Turkish Journal of Computer and Mathematics Education *Vol. 9 No.02 (2018), 474-481*
DOI: https://doi.org/10.17762/turcomat.v9i2.13851

Research Article

Estimation Model Analysis and Development for Object-Oriented Software Testing

Mahesh Manchanda
Professor, Department of CSE (Computer sc), GEHU-Dehradun Campus

Abstract:

For practitioners to make informed decisions about resources like time and money, it's crucial that they get an accurate picture of the scale of software projects early in the Software Development Life Cycle (SDLC). The current study proposes an adaption of the standards-based Function Point Analysis (FPA) model to meet the need for a uniform approach for estimating the size of OO software development and testing. At the SDLC's analysis and design stage, when UML diagrams are used, the adapter records the many characteristics of the OO software. During the analysis phase, the size of the development and testing phases is estimated using components derived from the UML use case model. Estimation components are derived from the UML object model and used to calculate the amount of development during the design process..

Keywords: Software Development Life Cycle (SDLC), Function Point Analysis (FPA), Unified Modeling language (UML)

Introduction

Estimation is the process of determining, on the basis of software requirements, the most accurate measurement of the size needed to build and test the programme. The size estimations are then included into the planning, iteration, budgeting, investment analysis, pricing, and bidding phases of the project. Estimation issues in software development projects have been studied and worked on since the 1960s by scholars and professionals in the field. The size of the programme has been shown to have a direct correlation to its effect on time, money, and upkeep by several researches. Accurately managing the many moving parts of the Software Development Life Cycle (SDLC) requires reliable software estimate. There has to be oversight on the next iteration of the product as the programme ages. As a result, it is important to do estimating prior to development in order to maintain product control. Both internal and external factors might be considered when making an estimate. The size, complexity, and modularity of a software project are all internal properties that may be estimated. Attributes that may be evaluated by considering the project's context are called external attributes. Reliability, understandability, maintainability, and so forth are all examples of external qualities. Attributes that are visible to the outside world are very subjective and hard to measure. Therefore, in this study, we focused on size, an intrinsic characteristic, to develop an estimating model. The amount of functionality that must be built for a software project is a good proxy for the size of the product. The software development life cycle (SDLC) cannot progress without first estimating the size of the project. With this information, a comprehensive strategy for the software project may be produced at an early stage, including the other metrics like effort, cost, and productivity. Top-down and bottom-up approaches exist for estimating sizes. Expert opinion forms the basis of top-down estimating, whereas an estimation model guides bottom-up approaches. Several scientists are responsible for the precision of the expert estimate. During the early phases of the software development life cycle (SDLC), the expert estimation may be used to provide an estimate of the software project's size. Due to the general consensus that estimators can be relied upon when putting the approach to work, expert estimation is increasingly adopted. If the project uncertainty is great and the estimator's skills are limited, the resulting estimate will be inaccurate. Incorrect estimations might cause delays and additional costs after the development process has begun. The probability of a project's success or failure is proportional to how well its estimates were made. The following are common limitations of expert estimates::

i) Do not follow any pattern
ii) Traceability is not easy.

A method for creating an objective bottom-up estimate model is crucial for fixing the aforementioned issues. There are two types of bottom-up estimating models. Function Point Analysis (FPA), Mark II Function Point Analysis (FFA), and COSMIC-FFP are all examples of technology-independent techniques, whereas Lines of Code, Database, and UML are all examples of technology-dependent techniques. Figure 1 illustrates software project management size estimating techniques.

