1-1-2008

Estimating Agile Software Project Effort: An Empirical Study

Lan Cao
Old Dominion University, lcao@odu.edu

Estimating Agile Software Project Effort: An Empirical Study

Lan Cao
Old Dominion University
lcao@odu.edu

ABSTRACT

This paper describes an empirical study of effort estimation in agile software development. Estimated effort and actual effort of a 46-iteration project are collected and analyzed. The results show that estimation in agile development is more accurate than that in traditional development even though agile developers still underestimate the effort. However, estimation accuracy is not improved over time as expected by agile communities.

Keywords

Software project effort estimation, agile software development, agile planning.

INTRODUCTION

It is long been realized that an important aspect of software development project is to know how much it will cost (DeMarco 1982). In most cases the cost factor is labor (Shepperd et al. 1997). Estimating development effort is central to the planning, management and control of a software project. Underestimate will cause schedule pressure which results in higher defects rate and development cost (Abdel-Hamid 1990). On the other hand, overestimate will cause waste of resources and hurt the competitiveness in bidding a contract.

Estimation is a challenge in every software project. Nearly 60% of all software projects are either failed or challenged with cost or time overruns reported by Standish Group CHAOS 2006 report. There are many factors that cause estimation error: lack of expertise and user inputs, frequent requirement changes, complexity, incomplete requirements, and users lack of understanding of requirements (DeMarco 1982; Genuchten 1991; Lederer et al. 1995; Phan et al. 1988; Subramanian et al. 1995).

Iterative and incremental development, which has been gradually adopted by many software development projects, provides quick feedback and process transparency. A study compared project overruns of flexible and sequential development reveals that projects which employ a flexible development model experience less effort overruns than do those which employ a sequential model (Molokken-Ostvold et al. 2005). As a result, cost estimation accuracy has been largely improved over the years (Rubinstein 2007). The recent Standish Group CHAOS report reveals the large improvement in average overrun (43% in 2002, 189% in 1994). However, effort prediction in agile software development is more challenging as there is usually no upfront requirement analysis phase (Ramesh et al. 2008). The lack of detailed analysis and specification of requirements makes it more difficult to identify tasks in the beginning of a project. This very nature of agile development dictates that underestimation should be much higher than that in traditional approaches. In agile development, agile planning continuously adjust the estimates throughout the development process (Cohn 2005), while traditional projects estimate at the planning stage. The impact of agile planning on the accuracy of effort estimation remains unknown. The main purpose of this research is to investigate the effort estimation in agile software development. Especially, I try to answer the following research questions: 1) *How accurate is effort estimation in agile approaches? 2) Is effort estimation in agile approach getting more accurate over time?* 3) *Are effort estimation of bugs and stories the same?*

To answer these questions I conducted a study in effort estimation in a large-size agile project. Estimated effort and actual effort of 46 two-week iterations were collected and analyzed. The remaining part of the paper is organized as follows: Section 2 describes the previous work on software project estimation and why estimation is different in agile projects. Section 3 describes research design. The results are presented in section 4 and finally, section 5 concludes and describes the future work.

RELATED WORK

Estimation Approaches

Over the past years, many effort estimation models have been developed such as COCOMO, Function points or other algorithm driven methods (e.g. Albrecht et al. 1983; Bailey et al. 1981; Basili et al. 1981; Boehm 1981; Conte et al. 1986; Kemerer 1987; Matson et al. 1994; Putnam 1978; Shepperd et al. 1997; Walston et al. 1977). Heemstra (1992) provides a detailed overview of some software effort estimation models prior to 1992. Here we only briefly discuss COCOMO (Boehm 1981) which estimates the effort as the function of size and productivity factors. The COCOMO model was developed based on a regression analysis of 63 completed projects. COCOMO relates the effort required to develop software (person-months) to project size.

$$\text{Effort} = a\text{Size}^b$$

Where size is typically measured as lines of code (LOC), b is a productivity parameter and a is an economies or diseconomies of scale parameter. The development environment such as people, process and product factors is represented in terms of 15 cost drivers that can be adjusted to explain the project variations. The COCOMO detailed model divides the project into four phases (product design, detailed design, coding/unit test, and integration test). The 15 cost drivers are different in each phase, and COCOMO estimates the cost of each phase separately.

