IJITKM Volume 7 • Number 1 • December 2013 pp. 108-111 (ISSN 0973-4414)

Research Challenges of Agile Estimation

Rashmi Popli, Dr. Naresh Chauhan
Assistant Professor, YMCA University of Science & Technology
Professor, YMCA University of Science & Technology
rashmimukhija@gmail.com

Abstract
Projects that are over-budget, delivered late, and fall short of user's expectations have been a common problem area for software development efforts for years. Agile methods, which represent an emerging set of software development methodologies based on the concepts of adaptability and flexibility, are currently touted as a way to alleviate these reoccurring problems and pave the way for the future of development. The estimation in Agile Software Development methods depends on an expert opinion and historical data of project for estimation of cost, size, effort and duration. In absence of the historical data and experts the previous method like analogy and planning poker are not useful. This paper focuses on the research work in Agile Software development and estimation in Agile. It also focuses the problems in current Agile practices.
Keywords: Agile Software Development, Estimation, Extreme Programming.

1. Introduction
The formation of the Agile Alliance in 2001 and the publication of the Agile Manifesto formally introduced agility to the field of software development. The Agile Manifesto presented an industry-led vision for a profound shift in software development, through 12 principles [1]. According to the Agile Manifesto, Agile methods stress values such as individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan. The main methods that fall under the Agile umbrella include XP, Scrum, Crystal, DSDM, FDD and ASD.

2. Causes of Inaccurate Estimates in Agile Software Development
An estimate is a prediction of how long a project will take or how much it will cost Estimation and planning are related topics, but estimation is not planning, and planning is not estimation. Estimation should be treated as an unbiased, analytical process and planning should be treated as a biased, goal-seeking process. An effective software estimate provides the information needed to design a workable software development plan. A software estimation process that is integrated with the software development process can help projects establish realistic and credible plans to implement the project requirements and satisfy commitments. It also can support other management activities by providing accurate and timely planning information. Perhaps the most important element in the success or failure of a project are estimates of its scope, in terms of both the time and cost that will be required and the plans based on those estimates. There is a natural erroneous tendency associated with any form of estimation primarily because "an estimate is a probabilistic assessment of a future condition" and accuracy can therefore rarely be expected in the estimation process. Various causes of inaccurate estimates in Agile software development are discussed below.

Methodology refers to the estimation process adopted and includes the steps undertaken to produce the estimate and also the means of examining and reviewing the estimates relating to past projects Estimation inaccuracy can be caused from a lack of procedures and policies on how to deal with failures and avoid repeating mistakes by learning from past experiences.

Political forces at work within a project or company can often drive estimation inaccuracy. This is usually in the form of managerial pressure to stay within or meet the estimate[2].The estimation process can be impacted negatively by these pressures resulting in time or cost constraints .When estimates are produced simply in order to satisfy managers or customers it will inevitably lead to inaccuracy.

User Communication refers to the factors relating to the customers and their changing requirements throughout a system's life-cycle. This is usually the most prominent factor in causing project estimates to be inaccurate [3, 4]. Incomplete or unclear requirements specification at the beginning of an IS development project is typically due to the fact that customers have to determine what their requirements are and they are usually unaware what the current state of the art is, or what the competition is doing . This leads to difficulty in producing a complete set of requirements and thus estimation inaccuracy is inevitable

Management Control Problems caused by Management control include management reviews, and comparison between estimates and actual. When management fails to participate in the preparation of the estimate, and does not monitor the accuracy of the estimate, this is believed to contribute to the estimate being inaccurate. Inaccuracy also occurs when management does not refer to the estimate when conducting performance reviews of estimators and other project personnel [3, 4].

Uncertainty It refers to the changing requirements of the customers, if customer change his perspective about project then it leads to inaccuracy**.** Moving back toward the beginning of the project where most estimation exercises occur one simple truth becomes apparent, knowledge dispels uncertainty. So there is need to gather better knowledge whether by leveraging history, mathematical algorithms and/or project specific information to make better estimates. Integrating agile techniques for knowledge capture in projects are tools for reducing uncertainty.

Self Knowledge Two psychologists, Joseph Luft and Harry Ingham developed a construct to understand personal awareness. The tool named Johari's Windowiii divides personal awareness into four different categories, as represented by its four quadrants: open, hidden, blind and unknown. The lines dividing the four panes are like window shades, which can move as an interaction progresses. The concept is adaptable to teams. Team level blind spots complicate estimation, planning and ultimately performance. Techniques to improve a team's self knowledge include forming stable teams, fostering intimate communications and ensuring retrospectives actually happen often. These tools minimize what isn't known by the team and surface misunderstandings quickly.

