Software Test Effort Estimation Methods

Matthias Kerstner

February 2, 2011

Abstract

Especially in the field of software engineering, effort estimation plays an essential role. Moreover, the task of defining test scenarios and test cases is often underestimated during the early stages of the software engineering process, due to the fact, that the amount and granularity of required test cases and test steps is not known in advance, for instance, as a result of missing or incomplete requirement specifications. Furthermore, the fact that fix price projects are becoming more and more common makes the process of prior (test) effort estimation even more important, in order to not exceed the budget agreed upon. This paper discusses three important aspects of this development. First, it elaborates on the changing practices in project pricing, followed by a discussion of the general problems involved in the field of effort estimation in software projects. Finally, a distinct selection of today's commonly used methods for test effort estimation will be presented: *Function Point Analysis*, *Test Point Analysis* and *Use Case Points*.

Keywords: Software Effort Estimation, Test Effort Estimation Methods, Function Point Analysis, Test Point Analysis, Use Case Points

Contents

List of Figures

1 Introduction

In general, the software engineering process represents a complex and often rather long-lasting endeavor. Consequently, the need for clear a and thoroughly documented list of (functional) requirements is essential for estimating the effort needed to complete this process, i.e. to produce the final product. Thus, the total cost estimation that will be agreed upon by the partners involved must be as accurate as possible to ensure a project's economic survival. The fact that more than 60% of today's software projects done in Europe are fixed price projects [Sne07], draws further attention on the effort estimation problem.

Moreover, commonly used software development models, such as the V-XT model[1], developed and managed by the Industrieanlagen-Betriebsgesellschaft mbH (IABG[2]), add additional constraints to this problem. This is due to the fact, that these models often do not provide fine grained enough means to be able to achieve thoroughly complete requirement documents at the early stages in which they are originally produced with respect to their corresponding development process. For instance, according to [Sne07], using the V-XT model effort estimations are primarily based upon the requirements specification ("Lastenheft"), written by the project's applicant, which especially for more complex applications cannot be regarded as completely accurate in the sense of that it contains every possible requirement combination.

Consequently, when using the V-XT model, [Sne07] argues that effort estimations exclusively based upon the requirements specification document produced by the applicant at a very early stage in the software development process can at no means lead to accurate estimation results. He backs up his statement by adding, that in the V-XT model requirements specification documents, as opposed to being complete, merely represent an applicant's "wish list", containing a list of functional requirements, as well as qualitative properties, i.e. compliance to the user requirements. Furthermore, he states, that currently no effort estimation method exists that is solely based upon the limited set of these attributes. Therefore, [Sne07] concludes, that the detailed information needed for accurate effort estimations is actually provided in form of the functional specification document ("Pflichtenheft") written by the contractor, solely based upon the requirements specification document. Unfortunately, at the point of the publication of the functional specification document, the contract has already been signed by the parties, thus the price has been fixed.

1.1 Effort Estimation

Consequently, independent of the estimation method used to calculate the effort needed, the more detailed the information provided is, the more accurate the estimation will be. Function-point, object-point, use case-point, COCOMO-I & II or even domain expert knowledge, all methods for effort estimation share the same need for information in order to be able to produce accurate results. Hereby, all estimation methods are based upon the *five core metrics* introduced by [LPM03], as depicted in Figure 1. These are

1. Quality
2. Quantity
3. Time
4. Costs
5. Productivity

[1] See http://www.v-modell.iabg.de/
[2] See http://www.iabg.de/

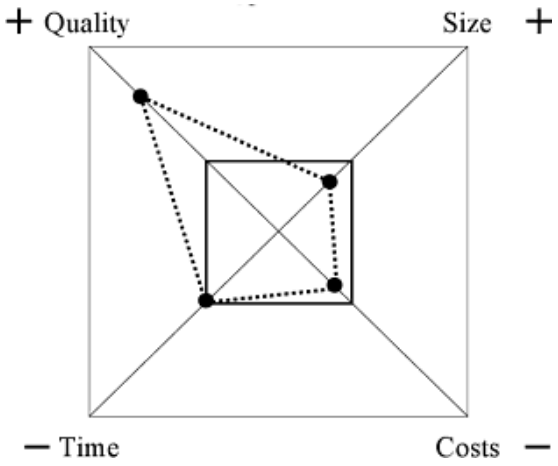

Figure 1: The Devils Square (reproduced from [BD08])