Estimation models have been reported to have high error rates in estimating software development effort. For example, one study reported an average of 600 percent error rate using COCOMO (Kemerer 1987). Function points (Albrecht et al. 1983) is also reported with mixed results (Jeffery et al. 1993a; Jeffery et al. 1993b; Kemerer 1987; Matson et al. 1994). The major reason is that models were developed based on specific projects and were hard to generalize to other environments. For example, one study found that these models were only valid within the organizations in which they were developed (Basili et al. 1981). Another study on the factors impacting estimation found that variable "company" accounts for 68% of the variance of productivity (Maxwell et al. 1999). So companies need to establish their own software matrix for accurate estimation. Even within one company, there are many factors influence productivities: people, process, product, technology, etc. However, research has found that including a large number of factors into the model was not very helpful to increase the estimation accuracy (Krichenham 1992). One reason was that the interplays among the factors were not captured in the model. The adjustment of these factors was treated as if they were independent even though many of them were intertwined with each other.

Beyond the formal models, there are other approaches such as case-based, rule-based, expert-based estimation and analogy (e.g. Jorgensen 2004; Mukhopadhyay et al. 1992; Shepperd et al. 1997). Among them, the expert-based estimation is the dominant strategy used in the industry (Heemstra 1992; Molokken et al. 2003). The expert-based estimation includes intuition/experience and expert judgment supported by historical data, process guideline and checklists (Jorgensen 2002). Many studies suggest that experts have the same or better accuracy as the formal models (Jorgensen 2002).

Estimation in Agile Software Development

Agile effort estimation is different from that in traditional software development. An important source of estimation difficulty in agile development is requirements volatility. There are two major sources of requirements volatility: the probability of a change in requirements in the future, and the vagueness of the requirements (Savolainen et al. 2001). Users rarely have well defined needs, least of all in the early stages of the product's development. So instead of complete and accurate requirements, users' concepts of the problem evolve during the development process (Mrenak 1990). In agile development, users keep requesting changes throughout the development process. As a result, agile project scope is continuously adjusted throughout the project. New tasks are discovered, as customers keep requiring new features. Tasks due to underestimation are discovered during development. Also, planned features are removed or deferred because of the schedule constraints. Unlike in the traditional approach where most of the new tasks are discovered during the detailed design phase, in agile development new tasks are discovered in each iteration. For example, in a project involving 30-iterations, less than 50% of the tasks were found during the exploration and analysis phase, 25% new tasks were found during the next ten iterations, and 25% were found in the last ten iterations (Fuqua et al. 2003). This amounts to serious underestimation of the effort (100% to be exact).

Another distinct characteristic of agile development is the short feedback loop. Traditional projects estimate at the beginning of the project based on initial plans. Agile projects don't use traditional upfront plans. Agile planning includes a release plan and a series of iteration plans. A release plan includes a list of the key features that will be developed for the next release. An iteration plan is a tactical development plan for a specified period (usually 2 to 4 weeks). The iteration plan takes the highest priority features, and based on developer estimates, selects a set of them to work on for the iteration. Each story is broken

down into tasks by the development team. Tasks can then be individually estimated, and developers can sign up to work on them. Project managers can now use the iteration cycle to track progress in terms of completed features (stories), and outstanding tasks. According to agile planning literature (Beck 2000; Beck et al. 2001), "yesterday's weather" is an excellent source for predicting today's weather. i.e., data from a preceding iteration predicts how much work can be finished of the coming iteration. This adjustment happens at the beginning of each iteration. An estimate of velocity (similar to productivity) is adjusted for each iteration based on the actual velocity achieved in the iterations that have been completed until then. The data used in making this estimate are the actual work time and the number of stories implemented in each iteration. It is claimed that the continuous adjustment quickly improves the agile estimation, and estimation is quite accurate after a few iterations (Cohn 2005).

This statement aligns with the conventional wisdom that the estimates improve as a project process. The uncertainty decreases significantly as the developers obtain more knowledge about the project. In agile software development, it is claimed that as the development progresses, the estimates gets very accurate after the first few iterations due to the cumulative effect of learning (Beck 2000; Beck et al. 2001; Cohn 2005). However, there is no study testing this claim.

