimation Techniques for Software Development," *International Journal of Software and Web Sciences (IJSWS)*, pp. 2279–0071, May. 2013.
[10] Albrecht A., "Measuring Application Development Productivity," Proc. IBM Applications Development Symp, pp.89-92, 1979.
[11] Albrecht A. and Gaffney J., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*., vol. SE-9, No. 6, Nov. 1983.
[12] Gunjan K. Bhati and Pooja, "An Approach for Software Size Estimation," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, pp. 19–25, Feb. 2014.
[13] Shari L. Pfleeger, Felicia Wu and Rosalind Lewis, "Software Cost Estimation and Sizing Methods: Issues and Guidelines," RAND Corporation, 2005.
[14] (2015) Wikipedia. [Online]. Available:http://en.wikipedia.org/wiki/COSMIC_software_sizing.
[15] Smith J., "The Estimation of Effort Based on Use Cases," Rational Software Cord, 1999.