4. Agile Approaches to Estimation

Agile estimation is categorized in expert opinion, analogy, disagreegation and planning poker. Agile methods adopt the practice of accepting last minute changes in the software through gathering the requirements in iterative and incremental manner. Some approaches of estimation in Agile are as below.

Velocity Measurement: The requirements normally exist in the form of user-stories [Beck, 2006]. A user story describes some feature or other piece of work. Story point assigns a relative size to each story and is used to estimate the velocity of project also. Velocity is the number of developed story points per iteration. The approach described here starts with *time boxed scheduling* – since the iteration are fixed. The ideal day is the time that is fully devoted to the task without any interruption. This approach has the statistical problems. Agile estimation methods are completely human intuition based methods and hence may lead to create biased situation.

Expert Opinion: In an expert opinion- based approach, an expert comment on the size and time after investigating the stories. The expert relies on the intuition and provides estimate for agile project. Planning poker is most popular estimating technique in which the participants are developers, designers and testers. It requires the opinion of multiple experts.

Estimation by Analogy: An educated guess based on past experience of the Project Manager, and, hopefully, also based on the collective experience and knowledge of several Project Managers drawn from the results of may specific projects. Estimation by analogy starts will the selection of a similar project which acts as a central metaphor for the new project. The estimator will then decide how closely the two projects resemble each other and whether there are any mitigating circumstances that will affect the effort required to finish the project, how much the project might cost and how long it will take. Based on these differences the estimator will apply a correct factor and an estimate is created.
Structured knowledgebase of past experience: This uses the concept of estimation by analogy but builds a structured knowledgebase that is used to accumulate experience from many projects and Project Managers. The key to its success is an intelligent way of classifying and quantifying the many variables that affect the time and effort.

Parkinson: Using Parkinson's principle "work expands to fill the available volume" , the cost is determined (not estimated) by the available resources rather than based on an objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort is estimated to be 60 person-months. Although it sometimes gives good estimation, this method is not recommended as it may provide very unrealistic estimates. Also, this method does not promote good software engineering practice.

Price-to-win: The software cost is estimated to be the best price to win the project. The estimation is based on the customer's budget instead of the software functionality. For example, if a reasonable estimation for a project costs 100 person-months but the customer can only afford 60 person-months, it is common that the estimator is asked to modify the estimation to fit 60 person months' effort in order to win the project. This is again not a good practice since it is very likely to cause a bad delay of delivery or force the development team to work overtime.

Bottom-up: In this approach, each component of the software system is separately estimated and the results aggregated to produce an estimate for the overall system. The requirement for this approach is that an initial design must be in place that indicates how the system is decomposed into different components.

Top-down: This approach is the opposite of the bottom-up method. An overall cost estimate for the system is derived from global properties, using either algorithmic or non-algorithmic methods. The total cost can then be split up among the various components. This approach is more suitable for cost estimation at the early stage.

5. Summary of Previous Reviews:

Research work in Agile Estimation was done by Abrahamsson et al. [6], Cohen et al. [7], and Erickson et al. [8]. These three reports describe the state of the art and state of the practice in terms of characteristics of the various agile methods and lessons learned from applying such methods in industry.

Abrahamsson et al. [6] demonstrated how to collect metrics to measure productivity, quality and schedule estimation,cost and effort estimation for an Agile Software Development project using XP. Abrahamsson et al discussed the concept of agile development, presents processes, roles, practices, and experience with 10 agile development methods, and compares the methods with respect to the phases that they support and the level of competence that they require. Abrahamsson et al. provide evidence that agile methods are ‘‘effective and suitable for many situations and environments”.

Williams et al. [7] investigated the usage of a subset of XP [5] practices at a group in IBM. The product developed at IBM using XP was found to have significantly better pre-release and post-release quality compared to an older release. The teams using XP reported an improvement in productivity, schedule, cost and effort estimation. In addition, customers were more satisfied with the product developed using XP because the teams delivered more than what the customers had originally asked for.

Maurer et al. [8] studied the development of a web based system by nine full time employees in a small company that used XP and observed substantial productivity gains compared to their pre-XP timeframe.

Heemstra surveyed 364 organizations and found that only 51 used models to estimate effort and that the model users made no better estimate than the non-model users [10]. Also, use of estimation models was no better than expert judgment.

Finnie and Wittig applied artificial neural networks (ANN) and case-based reasoning (CBR) to estimation of effort [12]. Using a data set from the Australian Software Metrics Association, ANN was able to estimate development effort within 25% of the actual effort in more than 75% of the projects

A survey of software development within JPL found that only 7% of estimators use algorithmic models as a primary approach of estimation [12].

6. Problems with Agile Estimation methods

Today, almost no method can estimate software with a high degree of accuracy. This state of the practice is created because
(1) There are a large number of interrelated factors that influence the software development process of a given development team and a large number of project attributes,
(2) The development environment is evolving continuously.
(3) The lack of measurement that truly reflects the complexity of a software system.