Ideally team velocity is kept stable over the iterations. However, a field report shows that velocity fluctuates with a 20% variation (Fuqua et al. 2003). Unstable velocity makes the estimation difficult. More interestingly, the developers commonly worked with the assumption that their velocity was stable, without carefully collecting and analyzing relevant data.

RESEARCH METHOD

This project is a desktop support system for both Windows® and Macintosh ®. The development team followed eXtreme Programming (XP) method. The project lasted 46 two-week iterations (460 working days). The team had 15 developers working on the project during the first 34 iterations. From iteration 35 to 46, many developers were removed from this project and only 5 developers stayed (including a new developer who joined the project from iteration 35).

Data Item	Description	Unit	Comments
Priority	Priority of the story		
Title	Title of the story		
Name	Short description of the story (not included in table 3)		
Platform	w -- Windows m – Macintosh		
Owner	The owner of the story – the developers who are assigned to work on this story		
Initial Story	If it is an initial story	0—no 1---yes	
Initial Est	The estimated time for a story	Ideal hour	Only available for iteration 1-6
Initial Task Est	The total time for all the initial tasks of a story	Ideal hour	Derived from task level. Sum of Task Est for initial tasks only (Table 2)
Total Task Est	The total time for all the tasks (including initial and non-initial tasks) of a story	Ideal hour	Derived from task level. Sum of Task Est for all tasks (Table 2)
Actual Time	The total actual time for all tasks	Ideal hour	Derived from task level, sum of Task actual time (Table 2)
EC	Engineering complete---the story is considered complete by the owner	0 --- no 1--- yes	
CC	Customer complete—the story is considered complete by the customer	0 --- no 1--- yes	

Table 1. Data Items at Story Level

Data Item	Description	Unit	Comments
Priority	Priority of the task		
Task	Code of the task		
Name	Short description of the task, not included in table 2		
Initial Task	If it is an initial task		
Owner	The owner of the task – the developers who are assigned to work on this task		
Initial Est	The estimated time for a task	Ideal hour	Some data are missing
Actual Time	The actual time for a task	Ideal hour	Some data are missing
Complete	If the task is complete	0 --- no 1--- yes	

Table 2. Data Items at Task Level

The data collected is at two levels: story level and task level. The data includes estimates and actual effort data on more than 1000 stories (about 6900 tasks) in 46 iterations. The story level data includes all the stories and bugs planned for each iteration. Some estimated and actual effort data at story level are derived from data at task level, which contains the basic information, estimation and actual effort of each task. Table 1 summarizes the data items at story level and table 2 summarizes the data items at task level.

Table 3 is the example of story level data (iteration 8). Table 4 is the example of task level data (story 70m of iteration 8). The two examples also show that some estimation and actual data are missing. The missing data points at task level results in the inaccuracy of data that are derived from them at story level. In this example, among the 23 tasks, 2 tasks missed both estimated and actual time data, 7 tasks missed actual time data, and 2 tasks missed estimated time data. As a result, in story level (table 3) for story 70m, the initial task estimation (77), total task estimation (85) and actual time (67) were not accurate. It is hard to compare the estimated and actual effort of each story.

To deal with the missing data issue, I removed all the tasks that have missing data on either task estimation or actual time. For example, after the 11 tasks with missing data are removed, the initial task estimation is still 69, the total task estimation is 69 and the actual time is 62. There are about 4000 tasks left after all tasks with missing data were removed. Then I used the average size of the stories as the data unit for data analysis.

Some iterations were refactoring iterations during which no new features were developed. I removed the refactoring iterations, as a result, there are 31 iterations that involve new story development and 35 iterations that involve bug fixing.

In this research, the accuracy of estimation is measured as mean magnitude of relative error (MMRE).

$$\text{MMRE}= \sum_{i=1}^{i=n}\left(\frac{|\, actual - etimated \,|}{actual}\right)\frac{100}{n}$$

Where n is the number of observations.