The resulting "devil's square", as shown in Figure 1, depicts the problem of finding the best fitting combination of these factors. Hereby, *productivity* is a constant, that is fixed for a company at the time of a project. *Quantity* represents the "size" of a project. In terms of software, there are a variety of possible metrics available to measure its size, for instance, lines of code (LOC), function-points or use case-points. The final choice depends on the data available, such as data-flow diagrams for function-points or UML diagrams for object-points. In software engineering, *quality* is represented as the metric of compliance to user requirements. Thus, in order to be able to measure quality, the degree of conformance must previously be made quantifiable. *Costs* on the other hand are calculated using a rather trivial estimation function, based on the product of quantity and quality, divided by the corresponding productivity. Finally, *time* is expressed as a logarithmic function based on the effort.

1.2 Cone Of Uncertainty

As already mentioned before, a crucial level of information is needed to achieve meaningful estimations. Unfortunately, especially at early stages in the development process, estimations may not be as precise as they should be. This fact is also referred to as the "Cone of Uncertainty". The idea goes back to a study by NASA showing that estimation calculations done at the beginning of the project's life cycle suffer from a very high uncertainty factor that steadily decreases during the progress of the project. Figure 2 shows a highly simplified version of the cone of uncertainty. It describes the fact, that the uncertainty factor changes over the period of time a project lasts. Hereby, the factor gradually decreases over time from initially four to zero at the end of the project's life cycle, forming a cone-like structure.

As a consequence, it once again has to be stated, that effort estimations, normally done at early stages of the project life cycle, will contain a certain inaccuracy factor. Moreover, when it comes to specifying test requirements (test cases, test steps, etc.) that are structured and designed according to the (incomplete) functional requirements specifications, the problem of uncertainty unveils itself once again. Therefore, a whole set of test effort estimation methods exist, that try to accurately estimate to amount of work needed (i.e. the costs), to thoroughly test a system, based on the functionality initially specified. The upcoming section focuses on the crucial aspect of test effort estimation by elaborating on a distinct set of test effort methods commonly used in practice.

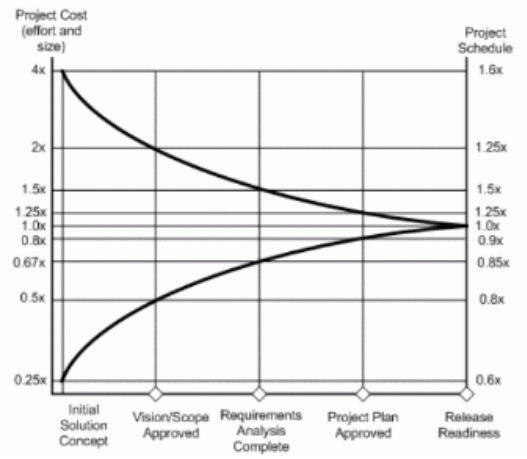

Figure 2: Cone of Uncertainty [Coo03]