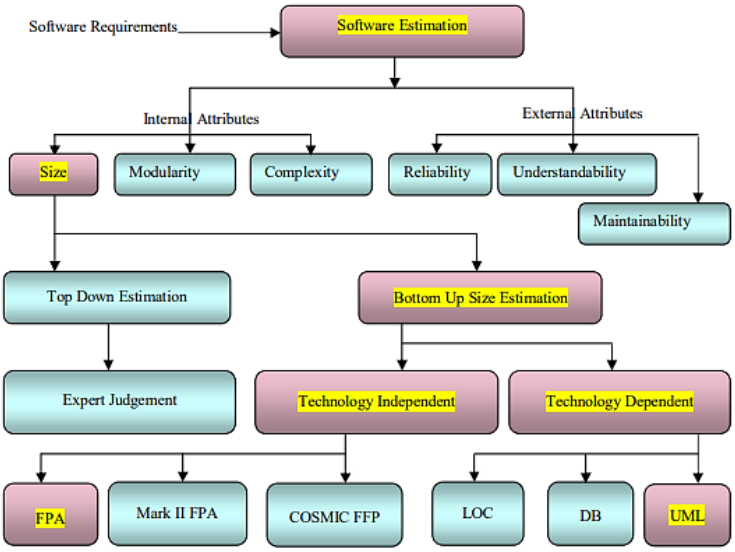

Figure 1.Size estimation processes in software project management

Technology Independent Estimation Methods

There are a number of different estimating approaches available for determining how big a software project will be. IBM's function points are often used to estimate the size of a software project and, by extension, to anticipate the amount of time and money needed to complete the job. The FPA's most notable quality is that it may be used at the preliminary analysis stage of the software development life cycle (SDLC). After the establishment of the International Function Point User Group (IFPUG), its implementation became increasingly commonplace. The function point method had been used to estimate a large number of software projects. Mark II FPA and COSMIC - FFP are two notable enhancements to the FPA technique that have been extensively used in practise.

Object Oriented Size Estimation Models

Estimating the size of an OO programme is a difficult task at the beginning of the software development life cycle. The overall size of an OO system is affected by two factors: development and testing. The size of a piece of software is often calculated by adding the sizes of its source code and test cases/test drivers together. The former is the size of the programme during development, while the latter is the size of the software during testing. The amount of a piece of software that has been coded in order to realize its functionality, excluding any code that is necessary for testing, is known as its "development size." Software quality demands that considerable time and effort be invested in testing. In order to avoid unfavorable reactions from customers and reputational harm to the company

developing the software, the size and effort estimate for testing must be adequate to provide enough time to conduct tests and guarantee that the product enters the market adequately tested. In reality, software companies delegate the task of estimating the testing scope to a dedicated test manager, who then plans the software testing management operations appropriately. Using the many UML diagrams and requirements accessible at each stage of the software life cycle, researchers have contributed to the calculation of the testing effort. There was a pressing need to modify the FPA strategy in light of the widespread use of object oriented methodology in the software industry. Due to a mismatch in the features supplied by OO, the FPA approach cannot be directly implemented on OO software. However, with some tweaks, this technique may be used to predict how much time will be needed for creating and testing OO programmes. The object model and the use case model are two examples of UML diagrams that have been suggested for use in FPA to estimate the size of OO software throughout development and testing. The big picture of OO size estimate is shown in Figure 2. This graphic illustrates the software development and testing size estimate processes that coexist in software management. There are several points in the software development life cycle (SDLC) when size estimate may be done. The test size may be calculated at any point in the SDLC lifecycle, whereas the development size can be determined during the analysis and design phase.