Priority	Title	Platform	Owner	Initial Story	Initial Est [ideal hours]	CC	EC	Initial Task Est [hours]	Total Task Est [hours]	Actual Time [hours]
1	Story 87w	w	HT	0	0	1	1	8	8	13.50
2	Story 100m	m	DH	0	0	1	1	18	21	26.00
3	Story 100w	w	DH	0	0	0	0	0	0	6.00
4	Story 89m	m	KM	0	0	1	1	0	0	4.00
5	Story 90w	w	JC	0	0	1	1	0	0	7.00
6	Story 102	m/w	CP	0	0	1	1	0	0	12.00
7	Story 70m	m	DH	0	0	0	0	77	85	67.00
8	Story 70w	w	DH	0	0	0	0	8	8	7.00
9	Story 94m	m	BK	0	0	0	0	14	14	13.00
10	Story 94w	w	JRC	0	0	1	1	4	4	1.00
11	Story 22w	w	FZ	0	0	0	1	11	11	12.00
12	Story 99m	m	TR	1	0	0	0	87	93	102.00
13	Story 99w	w	MK	1	0	0	0	32	45	40.50
14	Story 108m	m	MR	1	0	1	1	2	2	5.00
15	Story 108w	w	MK	1	0	1	1	0	0	0.00
16	Story 109m	m	MR	1	0	1	1	0	0	2.00
17	Story 109w	w	MK	1	0	1	1	0	0	0.00
18	Story 90m	m	FZ	1	0	0	0	7	9	8.00
19	Story 69m	m	ZN	1	0	0	0	26	27	2.50
20	Story 69w	w	ZN	1	0	0	0	11	11	4.00

Table 3. Example Data at Story Level of Iteration 8

Priority	Task	Initial Task?	Initial Est [ideal hours]	Owner	Actual Time [hours]	Complete?
7.1	A	1	6	SC	8	0
7.2	B	1	24	MR	3	0
7.3	C	1	5	JC	8	1
7.4	D	1	2	JC	4	0
7.5	E	1	10	SC	19	1
7.6	F	1	2	JC	2	1
7.7	G	1	6	DH	7	1
7.8	H	1	2	JC		
7.9	I	1	1	JC	1	1
7.10	J	1	1	DH		
7.11	K	1	1	SC		
7.12	L	1	1	DH		
7.13	M	1		JP		
7.14	N	1	2	CP	1	1
7.15	O	1	3	CP	1	1
7.16	P	1	4	CP	4	1
7.17	Q	1	4	CP	4	1
7.18	R	1	3	MR		
7.19	S	0		CP	1	1
7.20	T	0		JC	4	0
7.21	U	0				
7.22	V	0	2			
7.23	W	0	6			

Table 4. Example Data at Task Level of Story70m in Iteration 8

RESULT

Actual versus Estimates

Figure 1 and 2 show the actual vs. estimates of effort on stories and bugs, respectively. The solid lines show that the actual equals estimates. The data shows that underestimates (actual >estimate) is more frequent and severer than overestimate for both stories and bugs. However, bugs are more difficult to estimate as the actual efforts are significant greater than the estimates.

The results show the similar distribution scatter of the traditional projects (Demarco 1982, Little2006). Breaking a big task into smaller pieces seems not totally solving estimation problem. Developers still underestimate the effort ate story/task level.

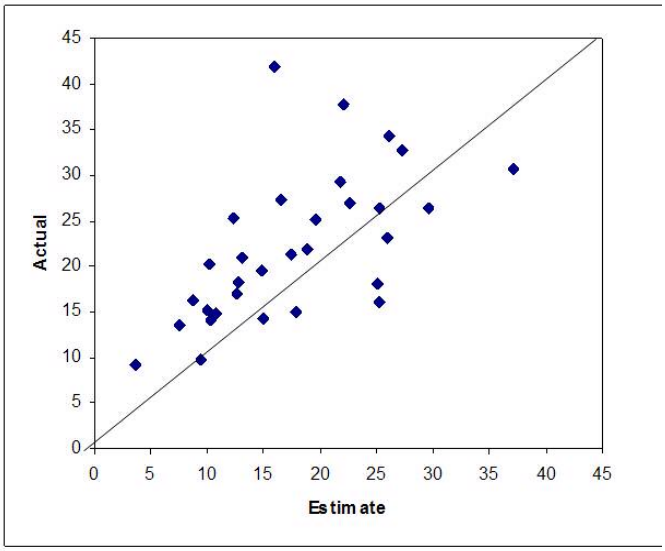

Figure 1. Estimated Effort vs. Actual Effort in Ideal Hours (Stories)

