

Estimation as Hypothesis

By

Alan Shalloway



Copyright ©, 2001, Net Objectives

Abstract

Estimates in software development have evolved into deadlines. This is not only an ineffective way to develop software, but it also puts tremendous pressure on developers, resulting in an unbalanced lifestyle and making their companies less attractive to them. Deadlines are based on several fallacies, the chief amongst them are that working more hours results in more production and that staff needs pressure to motivate them.

This article presents an alternative to using estimates as deadlines. The authors propose that an estimate of a project's completion should be considered an hypotheses of when a project will be completed, not a deadline for its completion. The factors needed to properly test estimates are considered and discussed in the framework of an agile methodology (such as XP). Within this framework the theorized completion date depends upon the rate of production and the extent of scope as determined by the business owner.

The belief system underlying deadlines is contrasted with that of using an estimate as an hypothesis.

Note: This article assumes some familiarity with XP and Agile Methods. If this is not true, please read the appendix before proceeding.

Voltaire

Doubt is uncomfortable,
Certainty is ridiculous.

Estimation

Myths lost and myths gained

Although XP and Agile Methods have destroyed many given truths in the software arena, they also seem to have created one of their own. Many people are under the misunderstanding that many XPers would say that one can't say when a project will be completed. (I was one of these people until Ron Jeffries corrected me). Factors such as changing scope, lack of knowledge of how long a task will take seems to make it impossible, or at least undesirable to say "the project will done by so and so date".

Given the practice of an estimate becoming a deadline, we'd probably agree. However, it also results in making it difficult to improve on one's estimation abilities. We feel estimation can become something we are good at, not merely a hard to grasp art. It just needs to be viewed from a different angle.

Estimates in an Agile process

There is no reason one cannot make estimations in an Agile process, and many people do. However, two main things should be noted about this estimation. First, in following an iterative process, deciding on what to do is based on what the customer is telling the developers at the beginning of each interval. This means estimation is based on the rate of production and only indirectly indicates the date of completion. Second, the estimate is always suspect. As the team learns more about their project, scope will change as will the team (their abilities). Agile processes are founded on the notion that truth is paramount. Therefore, as our "truths" change, so must our estimates. Hence, estimates become hypotheses about the current, truthful, state of the project.

Does this mean estimates don't work or can't be relied on? No, it just means we acknowledge the inherent uncertainty of our estimating process and take steps to improve it.

Estimates as Deadlines

Deadlines

Before pursuing how estimates should work, let's investigate a little more about how they are typically used.

Merriam Webster's dictionary defines *estimate* as:

"to determine roughly the size, extent, or nature of c : to produce a statement of the approximate cost of"

It also defines deadline as:

“**1** : a line drawn within or around a prison that a prisoner passes at the risk of being shot

2 a : a date or time before which something must be done **b** : the time after which copy is not accepted for a particular issue of a publication”

While the second definition of a date or time is the more commonly used one, the first one of a line drawn around a prison seems more appropriate. How many developers have had the experience of their project being a prison they can't get out of, with the threat of death (of the company or their job, anyway) if they cross the deadline without the project in hand? What started out as a best guess now becomes something you will be “killed” for if you don't meet.

No wonder people have such a dread of deadlines. Is it any wonder that many agile practitioners say we won't have deadlines? Is there a better way to manage our estimates? We think so.

Avoid deadlines, not estimates

Deadlines can be counter-productive. However, we don't need to throw out the baby with the bathwater. Yes, deadlines are often a bad thing in an agile process, but an estimate of when a project will be completed can still be a very useful thing.

Let's consider one of the fundamental values of any agile process: truth – telling it, looking for it, honoring it. Let's contrast this with the motivations behind converting an estimate into a deadline.

Deadlines are used for motivation

Deadlines are typically used to force people to work harder to meet it. It's often an arbitrary date some higher up gave as when a project is to be completed. Often the thought is that if pressure is put on developers, they will work harder (and more hours) and therefore get the software produced more quickly than without the deadline.

Deadlines assume the wrong things

However, using deadlines this way assumes that:

1. The staff needs to be motivated.
2. That working more hours translates into more production.
3. The short term benefit of getting the software done quicker (which we don't think happens much anyway) outweighs the longer term cost of staff burnout.