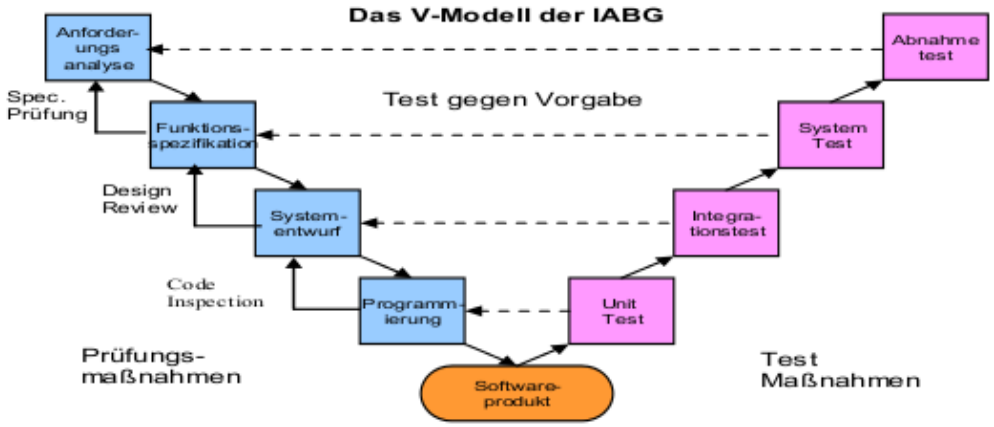

Figure 3: The V-XT Model (reproduced from [Sne07])

2 Test Effort Estimation Methods

Especially in the field of software engineering, the process of testing plays a crucial part to ensure a high level of quality [AB07]. According to [Rub95], testing activities make up 40% of the total software development effort, as depicted in Figure 4. The fact, that conventional estimation techniques tend to put a large focus only on the actual development effort instead of including the broad range of additional activities involved, such as the crucial process of testing the software, adds further complications to achieve accurate overall effort estimations. According to [Nag01], this problems is based on the fact, that when techniques for estimating development effort evolved, the concept of estimating test-engineering time was completely overlooked.

Independent of the size of the software to be tested, certain preconditions must be met in order to be able to design appropriate test cases to achieve a (guaranteed) level of code coverage. Thus, as already mentioned in section 1, requirement specifications must be as detailed as possible to ensure a high level of testability. Furthermore, in order to be able to estimate the effort needed to design, implement and execute these tests special methods are needed. This section is dedicated to discuss a distinct selection of test effort estimation

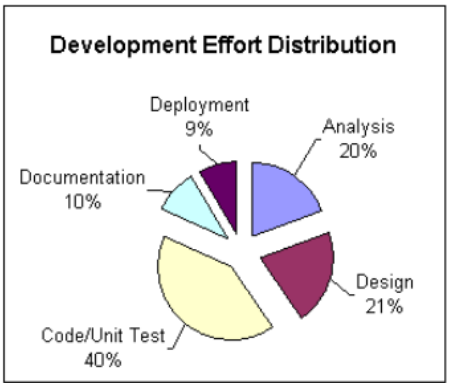

Figure 4: Effort Distribution in Software Projects [Nag01]

methods commonly used in practice. First, *function point analysis* (FPA) will be discussed, followed by *test point analysis* (TPA), which uses metrics generated by function-point analysis and finally *use case-points* (UCP).

2.1 Function Point Analysis

Since its publication, function point analysis has proven to be a reliable method for measuring the size of software, thus providing test designers, as well as project managers with a solid tool to estimate the efforts needed. According to [Hel03], apart from purely measuring its output (i.e. un-/adjusted function points), FPA additionally is extremely useful in estimating project efforts, managing change of scope, measuring productivity, as well providing means for communicating functional requirements.

In contrast to other test effort estimation methods, FPA was initially designed to be used from the user's point of view. The designers had realized, that the best way to gain an understanding of the users' needs was to approach the problem from their perspective and how they viewed and interpreted the results an automated system produces [Hel03]. Consequently, FPA's evaluation mechanism is based upon the ways users interact with the system's capabilities. Hereby, software systems provide users with a distinct set of five basic functions, in order to do their jobs. These functions are separated into two different categories:

1. Data Functions
2. Transactional Functions

Whereas data functions address the user's data requirements, that in return can be divided into the sub-categories internal logical files (ILF) and external interface files (EIF), transactional functions cope with the user's need to access data through either external inputs (EI), external outputs (EO) and external inquiries (EQ) [Hel03]. Apart from these five functional components there exist two so-called adjustment factors needed to calculate the actual FP, the first being the functional complexity and the second the value adjustment factor (VAF). The functional complexity represents the complexity of a single parameterized function (i.e. a test step), that is determined by a combination of the amount of data fields and their corresponding data types [Pil10], as shown in Figure 6.