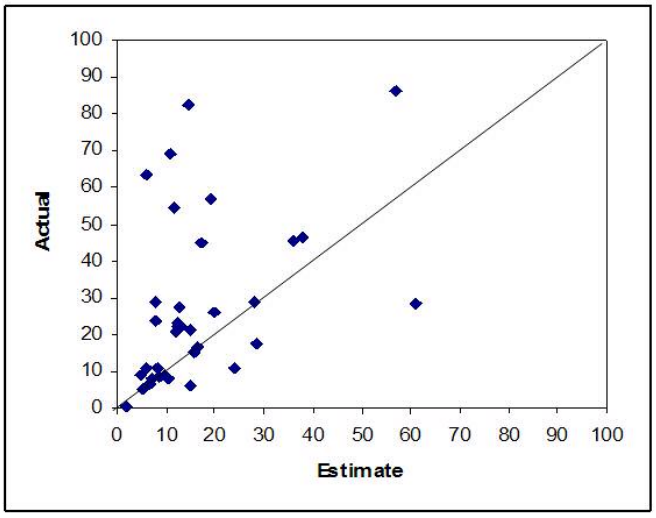

Figure 2. Estimated Effort vs. Actual Effort in Ideal Hours (Bugs)

MMRE over Time

Figure 3 shows the data of MMRE at all iterations (the data on iteration 14 and 43 are missing).

First, Figure 3 shows that on average bugs have bigger MMRE than stories. The effort of fixing a bug is more unpredictable than the effort of implementing a new story. The average MMRE is 19% for stories and 28% for bugs.

Second, the effort estimation is not improved over time as agile methods claimed. This results confirms the findings of a recent study on non-agile projects (Little 2006), which found that the uncertainty range was nearly identical throughout the project stages. This tells us that other factors such as resource availability, requirements change, insufficient refactoring, lack of unit tests, communication problems, and distractions may have significant impacts on the estimation. The continuous adjustment and quick feedback loop will not eliminate the uncertainty.

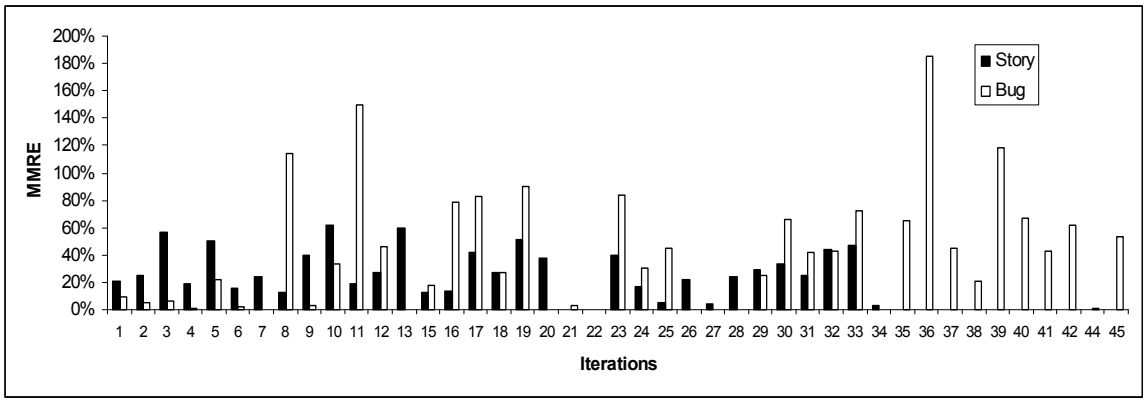

Figure 3. MMRE of Each Iteration

Accuracy

Figure 4 shows the cumulative probability distribution as the lognormal distribution of the ratio of actual to estimated effort of stories (the cumulative distribution for bugs is similar). This curve shows the percentage of data points that have a value lower than the value of actual/estimate. For example, about 25% of the iterations have actual that were lower than the initial estimation (ratio =1.0).