We believe all of these assumptions are incorrect. First of all, a good team should consist of motivated people who need to be led. It does not consist of unmotivated people who need to be managed. Perhaps management may need to let their staff know the mission and values of their company to help motivate them, but if that doesn't work, management had better start looking for new people.

In the industrial age, it was possible to use people's bodies without their souls. This wasn't really a good idea even then, but it could work. In the information age, it doesn't stand a chance. People are paid to think, but they can't be externally pressured into it, at least not to do it well. If a project has a team of developers who really do need to be motivated, that project is in trouble.

Working harder is not the same as getting more done

The other problem with deadlines is the unstated assumption that forcing people to work harder to meet them is the best way to get things done. Working harder may get more done in the short term. However, more often than not, much of the productivity gained is lost if the extra pressure prevents the team from seeing a better way to do the job. This extra work created by not taking a better approach is usually more than the extra number of hours put in. The extra work actually slows the project down. No wonder a project dictated on overtime often seems hopeless for the developers. The longer it goes on this way the worse the cycle gets – working more inefficiently.

Work smarter not harder is a trap

Of course, as the pressure rises, developers often hear the argument – “work smarter, not harder”. But what does “working smarter” mean? This smacks of “buy low, sell high” in the stock market. In other words, of course I'd like to work smarter, not harder, *but how do I do this?* When we work, are we not always working as smart as we can? Clearly we can't raise our IQ just because our boss would like us to.

We must work on the right things

Working smarter means to work on the right things. This implies don't work on the wrong things. Working on the wrong things is wasteful, so working smarter not harder could be said to mean – to be more productive, work on the right things.

Do deadlines help us work smarter?

Will having a deadline that forces overtime help or hinder developers working on the right things? It is our opinion that they hinder. When people are forced to work extra hours, several things happen. For the first month or so, it can work out pretty well as the staff isn't burnt out yet. However, this doesn't last. As the project changes, people won't be taking the appropriate time to reflect on what the changes mean or imply. The result is the project can churn. As Yogi Berra once said – “we're lost but we're making great time.”

Extra hours numbs the problem

If management thinks they can get the extra production they need by simply having developers work more (with no or only minimal extra pay) they have little motivation to see if there is a better way for the developers to be working. An inefficient process will stay inefficient until there is a compelling reason to change it.

Overwork is deadly

As people work overtime for months on end, their personal lives become unbalanced. The motivation of extra money sometimes keeps them going (at

least before the dot bomb). However, what is this setting up for the company's future? The company is getting their best people to work extra hard to get a project done with the motivations of a rich reward at the end. The people are often thinking they can then leave this job and go to a better one (or retire) once they have all that money in the bank. This is short term thinking.

Not real hours anyway

Also, these extra hours are often vacuous. What often happens is that developers shift their personal lives into the office. Since they are working 16 hour days, they start running their errands in the middle of the day – since they'll be working at night. Companies start having social events during work hours to perk up the team.

An ironic situation

The irony is that all of this pressure is selling out the company's future for the present. Yet, there is little evidence that this pressure will get the job done sooner. We suspect this myth is held in place because there have been teams that have built projects in very short periods of time that worked seemingly endless hours. However, in all projects of this type that we have known of or been involved in, the hours were put in because people were excited about what they were doing. They were juiced on the project.

Estimation as Hypothesis

Hypothesis

Let's now consider an estimate as an hypothesis. Merriam Webster's dictionary defines hypothesis as:

“a tentative assumption made in order to draw out and test its logical or empirical consequences”

No more failure

Using an estimate as an hypothesis has many advantages. First of all, it removes the stigma of updating an estimate as being a failure. In other words, if I discover that I've misestimated, I am not wrong, I have merely refined my estimate – I have learned something that is more accurate. Sure, if the estimate is now further out, I may not be pleased with that, but I can at least take satisfaction that I am now closer to the truth.

We want to avoid this classic situation

By re-evaluating our estimated project completion date, we can avoid the situation the lead developer found herself in in Jim McCarthy's Dynamics of Software Development.

You approach a milestone date and ask the lead developer, “How's it going? Are we going to hit the milestone?”

“Well,” she answers, “I've been doing really well for the last six weeks, but today, you know, things were really bad and I slipped six

months.”