Thus, for every of the five basic function types described above (ILF, EIF, EI, EO, EQ) there exists a complexity matrix, consisting of the distinct combination of its input parameters, resulting in a complexity rating ranging from low to high. The sum of the individual functional complexity based on these calculations, gives the so-called sum of unadjusted function points (UFP), as shown in Figure 5. Finally, using the

step type	complexity	X	o		X			X		
	simple			medium			complex			sum
External Input	o	3	0	6	4	24	o	6	0	24
External Output	o	4	0	o	5	0	o	7	0	0
External Inquiry	o	3	0	o	4	0	o	6	0	0
External Interface File	o	5	0	o	7	0	o	10	0	0
Logical Internal File	1	7	7	o	10	0	1	15	15	22
									UFP	46

Figure 5: Sample Calculation of UFP in FPA [Pil10]

		amount of distinct data fields**		
		... in data input		
data / step*		< 5 data fields	5-15 data fields	> 15 data fields
	< 2 data types	simple	simple	medium
	2 data types	simple	medium	complex
	> 2 data types	medium	complex	complex
		... in data output and queries		
data / step*		< 6 data fields	6-19 data fields	> 19 data fields
	< 2 data types	simple	simple	medium
	2-3 data types	simple	medium	complex
	> 3 data types	medium	complex	complex
		... in data base and interfaces		
groups per step***		< 20 data fields	20-50 data fields	> 50 data fields
	< 2 field groups	simple	simple	medium
	2-5 field groups	simple	medium	complex
	> 5 field groups	medium	complex	complex

* data base (file types referenced – FTR)
** data fields (data element types – DET)
*** field groups (record element types – RET)

Figure 6: Functional Complexity Rating Schema in FPA [Pil10]

VAF, that considers a system's technical, as well as operational characteristics, the FP can be calculated, using the following formula: $FP = UFP * VAF$. Hereby, the VAF itself consists of 14 individual factors, also called the total degree of influence (TDI) [Pil10], which amongst others are represented by data communication, distributed functions, performance and the complexity of data processing.

Unfortunately, FPA also has its drawbacks. According to [Nag01], one of FPA's crucial pitfalls is the fact, that it needs very detailed requirements to produce accurate results. As already mentioned earlier, requirement specifications tend to not be as complete as they should be. Thus, this leaves room for further improvement.

In his paper, [Pil10] suggested a three spreadsheet approach to estimate the test automation effort using FPA and TPA. Whereas the first two spreadsheets are used to collect all required test steps, in order to determine the rate of test step re-usability, as well as to actually calculate the resulting FP respectively, the third sheet uses the data from the previous FPA to determine the TP. The upcoming section 2.2 will discuss in TPA in greater detail.

2.2 Test Point Analysis

Test point analysis (TPA) represents a test estimate preparation technique that can be used to objectively prepare estimates for system- and acceptance tests [vVCD99]. However, it is important to note that TPA itself only covers black-box testing. Thus, it is often used in conjunction with FPA, that in return does not cover system- and acceptance tests. Consequently, FPA and TPA merged together provide means for estimating both, white- and black-box testing efforts. Furthermore, according to [vVCD99], TPA also offers the possibility to determine the relative importance of the functions to be tested, so that the available testing time can be used as efficiently as possible.