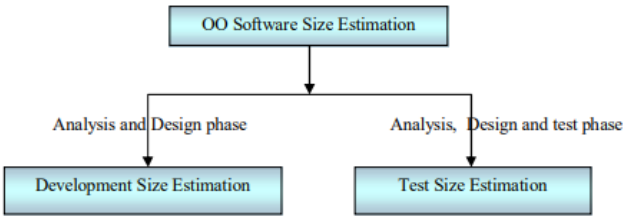

Figure 2. Overview of object oriented size estimation

Estimation at the Design Phase

Many scholars have offered methods for estimating the size of a project during the planning phase. The design phase unveils the full scope of the program's features. This stage starts off after the analysis is finished. During the design process, further details might be gleaned from the existing data. During the development process, a UML diagram may help clarify the proposed software's functionality. This suggests that the suggested software's size estimations may be more reliable. Numerous scholars have offered a method for estimating the scale of OO developments based on their object models that adapts the FPA methodology. During the design phase of the SDLC, researchers mapped the FPA components to an object model and size was predicted. To do this, we integrate the results of a standard, technology-agnostic FPA with those of a technology-dependent UML object model. There were many conflicting guidelines for estimating the OO software development size using the object model in the literature. Existing methods for estimating sizes fail to provide clear instructions to practitioners to aid in the estimate process. In addition, the complexity of a class grows when it has several connections with other classes. There is still some uncertainty about how to classify the components and what kind of adaptation process should be used for estimate. To get rid of the uncertainty, we offer the Object Model Function Point (OMFP) method for estimating the size of OO development projects. The OMFP method eliminates guesswork associated with identifying FPA parts and determining their complexity while keeping object model-based size estimates accurate. Approaches to size estimate during design give thorough instructions for experts to recognise and categorise estimation elements using an object model. Such methods provide precise estimates, but they need comprehensive design data to be calculated. As a result, efforts were made to approximate the software's size throughout the analysis process.

Estimation at Analysis Phase

Many scholars have advocated for development size estimate to occur during the analytical phase. The size was calculated by applying an FPA model to an OO requirements specification. Ashman (2004) used a model based on use cases to estimate the work required for a project. The use case model for size estimation has been the subject of a lot of study. The estimating model was developed by mapping the FPA to the use case model. The actor and use case weights in a use case diagram may be estimated using the two complexity tables. During the first stages of study, the use case size points were suggested. The use case diagram's actor, use case, exception, precondition, postcondition, and scenario weights are estimated using five complexity tables. Here we present a new method for sizing UML projects that takes into account both use case points and class points. At the analysis stage, sufficient information is provided by the UML use case model for size estimate. Current methods rely only on the use case diagram for their estimates, ignoring the wealth of data provided by the use case documentation. Furthermore, they don't follow any norms. This is why we've merged the technology-neutral FPA with the UML use case model to provide us yet another tool for size estimate. By making the necessary adjustments to the FPA approach, the Use Case Model Function Point (UCMFP) technique is offered for estimating the scale of new developments. The rules for each component of size estimation are provided to practitioners in full detail during the analysis phase of size estimation. These guidelines help in determining how difficult each element of an estimate really is. The UCMFP approach helped with OO software development size estimates but was unable to provide an estimate for system testing. As part of the Software Development Life Cycle (SDLC), system testing is performed to ensure the requirements are met. The equivalent size estimate must be made early in the SDLC process.

Literature Review

Umang Garg et.al (2016) Over the last decade, many have reflected on the value of testability. The degree to which flaws in the code can be easily identified during testing is a measure of how easily software can be tested. Along with other considerations, this study advocates factoring in the developer and the development process when predicting software testability. The capacity to test software is crucial to determining whether or not a project is feasible and poses an acceptable level of risk. As a result, testability analysis is crucial in the modern era, necessitating the identification of all conceivable aspects and stages of the software lifecycle that impact testability. In this study, we provide five novel criteria that influence the testability of object-oriented software; these factors all consider the developer and the development process. The effect of these variables on testability is also discussed.

Ashish et al (2012) has experimentally provided test metrics for software testing effort estimate based on the definition of software needs

Xiaochun et al (2008) test case number prediction model and test suite execution vector model were suggested as a means of estimating the time and energy required to conduct a test. In order to quantify the number of use case transactions and entity objects, this method introduces a new statistic called the use case verification point. The time required to execute tests is estimated based on the number of cases to test, the complexity of the cases, and the tester's position. Expert opinion is crucial to the precision of the estimate. The estimation of the complexity of test execution needs clear guidelines. Use case verification rules must be defined.

