Agile Project Management

Raymond E Boehm

Software Composition Technologies

Abstract- This presentation will educate measurement professionals to the real issues surrounding agile development. It gives an overview of what agile development entails and how it is different from traditional development. The reasons for measurement are presented. Story points are described. The possibility of using function points along with or instead of story points is discussed. Use case points are explained.

Introduction

Various forms of agile software development, such as eXtreme Programming (XP) and Scrum, are starting to be used by corporate America. In many cases, these developers are refusing to cooperate with the measurement professionals already working in these corporations. They will claim that agile does not lend itself to measurement in general and to function point analysis in particular. The real issues surrounding agile development are presented here.

The presentation gives an overview of what agile development entails and how it is different than traditional development. The concentration will be on the measurement aspects of agile.

In agile development, requirements are usually captured as stories. The first measurement that agile utilizes is story points. Story points will be described. They are not functional measures like function points. The possibility of using function points along with or instead of story points will be discussed.

In agile development, story cards must be decomposed into tasks. There has been dissatisfaction in the agile community with using function points to size these tasks. This is understandable. Many tasks are implementation activities that do not lend themselves to function point analysis. A measurement technique called use case points has gotten some attention in the agile community. This technique will be explained.

Understanding Agile Development

Agile development is actually an umbrella term to describe a number of different development methodologies. These include eXtreme Programming (XP), Adaptive Software Development (ASD), Crystal, Scrum and others. They all subscribe to the values and principles that can be found at www.agilemanifesto.org.

Barry Boehm and Richard Turner have written a book describing agile practices and contrasting them to traditional planned development.[1] They describe the differences by comparing them with respect to the following four characteristics:

1. **Application** – Agile applications are usually highly changeable, both during and after development. Agile teams and projects also tend to be smaller than those for traditional applications.
2. **Management** – In agile development, the customer becomes part of the development team. Plans are less documented. Communication in general becomes more personal and less documented.
3. **Technical** – In agile development, requirements are captured in informal user stories, instead of formal requirements documents. Agile development is done in short increments, with frequent releases of software to the user community. In agile, user acceptance testing is captured in executable test cases, as opposed to voluminous test cases and plans.
4. **Personnel** – The collocation of customers is usually an agile requirement. Agile developers tend to be highly capable generalists. Traditional teams often use specialists for functions like testing. These people often are unable to assume a development role. The agile team thrives on chaos; traditional teams, on order.

The remainder of this article will focus on the planning of iterations and releases. Iterations are often two week development cycles designed to implement some user stories. Releases typically take between one month and one year. They implement a usable subset of the application being developed.

Measurement in the Agile World

The first measures that come into play are associated with user stories and releases. Which stories are necessary to have a usable application? This is a release. A release typically takes between a month and a year. How big is the release in terms of ideal programming time or story points?

Agile development projects are organized into iterations. Iterations are short periods of development where several stories are implemented. Two weeks is a typical iteration period.

Iterations have to be planned in some detail. The stories are usually decomposed into tasks. These tasks are estimated and usually assigned to a developer or two. The estimation is usually in terms of ideal programming time. However, other measures, such as use case points, have been suggested.

The calendar time for the iterations is normally fixed. If progress is slower than anticipated, then some of the lower priority stories are dropped from the iteration and moved to the next iteration. If the team implements more quickly than anticipated, then stories are added.

While agile developers are often informal in their planning, they are usually obsessive in their tracking. The amount of ideal programming time, story points or use case points is carefully tracked during all of the iterations. This is referred to the velocity of the iteration.

The planning of releases and iterations is depicted on the burn down chart in (Figure 1). It shows the number of story points that are remaining in the release after all of the iterations. The first two iterations are actual, and the rest are predicted. It is taken from a draft of Mike Cohn's upcoming book on agile estimating and planning.[2]

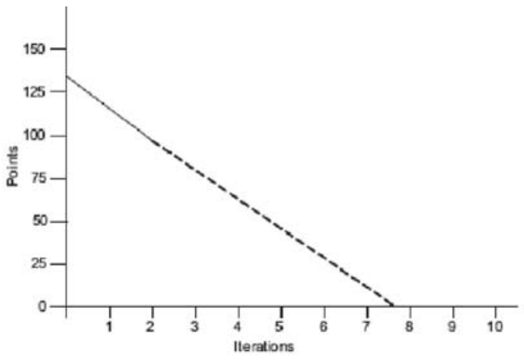

Figure 1. Burndown Chart

The initial estimates are necessary to plan the entire release, choose the appropriate size team and set customer expectations regarding the delivery of the release. The ongoing measurement resets velocity to keep the team operating at maximum efficiency. It also allows the team to communicate the impact of any changes in productivity or user requirements.

Ideal Time

According to Kent Beck, ideal programming time is the measure where you ask yourself, "How long would this take without distractions and disasters?"[3] Is it a distraction when your customer calls to discuss a clarification to the requirements? Is a corporate reorganization a disaster, or simply another day in paradise?