As mentioned earlier in section 2.1, [Pil10] used a three spreadsheet approach incorporating FPA and a consecutive TPA to estimate test automation efforts. Based on the metrics derived from FPA (UFP, VAF and finally FP), [Pil10] then was able to determine the corresponding amount of TP. Generally speaking, the factors involved in determining the TP are rather wide-ranging. Thus, there are a lot of dependent and independent factors that need to be taken into account. Figure 7 depicts the primary factors and steps involved in the procedure of calculating the amount of TP, which eventually leads to the total amount of test hours estimated. Since TPA is based upon the principles of a black-box test, formulating an estimate requires knowledge of three basic elements [vVCD99]:

1. Size of the software system
2. Test strategy
3. Level of Productivity

Basically, out of the first two factors the volume of test work required can be calculated, expressed in TP. Through a simple multiplication using the third factor, the level of productivity, the estimated amount of hours needed can be determined. Using TPA, the size of a software system is approximately equivalent to the amount of FP previously calculated through FPA. Nevertheless, again certain factors need to be taken into account, that have alternating influences on the testing size [vVCD99], as opposed to the amount of FP. These factors are *complexity*, *interfacing* and *uniformity*.

The second factor, the *test strategy* specifies which quality characteristics are going to be tested for each function, as well as the degree of coverage. Hereby, more important functions and characteristics are going to be tested more intensively, thus increasing the testing volume as such. Therefore, it is important to note, that [vVCD99] recommends to determine the importance of these characteristics in conjunction with the client to achieve a maximum level of testing conformance. [vVCD99] further differentiates between two factors that influence the thoroughness of testing in accordance with the client:

1. User-importance
2. User-intensity

Whereas the factor *user-importance* denotes the level of significance of a particular function, the *user-intensity* describes its level of usage. Thus, whereas a function that is going to be used throughout the day is characterized as more important by the user, a function that is used very rarely will not be marked a high priority. The same principle can be applied to the user-intensity factor.

Finally, the level of productivity is defined as the amount of time needed to realize a single test point, as determined by the test strategy and the system's size. Hereby, productivity itself is composed of two components, the *productivity figure* and the *environmental factor*, as shown in Figure 7. The productivity factor is specific to the individual organization, represented by the knowledge and skill of the test team.

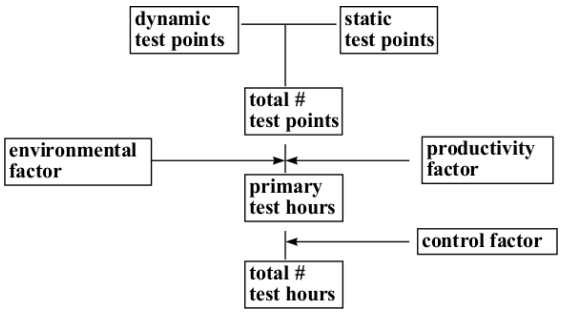

Figure 7: Overview of Test Point Analysis procedure [vVCD99]

On the other hand, the environmental factor describes the external influence of the environment on the test activities, including the availability of test tools, the level of the team's experience with the test environment, the test basis' quality as well as the availability of testware.

Thus, the overall procedure behind TPA, as depicted by Figure 7, is based upon the following steps [vVCD99]. First, the amount of *dynamic test points* have to be calculated, as the sum of the TP assigned to all functions. Hereby, TP are calculated for each individual function using the amount of FP, function-dependent factors (user-importance, user-intensity, complexity, uniformity and interfacing), as well as quality requirements. Secondly, the *static test points* are a result of determining the number of test points required to test static measurable characteristics. The total amount of TP is the sum of dynamic and static TP.

Following, the amount of primary test hours can be calculated using the following formula: $\sum(TP) * environmental factor * productivity factor$. According to [vVCD99], the primary test hours represent the volume of work required for the primary testing activities, represented by the amount of time required to complete the preparation, specification, execution and completion test phases. Finally, in order to get to the total amount of estimated test hours, the volume of work needed for additional management activities needs to be taken into account. The magnitude of these secondary test activities primarily depends on the availability of management tools, as well as the size of the test team. All in all, [vVCD99] states, that the resulting total amount of test hours represents an estimate of the total time needed for the entire set of testing activities, excluding process of formulating of the test plan.