There are several problems in existing estimation and tracking methods for agile software developments. First, it cannot be easily related to the time duration because story points represent the amount of work and the velocity differs from team to team. McDaid et al. [13] proposed a statistical methodology to predict the actual time needed to develop a selected functionality. As a result, they presented the probability of release completion by iteration. However, the result was highly empirical; hence, it is not sufficient for us to connect a story point to the time duration. Thus, a cost metric that is related to the time duration is needed.
Second, because a story point is a relative value, the total story point value can fluctuate with a slight variation in the baseline story point. To set the base use story, the agile team finds the simplest user story and determines story

points of other user stories based on the baseline. If the baseline story point changes, other story points also have to be changed. Thus, an absolute cost metric rather than a relative metric is necessary.
Third, after the story points and velocity are used to estimate the initial size of the project, a schedule is derived using a simple linear extrapolation. In addition, the velocity is measured at the end of iteration; it is less dynamic. In this case, it is difficult to reflect several cost factors, such as requirement changes or fluctuations in the velocity, all of which affect the software cost.
Thus, dynamic tracking for agile project is required.
In Agile environment, at the initial stage of a project, there is high uncertainty about various project attributes. The estimates produced at early stages are inaccurate, as the accuracy depends highly on the amount of reliable information available to the estimator. Agile Estimation methods may lead to the errors in case of inexperienced Agile team. Therefore, there is strong need of analyzing the factors that affect the estimation of the Agile project.

7. Conclusion and Future Work:

Research on estimation has been conducted for decades with immense quantities of models and tools produced. The Agile estimation methods are not suitable for organizations because these provide little details for justifying estimates and the estimates cannot be produced early in the life-cycle. These methods also have problems like Subjective inputs; Calibrated to past projects and may not reflect the current environment. So there is a need of some Algorithmic methods of estimation that is company specific and be suitable for Agile software development in general.

References:

[1]Angelis, L., Stamelos, I. & Moriso, M. (2001)" Building a Software Cost Estimation Model Based on Categorical Data. ",Proceedings of the 7th International Software Metrics Symposium.
[2]Lederer, A. L. & Prasad, J. (1991) "The Validation of a Political Model of Information Tradional Estimation MethodscDevelopment Cost Estimating", Proceedings of the 1991 conference on SIGCPR.
[3] Jorgensen, M. (2004b) "Top-Down and Bottom-Up Expert Estimation of Software Development Effort", Information and Software Technology, 46**,** 3-16.
[4] Jorgensen, M., Indahil, U. & Sjoberg, D. (2003) "Software Effort Estimation by Analogy and Regression Toward the Mean". Journal of Tradional Estimation Methods and Software, 68**,** 253-262.
[5] Keung, J., Jeffery, R. & Kitchenem, B. (2004) The Challenge of Introducing a New Software Cost Estimation Technology into a Small Software Organisation", Proceedings of the 2004 Australian Software Engineering Conference. Australia.
[6]P. Abrahamsson, Koskela, J., "Extreme Programming: A Survey of Empirical Data from a Controlled Case Study", Proceedings of International Symposium on Empirical Software Engineering, pp. 73-82, 2004.
[7] L. Layman, L. Williams, and L. Cunningham, "Motivations and Measurements in an Agile Case Study", Proceedings of ACM SIGSOFT Foundation in Software Engineering Workshop Quantitative Techniques for Software Agile Processes (QTE-SWAP), Newport Beach, CA, 2004.
[8] F. Maurer and S. Martel, "Extreme Programming: Rapid Development for Web-Based Applications", IEEE Internet Computing, 6(1), pp. 86-91, Jan/Feb 2002.
[9] L. Williams, W. Krebs, L. Layman, A. Antón, and P. Abrahamsson, "Toward a Framework for Evaluating Extreme Programming", Proceedings of Empirical Assessment in Software Eng. (EASE) 2004, Edinburgh, Scot., pp. 11-20, 2004.
[10] F. J. Heemstra, "Software cost estimation", Information and Software Technology, vol. 34, no.10, 1992, pp. 627-639.
[11] J. Hihn and H. Habib-Agahi, "Cost estimation of software intensive projects: a survey of current practices", International Conference on Software Engineering, 1991, pp. 276-287.
[12]G. R. Finnie, G. E. Wittig, "AI tools for software development effort estimation", Software Engineering and Education and Practice Conference, IEEE Computer Society Press, pp. 346-
353, 1996.
[13] K. McDaid, D. Greer, F. Keenan, P. Prior, P. Taylor, G. Coleman, "Managing Uncertainty in Agile Release Planning", 18th Int. Conference on Software Engineering and Knowledge Engineering (SEKE'06), pp. 138-143, 2006.