To many of us, this measure is reminiscent of lines of code. It seems like it should be intuitive and unambiguous. Unfortunately, as in the case of source lines of code, ideal time is neither.

In addition, ideal time may be hazardous to your health. Your management will tell you that a professional should be able to avoid distractions. Competent management will avert any disasters. Any gap that exists between ideal time and actual time can be closed with a little unpaid overtime!

Despite the problems, ideal time is used by many teams. Sometimes it will be used to estimate the time to implement a story. It is still the most common way to estimate task completion time. Like source lines of code, the measure will probably be used for a long time to come.

Story Points

In agile development, requirements are usually captured as stories. Mike Cohn wrote an entire book about writing user stories.[4] It mentions story points. He has a newer book that goes into more detail regarding the assignment of story points to user stories.[2]

According to Cohn, each story is given a story point value by the rest of the team. The points show relative expected effort to implement the story. For example, if the first story has a value of two story points and the next story is expected to take twice the amount of effort to implement, then the second story will be assigned four story points. Members of the team start by agreeing on a point value for a medium story and then assign story points to the other stories relative to that one.

Not all numbers are valid story point values, Cohn goes on to specify. Trying to decide whether a story was 10 or 11 points would imply more precision than the process is actually capable of. Instead, the possible values are 0 (for extremely small stories), 1, 2, 3, 5, 8, 13, 20, 40 and 100.

Assigning story points are a well accepted technique. A team using XP to develop an application for Credit Suisse Italy reported that using story points was one of the two techniques that allowed the team to "stay focused." [5] (An environment that minimized distractions was the other.) Fred Grossman, et al., reported that they used story points and found that it was important to make them abstract units instead of coupling them to ideal developer days.[6]

Story points are neither standard nor repeatable. The number of story points for a particular story may vary with the project and the project team. For example, Mike Cohn gives an example of the following story card in his case study: “As a player, I can restore a saved game.”[2] The imaginary team awards this 2 story points. If that same story comes up in another project, it might very well have a different number of story points. This is completely consistent with the story point technique.

Function Points

The International Function Point Users Group (IFPUG) has long taught a course, FP-211, titled “Estimating Project Size Early in the Life Cycle.” Using these techniques, it is possible to estimate the function point count from the story cards. However, the question is whether these function point counts represent implementation effort as well as the story points. Many practitioners feel they do not. The function point counts do not fully take the complexity of the transactions into account. Likewise, they do not assess the team’s capability to implement a particular story. For example, if the team had just built the capability to accept an American Express payment, accepting Visa might be much easier, but have the same number of function points.

However, the standard nature of function points may make there use unavoidable. Development in an outsourced environment may have a contractual obligation to report function point counts. Organizations with commitments to CMMI or other process improvement programs may require the collection of standard measurements. Function point counts are standard and repeatable. By definition, story points are not.

IFPUG has another course, FP-370, titled “Counting Object Oriented Applications and Projects.” It shows how to generate function point counts from use cases. There has also been work done to generate a more automated count from Unified Modeling Language.[7] A number of studies have shown that estimating small projects in this manner gives satisfactory results; but, there is concern that applying function point analysis to large use case based projects will not be.[8]

The IFPUG function point is probably the most commonly used of the function point measures. A newer, and less popular, form is the Full Function Point (FFP). Mk II is yet another definition of function points that is commonly used in the UK. At this point, agile practitioners who are aware of the difference seem to view all of them with equal distain.

Use Case Points

About 20 years after Alan Albrecht introduced the notion of function points, Gustav Karner described a measure called use case points. It, too, was intended to be an estimating technique.

Use case points were strongly influenced by the work of Ivar Jacobson and other object oriented methodologists.[9] The technique is primarily driven by the actors and use cases identified for the application. Following the steps under Actor Weight and Use Case Weight, below, will yield unadjusted weights for both. Adding these weights together will yield the number of unadjusted use case points.

The unadjusted use case points are multiplied by technical and environmental weights. This yields the total number of use case points. The significance and method of establishing these weights is described below, under Technical Complexity and Environmental Complexity,.

There is a tool that automatically calculates the use case points from descriptions of the actors and use cases.[10] The tool does not always match the value arrived at by human experts. In addition, the use cases must be written in Japanese. There are other tools that require the user to evaluate the complexity but that automate the calculations. In any case, the existence of these tools is an indication of the level of interest that exists regarding this technique.

Like function points, use case points were originally developed for estimating. Originally, a use case point was thought to take 30 hours to implement. Later studies have changed this number or made it a function of additional cost drivers.

Benta Anda has conducted several studies comparing the accuracy of use case point based estimates with actual results and with estimates generated by experts.[11] The use case point based estimates were fairly close to the actual development effort. They were usually closer than the estimates presented by the experts.