Concluding, it has to be said, that since TPA is based on the results of FPA, it also inherits its drawbacks. As already discussed in section 2.1, FPA requires thoroughly detailed requirement specifications to produce accurate results. Furthermore, according to [Nag01], modern object-oriented systems are often designed with use cases in mind. Consequently, FPA and TPA cannot be used in these cases. This is were *use case points* (UCP) come into play. This test effort estimation method will be discussed in the upcoming section 2.3.

2.3 Use Case Points

In contrast to FPA and TPA, test effort estimation using UCP is based upon use cases (UC). [Coc00] defines UC as a system's behavior under various conditions, based on requests from a so-called "primary stakeholder", i.e. one of the stakeholders. Thus, UC capture contractual agreements between these stakeholders about the system's behavior. Hereby, the main goal is to honor and protect the interests of all stakeholders involved. Furthermore, by collecting all possible behavioral sequences, based on requests submitted to the system, UC also capture the wide range of *scenarios* possible. So, according to [Coc00], UC basically represent what users want from a system. [Nag01] states, that the V-model should be the life cycle model of choice, due to the fact, that for each development activity it clearly associates a corresponding

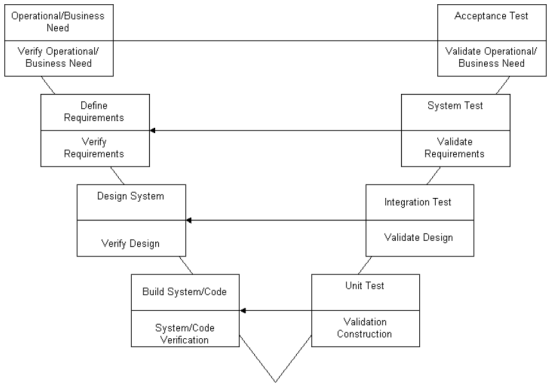

Figure 8: The V-Model [Nag01]

testing activity, as shown in Figure 8. Thus, subsequent steps involved in this model ensure that the required test documentation is complete and comprehensive. According to [Nag01], when developing an estimation method that is placed after the *operational/business needs* have been defined (see Figure 8), it becomes obvious that UC will serves as its corresponding input.

Thus, the primary task of UCP is to map *use cases* (UC) to *test cases* (TC) [Nag01]. Hereby, each scenario together with the corresponding exception flow for each UC serves as input for a specific TC. Basically, the amount of test cases identified through this mapping results in the corresponding test effort estimation. According to [Nag01], UCP is comprised of six basic steps that determine a project's required test effort:

1. Calculate unadjusted actor weights (UAW)
2. determine unadjusted UC weights (UUCW)
3. compute unadjusted UC points (UUCP)
4. determine technical and environmental factors
5. calculate adjusted UCP (AUCP)
6. compute final effort

The first step takes care of calculating the UAW, which is the sum of all actors multiplied by the so-called corresponding *actor weights*, based on the actor type, as shown in Figure 9a, using the formula $UAW = AW * actortype$. Hereby, actor types range from simple to complex. Secondly, the amount of UC need to be identified in order to determine the corresponding UUCW, which is represented as the sum of all UC multiplied by a weight factor depending on the number of transactions or scenarios, using the following formula: $UUCW = \sum(UC * UCW)$. Afterwards, based on the UAW and UUCW, the UUCP can be calculated using $UUCP = UAW + UUCW$. Based on the technical complexity factor (TCF) listed in Figure 9c, the so-called technical and environmental factors (TEF) can be determined, as the sum of the products of weights and *assigned values* (giving the *extended value*): $TEF = \sum(W * AV)$. In the fifth step, the AUCP is calculated based on the formula: $AUCP = UUCP * (0.65 * (0.01 * TEF))$. Finally, the estimated test effort using UCP can be calculated as a multiplication of the AUCP with a *conversion factor*: $Effort = AUCP * conversionfactor$. According to [Nag01], the conversion factor represents the test effort required for a language/technology combination, such as planning, writing and executing tests for a single UCP using Enterprise JavaBeans.