Aranha and Borba (2007) suggested a method for calculating the time required to execute tests in light of the test plan. The definition and verification of a metric for the amount and complexity of test executions. This estimation's testing protocol was designed using only plain language. Many test cases make up a test suite, and the execution points are derived through an examination of each test case's functional and non-functional features. This work takes into account eight functional features and three non-functional characteristics. Test execution effort is determined by multiplying the total number of execution points by the test team's efficiency. Norms and relative importance for each trait have been laid forth. The reasoning for the relative importance of the different criteria isn't entirely clear..

Zivkovic et al (2005) the designer's viewpoint, suggested a three-level abstraction hierarchy and an iterative estimation technique (IET) to increase the estimate's precision. The estimation process consists of three phases: the initial, intermediate, and ultimate stages. Use case diagrams were used in the preliminary stage. Sequence & activity diagrams were employed during the comparison portion of the study.

Estimation Model

Estimating software size is crucial for making informed decisions in any software company. The total size of an object-oriented software system is affected by both its development and testing phases. Estimating the scope of OO software development by mapping the Functional Performance Assessment to a use case model. Although this method made it easier to predict how much OO software would cost to create, it was unable to predict how much time would be needed for testing. The software system is tested in its entirety during system testing, a stage of the software life cycle used to verify and validate the requirements. At an early point in the software life cycle, it is crucial to evaluate the work required in the different tasks associated to system testing. UCM often offers comprehensive details on all the features handled by a certain application. The system requirements may be captured early in the software development life cycle, which has its own set of benefits. In addition, it records requirements during the analysis phase with several degrees of granularity, including short, fully-specified, and refined. UCM displays a wide range of behaviours that are sufficient for recording system testing procedures. UCM may be used to estimate system testing activities.

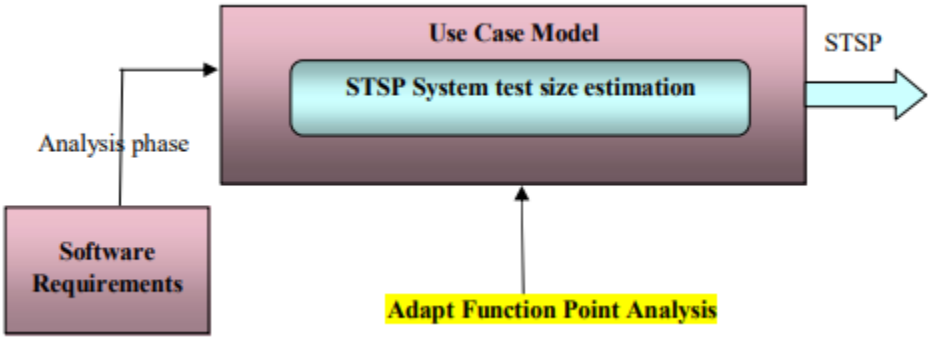

Figure 3. System test size point size estimation model

Testability Constructs & Factors

Many significant aspects impacting testability were discovered during an examination of testability across the software development life cycle in object-oriented stems. These considerations, as well as antipatterns and improvisational strategies, need to be illuminated. Important elements that aid in testability include:

- Key Object Oriented Features: Testability criteria appropriate for the design phase must be determined when a thorough understanding of the properties of object-oriented software has been achieved. Class Size, Coupling, Cohesion, Encapsulation, Inheritance, and Polymorphism are all key object-oriented traits that contribute to testability, according to the study. As they are major contributors to the primary quality variables influencing testability, they need a thorough examination from every angle.

- Object Oriented Metrics: Several of these characteristics are implemented in the form of well-known object-oriented metrics that are acceptable [10], [11] for use in testability estimates throughout both the design and coding phases.Many of the measures have shown a robust link with testability via empirical validation using commercial

Java applications and joint test classes[3, 14]. Consequently, the significance of these measures in determining testability has been negligible.