Use case points can also be used like story points in an agile environment. In practice, the unadjusted use case weight is often used to measure work. The technical and environmental complexities can be ignored because they end up reflected in the velocity of the iteration. Ignoring the actor weight is a practical matter. When would credit for the actor weight be awarded, when it was first encountered, pro rated over the application implementation or when the last use case interacting with the actor is implemented? The first and last possibility would overstate impact for that particular iteration. Pro rating would probably be more work than it is worth.

Actor Weight

Use the following criteria to assign a complexity and weight to each of the actors:

- A simple actor might be another application that accesses this application through an API. Its weight is 5.
- An average actor might be a user accessing the application through a text-based user interface. Its weight is 10.
- A complex actor might access the application through a graphical user interface. Its weight is 15.

These weights are summed to arrive at the unadjusted actor weight.

Use Case Weight

Evaluating the use cases requires a fair amount of knowledge about use cases. A course on use cases is beyond the scope of this article. However, the calculation of use case weight will be described for those who are interested.

Each use case consists of one or more transactions. Each step in the main success scenario is a transaction. Some extensions are also transactions; those that are a continuation of another transaction are not counted.

Use (Table 1) to assign weights to each use case based on their complexity. The complexity is established by the number of transactions. Sum the weights to arrive at the unadjusted use case weight.

Table 1. Use Case Weights

Complexity	Number of transactions	Weight
Simple	3 or less	1
Average	4 to 7	2
Complex	7 or more	3

Technical Complexity

The way that use cases are implemented have an impact on the cost, and therefore on the use case points. For example, an application that is designed to be portable between several different platforms will probably take longer to develop than one that only works on one platform. This would be the case even though the actors and use cases were exactly the same.

Take each of the attributes in (Table 2) and assign a value between 0 (for no impact) and 5 (for very high impact). Multiply that value by the weight and sum up the values. Multiply the sum by .01 and add .6 to get the technical complexity factor.

Table 2. Technical Complexity

Factor	Desription	Weight
T1	Distributed system	2
T2	Performance objectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Resuable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent use	1
T11	Security	1
T12	Access for third parties	1
T13	Training needs	1

Environmental Complexity

The team that performs the implementation obviously has an impact on the cost. For example, an application development team that is familiar with the development process would be able to perform that implementation more quickly. The environmental complexity factor accounts for this.

Take each of the attributes in (Table 3) and assign a value between 0 (not the case) and 5 (very much the case). Multiply that value by the weight and sum up the values. Multiply the sum by -.03 and add 1.4 to get the environmental complexity factor.

Table 3. Environmental Complexity

Factor	Desription	Weight
E1	Familiar with the development process	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

Conclusion

While agile developers pride themselves on informal project planning, they nonetheless make use of a number of measures to estimate and plan iterations and releases. The main measures include:

- Story Points – are used in one form or another by virtually all agile practitioners.
- Ideal Time – is sometimes used to estimate user stories and usually used to estimate the tasks in iterations.
- Function Points – are not in favor by the agile community. However, many organizations

have process improvement or outsourcer governance needs that make the use of function points necessary.

- Use Case Points – are the subject of a fair amount of interest in the agile community, especially those using use cases as part of their development process.

As agile development continues to move into the mainstream of IT, there may be some changes. Agile projects may get larger. Organizations may require that they be measured in traditional ways. In any case, more research is necessary to find the proper mix of measures.

References

[1] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley, 2004.

[2] M. Cohn, *Agile Estimating and Planning*: Addison-Wesley, 2005.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley, 2000.

[4] M. Cohn, *User Stories Applied: For Agile Software Development*. Boston: Addison-Wesley, 2004.

[5] P. Bossi, "eXtreme Programming applied: a case in the private banking domain," presented at OOP2003, Munich, 2003.

[6] F. Grossman, J. Bergin, D. Leip, S. Merritt, and O. Gotel, "One XP experience: introducing agile (XP) software development into a culture that is willing but not ready," presented at The Conference of the Centre for Advanced Studies on Collaborative Research, Markham, Ontario, Canada, 2004.

[7] G. Cantone and D. Pace, "Applying Function Point to Unified Modeling Language: Conversion Model and Pilot Study," presented at 10th International Symposium on Software Metrics, Chicago, Illinois, 2004.

[8] K. Vinsen, D. Jamieson, and G. Callender, "Use Case Estimation - The Devil is in the Detail," presented at 12th IEEE International Requirements Engineering Conference, Kyoto, Japan, 2004.

[9] I. Jocobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, England: Addison-Wesley, 1992.

[10] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, and Y. Maegawa, "Estimating Effort by Use Case Points: Method, Tool and Case Study," presented at 10th International Symbosium on Software Metrics, Chicago, Illinois, 2004.

[11] B. Anda, "Empirical Studies of Construction and Application of Use Case Models," in *Department of Informatics*: University of Oslo, 2003.

About the Author

Raymond Boehm is Software Composition Technologies's principal consultant. He is an IFPUG CFPS, a QAI CSQA and a member of the ACM and the IEEE. Contact him at rayboehm@softcomptech.com.