Actor Type	*Description*	*Factor*
Simple	GUI	1
Average	Interactive or protocol-driver interface	2
Complex	API / low-level interactions	3

(a) UCP Actor Weights (AW)

Use Case Type	*Description*	*Factor*
Simple	<=3	1
Average	4-7	2
Complex	>7	3

(b) UCP Use Case Weights (UCW)

Factor	*Description*	*Assigned Value*
T1	Test Tools	5
T2	Documented inputs	5
T3	Development Environment	2
T4	Test Environment	3
T5	Test-ware reuse	3
T6	Distributed system	4
T7	Performance objectives	2
T8	Security Features	4
T9	Complex interfacing	5

(c) UCP Technical Complexity Ratings

Figure 9: UCP Factors (reproduced from [Nag01])

3 Conclusion

This paper focused on the evident problems of estimating the (test) effort needed for software engineering projects. Due to the increasing number of fixed price projects, accurate requirement specifications are becoming even more important, in order to ensure a project's economic survival. By using the V-XT model as an example, it has been made clear why requirement specifications that have been developed at very early stages in the project's life cycle, will always contain a certain amount of uncertainty, that gradually decreases as the project progresses, resulting in a "cone of uncertainty".

Furthermore, this paper also discussed a selection of test effort estimation methods commonly used in practice. Hereby, a special focus was put on function point analysis, test point analysis and finally use case points. Although these methods may use different approaches, in order to determine the effort estimations, they all share one common prerequisite: a certain level of information accuracy, onto which the effort calculation will be based. Thus, they all will produce more or less inaccurate results, independent of their complexity, once the requirement specifications are incomplete.

References

[AB07] Eduardo Aranha and Paulo Borba. An estimation model for test execution effort. *Empirical Software Engineering and Measurement, International Symposium on*, 0:107–116, 2007.

[BD08] Manfred Bundschuh and Carol Dekkers. *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement*. Springer, Berlin, 1 edition, November 2008.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[Coo03] Microsoft Coorporation. Cone of uncertainty, 2003. http://www.microsoft.com/china/technet/images/itsolutions/techguide/innsol/images/msfpmd07.gif.

[Hel03] Roger Heller. An introduction to function point analysis, 2003. http://www.qpmg.com/fp-intro.htm.

[LPM03] Sr. Larry Putnam and Ware Myers. *Five Core Metrics: The Intelligence Behind Successful Software Management*. Dorset House Publishing Company, Incorporated, 1 edition, May 2003.

[Nag01] Suresh Nageswaran. Test effort estimation using use case points. *Technology*, (June), 2001.

[Pil10] Michael T. Pilawa. Metrics for test automation effort estimations. *Testing Experience, The Magazine for Professional Testers*, 11(1):22–27, 2010.

[Rub95] Howard A. Rubin. Worldwide benchmark project report, 1995.

[Sne07] Harry M. Sneed. Die Bedeutung der Projektaufwandsschätzung für den Festpreisprojekterfolg. *Rundbrief der Gesellschaft für Informatik e.V.*, 2007(1):55–64, 2007.

[vVCD99] Drs. Erik P.W.M. van Veenendaal CISA and Ton Dekkers. Testpointanalysis: a method for test estimation, 1999. www.ifpa.nl/Images/ESCOM1999%20Test%20Point%20Analysis%20-%20A%20method%20for%20test%20estimation_tcm158-35882.pdf.