- Six Major Contributing Factors: Controllability, observability, complexity, traceability, understandability, and built-in test are six quality variables that have been proven to impact overall testability [6]. Researchers have focused on diverse aspects of the software development process[3, [12], [15], [17], yielding a wide range of theories, models, and metrics for quantifying these aspects, all of which contribute to enhanced testability. While these factors have been shown to improve testability and reduce testing time and effort individually, the research has yet to demonstrate their collective benefit. In accordance with Tables 1 and 2,

Functional parameter estimation							
Test set #	File	Files			DT		
		ESEQ	ES	Comp	ESEQ	DF	Comp
1	IRF	6	38	15	6	23	6
2		5	61	15	5	21	6
3		5	26	10	5	11	6
4		5	26	10	5	12	6
5		7	29	15	7	22	6
6		6	24	15	6	32	6
7		4	19	7	4	29	6
8		2	7	7	2	5	4
9		2	8	7	2	3	3
Total IRF Complexity				101			-
10	ERF	3	10	7	2	8	4
Total ERF Complexity				7	-	-	-
Total DT Complexity							53

System testing technical complexity factors estimation		
Term	Factor	Wt
FF	Test environment	1
	Test documentation	2
	Test conditions	3
	Test ware	1
NFR	Performance	2
	Load testing	2
	Security testing	2
	Stress testing	0
OF	Testing tool	0
	Testers skill	0
	Innovative technology	0
	Multiple test configurations	0
	Geographical distribution of team	0
	Product size	0
Total STTCF Complexity		13

Table 1.System test size point estimation for the application

Project	Number of usecases	Complexity due to		STSP	UCMFP	UCMFP development Effort (hrs)	STSP Test Effort (hrs)
		IRF and ERF	DT				
P1	13	75	46	94.91	83.46	27.1162	15.4229
P2	13	86	54	109.82	118.77	38.5884	17.8458
P3	16	91	66	122.59	116.63	37.8931	19.9209
P4	13	79	51	101.72	110.21	35.8072	16.5295
P5	11	66	32	76.44	82.39	26.7685	12.3923
P6	10	69	49	92.53	85.6	27.8114	15.0361
P7	11	84	52	106.08	98.44	31.9832	17.238
P8	11	83	61	112.04	107	34.7643	18.2065
P9	12	77	49	98.14	104.86	34.0690	15.9478
P10	10	84	42	98.28	83.46	27.1162	14.3374
P11	14	68	46	104.9	113.42	36.8502	17.0463
P12	13	82	52	109.45	104.86	34.0690	17.7856
P13	12	64	43	83.34	90.95	29.5497	13.5428
P14	10	72	49	94.56	87.74	28.5067	15.366
P15	12	79	63	111.65	105.93	34.4167	18.1431

Table 2 Test size estimation results

Choice of Estimation Methods

The ability to accurately predict the size of a piece of software is crucial for making good choices at every stage of the SDLC. Software project management relies heavily on this step, since it facilitates other crucial processes like budgeting, profit forecasting, resource planning, scheduling, etc. An correct estimate generates profit for a company, while an inaccurate one might cause financial harm. Size may be estimated using either a bottom-up or top-down method. The programme may be divided into bigger tasks using bottom-up estimates, with each of those jobs then being split into smaller, more specific tasks. This method yields reliable estimates since it takes into account the granular details of the work to be done. In addition, it is dependent on certain estimate models. Top-down estimation is based on the application of past practical knowledge. The estimate may be off since it is based on the judgement of experts.

Conclusion

The ability to test software is rising up the ranks of priorities for developers and testers. There are several unfilled research needs regarding testability after reviewing it throughout the software development life cycle and in its many manifestations. Important quality aspects in the study's design and coding phases need to be examined constructively and quantitatively. While some of these considerations may have previously been addressed, analysing how they all work together to preserve essential object-oriented properties remains an open research question. Testability research that focuses on the designing phase rather than the coding step is also gaining traction. The focus should not be limited to the design phase; rather, it should be expanded to include the source code and testing phases, where testability can be improved by taking advantage of what was learned during the design phase. As a result, software developers will be able to produce software that is highly dependable, readily maintained, and easily tested, all while reducing testing effort and development expense.