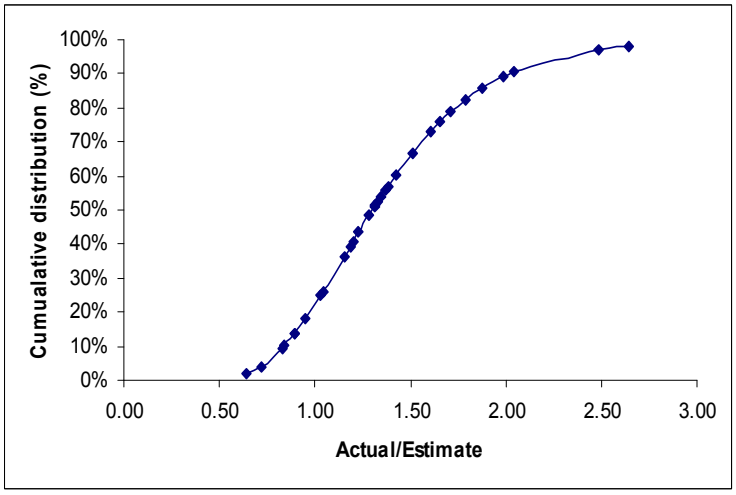

Figure 4. Cumulative Distribution (Stories)

The lognormal distribution tells us that the factors impacting estimation are interdependent to each other. To identify those factors and their relationships is critical for accurate estimation. The S-curve confirms the previous studies (DeMarco 1982; Little 2006), but it has significant narrower range of the x-axis which is the ratio of actual to estimate. Furthermore, the ratio of P90 (90 percent confidence of the target being met) to P10 (10 percent confidence of the target being met) is 2.42, which is lower than the reported data on traditional projects (Table 5). That suggests that agile estimate is more accurate than that of traditional development. This result is confirmed with the findings of a previous study (Molokken-Ostvold et al. 2005).

Studies	P90/P10	Range of ratio (actual/estimate)
This study	2.42	0. 6- 2.6
Little (2006)	3.25	About 0.7-9.0
Demarco (1982)	5.2	About 0.8-8.0
Boehm (1981)	4.0	NA

Table 5. Comparison of Estimate Accuracy

CONCLUSIONS

Effort estimation in agile software development is studied in this research based on data collected from a large agile project. The results show that estimation in agile development is more accurate than traditional approaches but certain degree of underestimation still exists. The study also shows that the estimation is not improved over time as claimed in agile literature. The study contributes to both the research on software estimation and agile software development.

The quick feedback from previous iterations and the transparency of development process in agile methods enables continuous adjustment of the estimation during the development process. Moreover, the estimation in agile projects is at task/story level, which is small in size. The continuous adjustment and small task size improve the overall estimation accuracy. However, confirmed with the findings of a previous study on 106 non-agile projects (Little 2006), it is interesting to find that the estimation is not getting better overtime during the development process, as claimed in agile methods. This tells us that the impact of other factors on the estimation process is significant, which needs to be investigated in the future research. More data needs to be collected from projects in other organizations to validate the results.

REFERENCES:

1. Abdel-Hamid, T.K. "Investigating the cost/schedule trade-off in software development," *IEEE Software* (7:1) 1990, pp 97-105.
2. Albrecht, A.J., and J. E. Gaffney, J. "Software function, source lines of code, and development effort prediction: a software science validation," *IEEE Transactions on Software Engineering* (SE-9:6) 1983, pp 639-648.
3. Bailey, J.W., and Basili, V.R. "A meta-model for software development resource expenditures," *Proceedings of Fifth International Conference on Software Engineering*, San Diego, CA, 1981, pp. 50-60.
4. Basili, V.R., and Freburger, K. "Programming measurement and estimation in the software engineering laboratory," J*ournal of Systems and Software*, 2, 1981, pp 47-57.
5. Beck, K. Extreme Programming Explained: Embrace Change Addison-Wesley, Boston, 2000.
6. Beck, K., and Fowler, M. Planning Extreme Programming Addison Wesley Longman, New York, NY, 2001.
7. Boehm, B.W. Software Engineering Economics Prentice-Hall, Englewood Cliffs, NJ, 1981.
8. Cohn, M. Agile Estimating and Planning Prentice Hall PTR 2005.
9. Conte, S.D., Dunsmore, H.E., and Shen, V.Y. Software Engineering Metrics and Models Benjamin/Cummings Publishing Co, Menlo Park, CA, 1986.
10. DeMarco, T. Controlling Software Projects: Management, Measurement and Estimation Yourdon Press, New York, NY, 1982.
11. Fuqua, A., and Hammer, J.M. "Embracing Change: An XP Experience Report," *XP 2003*, 2003.
12. Genuchten, M.V. "Why is software late? An Empirical study of reasons for delay in software development," *IEEE Transactions on Software Engineering* (17:6) 1991, pp 582-590.
13. Heemstra, F.J. "Software Cost Estimation," *Information Software Technology* (34:10) 1992, pp 627-633.
14. Jeffery, D.R., Low, G.C., and Barnes, M. "A Comparison of Function Point Counting Techniques," *IEEE Transactions on Software Engineering* (19:5) 1993a, pp 529-532.
15. Jeffery, R., and Stathis, J. "Specification Based Software Sizing: An Empirical Investigation of Function Metrics," NASA Goddard Software Engineering Workshop, Greenbelt, MD, 1993b.