References

[1] U. Garg and A. Singhal, "An estimation of software testability using fuzzy logic," *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, Noida, India, 2016, pp. 95-100, doi: 10.1109/CONFLUENCE.2016.7508094.

[2] Ashish, S. and Dharmender, S.K. “Applying requirement based complexity for the estimation of software development and testing effort”, ACM SIGSOFT Software Engineering Notes, Vol. 37, No.1, pp. 1-11, 2012.3

[3] Xiaochun, Z., Bo, Z., Fan, W., Yi, Q. and Lu, C. “Estimate test execution effort at an early stage: An empirical study”, IEEE International Conference on Cyberworlds, pp. 195-200, 2008

[4] Aranha, E. and Borba, P. “An estimation model for test execution effort”, First International Symposium on Empirical Software Engineering and Measurement, pp. 107-116, 2007.

[5] Zivkovic, A., Hericko, M., Brumen, B., Beloglavec, S. and Rozman, I. “The impact of details in the class diagram on software size estimation”, Informatica, Vol. 16, No. 2, pp. 295-312, 2005

[6] Zivkovic, A., Rozman, I. and Hericko, M. “Automated software size estimation based on function points using UML models”, Journal of Information and Software Technology, Vol. 47, No. 13, pp. 881-890, 2005.

[7] Fernandez, N.C., Abrahao, S. and Pastor, O. “Towards a functional size measure for object oriented systems from requirements specifications”, Proceedings of the Fourth International Conference on Quality Software, pp. 94-101, 2004.

[8] Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L. and Guoliang, Z. “Generating test cases from uml activity diagram based on gray-box method”, 11th Asia-Pacific Software Engineering Conference, pp. 284-291, 2004.

[9] Zivkovic, A. and Hericko, M. “Tips for estimating software size with FPA method”, IASTED International Conference on Software Engineering, Acta Press, pp. 515-519, 2004.

[10] Andrews, A., France, R., Ghosh, S. and Craig, G. "Test adequacy criteria for uml design models", Journal of Software Testing, Verification and Reliability, Vol. 13, No.2, pp. 95-127, 2003
[11] Antoniol, G., Fietum, R. and Lokan, C. "Object oriented function points: An empirical validation", Empirical Software Engineering, Vol. 8, No. 3, pp. 225-254, 2003
[12] Chen, T.Y., Poon, P.L. and Tse, T.H. "A choice relation framework for supporting category-partition test case generation", IEEE Transactions on Software Engineering, Vol. 29, No.7, pp. 577-593, 2003
[13] Zivkovic, A., Hericko, M. and Kralj, T. "Empirical assessment of methods for software size estimation", Informatica, Vol. 4, pp. 425-432, 2003
[14] Uemura, T., Kusumoto, S. and Inoue, K. "Function point analysis using design specification based on the Unified Modeling Language", Journal of Software Maintenance Evolution-Research Practice, Vol. 13, No. 4, pp. 223-243, 2001.
[15] Janaki Ram, D. and Raju, S.V.G.K. "Object oriented design function points", First Asia Pacific Conference on Quality Software, Hong Kong, pp. 121-126, 2000
[16] Uemura, T., Kusumoto, S. and Inoue, K. "Function Point measurement tool for UML design specification", Sixth International Symposium on Software Metrics, Vol. 62, pp. 62-69, 1999.
[17] Veenendaal, E.V. and Dekkers, T. "Test point Analysis: a method for test estimation", Project Control for Software Quality, 1999.
[18] Antoniol, G., Lokan, C., Caldiera, G. and Fiutem, R. "A function point-like measure for object oriented software", Empirical Software Engineering, Vol. 4, No. 3, pp. 263-287, 1999.