16. Jorgensen, M. "A review of studies on expert estimation of software development effort," *Journal of Systems and Software* (70) 2002, pp 37-60.
17. Jorgensen, M. "A review of studies on expert estimation of software development effort," *The Journal of Systems and Software* (70) 2004, pp 37-60.
18. Kemerer, C.F. "An empirical validation of software cost estimation models," *Communications of the ACM* (30:5) 1987, pp 416-429.
19. Krichenham, B.A. "An empirical validation of software cost estimation models," *Communications of the ACM* (30:5) 1992, pp 211-218.
20. Lederer, A.L., and Prasad, J. "Causes of inaccurate software development cost estimates," J*ournal of Systems and Software* (31:2) 1995, pp 125-134.
21. Little, T. "Schedule Estimation and Uncertainty Surrounding the Cone of Uncertainty," *IEEE Software* (May/June) 2006, pp 48-54.
22. Matson, J.E., Barrett, B.E., and Mellichamp, J.M. "Software development cost estimation using function points," *IEEE Transactions on Software Engineering* (20:4) 1994, pp 275-287.
23. Maxwell, K., Wassenhove, L.V., and Dutta, S. "Performance evaluation of general and company specific models in software development effort estimation," *Management Science* (45:6) 1999, pp 787-803.
24. Molokken-Ostvold, K., and Jorgensen, M. "A comparison of software project overruns - flexible versus sequential development models," *IEEE Transactions on Software Engineering* (31:9) 2005, pp 754 - 766.
25. Molokken, K., and Jorgensen, M. "A Review of Surveys on Software Effort Estimation," *Proceedings of International Symposium on Empirical Software Engineering*, 2003.
26. Mrenak, G. "Evolving concepts, or why users often don't recognize the software they asked for," *Proceedings of the seventh Washington Ada symposium on Ada*, McLean, Virginia, United States, 1990.
27. Mukhopadhyay, T., Vicinanza, S.S., and Prietula, M.J. "Examining the feasibility of a case-based reasoning model for software effort estimation," *MIS Quarterly* (16:1) 1992, pp 155-171.
28. Phan, D., Vogel, D., and Nunamaker, J. "The Search for perfect project management," *Computerworld*, 1988, pp 95-100.
29. Putnam, L.H. "A general empirical solution to the macro software sizing and estimating problem," *IEEE Transactions on Software Engineering* (SE-4:4) 1978, pp 345-361.
30. Ramesh, B., Cao, L., and Baskerville, R. "Agile Requirements Engineering Practices and Challenges: An Empirical Study," *Information Systems Journal*:forthcoming,
31. Rubinstein, D. "Standish Group Report: There's Less Development Chaos Today
32. Savolainen, J., and Kuusela, J. "Volatility analysis framework for product lines," *Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context*, Toronto, Ontario, Canada, 2001.
33. Shepperd, M., and Schofield, C. "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering* (23:11) 1997, pp 736-743.
34. Subramanian, G.H., and Breslawski, S. "An Empirical Analysis of Software Effort Estimate Alterations," *Journal of Systems and Software* (31:2) 1995, pp 135-141.
35. Walston, C.E., and Felix, C.P. "A method of programming measurement and estimation," *IBM Systems Journal* (16:1) 1977, pp 54-73.