We can track our productivity and therefore improve it

Using estimation as an hypothesis is based on the notion that dealing consistently with truth is always better than ignoring the truth. The truths we must deal with are:

- how much stuff do we have to do?
- how fast can we do it?
- how much stuff is being added?
- how much stuff is being removed?

(and yes, stuff *is* a technical term).

If the staff of a project can answer these four questions, they can predict pretty well how long their project will take to complete. Getting good at all four of these questions may not be easy, but not paying attention to them will definitely not improve your abilities in this area.

We restate these truths into more measurable statements:

1. What is the time required to implement the stories the business owner is currently saying must be done to complete the project?
2. How well does the team implement stories they have estimated? (i.e., how accurately are their estimates of how long a story will take just prior to doing it?)
3. How many productive hours can the team bring to bear on the project?
4. How many new stories are added on a weekly basis?
5. How many existing stories are removed on a weekly basis?

Ironically, although these are often not measured on agile projects, there are many ways to measure these in the agile literature.

Benefits

By viewing our estimate as an hypothesis, we are willing to relook at it continuously. This can help us identify (and more importantly, to do something about) our risks. We are more willing to be responsive (agile) because there is less of a stigma to doing so.

This simple metric of where I spend my time and how long it took to complete something compared my prediction for the task enables me to become a better estimator.

The team psyche will improve. All team members (developers, project managers, business owners) are all focused on getting at the truth – when will the project be completed. If the answer is not acceptable, we can brainstorm about better approaches or kill it before it dies an agonizing death on its own.

We'll also avoid the living in hope syndrome:

“Oh sure, we recognize it's taken us 2 months to do what we thought we'd do in the first month of the project, but we'll make up for that in the rest of the project.”

Why would we expect things to go faster after our initial test has shown us it is going slower? It's just another triumph of hope over experience.

Testing our hypothesis

There are many ways to test our hypothesis. At the start of projects, we like to do a set of small tasks that is representative of the full set (or at least a significant portion of them). By doing this, we can see if our estimates of the known tasks are at least reasonable. We are looking for order of magnitude errors at this point.

Summary

I have discussed how most software development projects convert an initial estimate (or the owner's wishes) into a deadline. Developers are then forced to live under the shadows of this and are encouraged to use brute force (overtime) to achieve the deadline. Burnout and delayed projects are often the result.

An alternative is to view estimation as a process of making hypotheses about how long a project will take. This allows us to test it (the estimate) and focus on the truth of the project. It also has the benefit of removing the stigma and dread of failure, making for a more enjoyable, energetic and productive team.

Next Steps

Bibliography

eXtreme Programming Explained. Kent Beck. Still the best book on XP. Even if you aren't cut out to be an eXtreme developer, this is a brilliant book.

Peopleware. DeMarco and Lister. A timeless book that applies to teams as much now as it did in the 80s.

www.netobjectives.com We have many resources listed at our web-site.

Community of Practice

At Net Objectives, we have found that developers learning together learn faster. We host a community of practice to support this, organized around online discussion groups. We facilitate discussions as needed and use them to determine additional information to post on the site. Our communities give learners a chance to learn together.

Visit **www.netobjectivesgroups.com** for more information, or to participate.

Courses

Net Objectives offers the following courses:

- Use-Case-Based Requirements Analysis
- Agile Development Best Practices
- Design Patterns Explained: A New Perspective on Object-Oriented Design
- Refactoring, Unit Testing, and Test-Driven Development
- ...and many more

You can get more information at: <http://www.netobjectives.com>

We do public and on-site course. Contact Alan at alshall@netobjectives.com for more information.

Net Objectives e-zine

You can also sign up for Net Objectives' e-zine which distributes articles like this by sending an e-mail to **info@netobjectives.com** with "subscribe" in the heading. It should contain your name, company name, e-mail address, your role and the city and state where you live (or country if you are not in the US).

Appendix – Concepts Assumed

XP - Extreme Programming

This is a very lightweight process that is based on the notion that things will change so much you can't anticipate it, you must quickly respond to it. XP follows several practices, the most significant ones being:

- paired programming
- up-front testing
- only build things the customer requests
- do the simplest thing that could possibly work
- iterative, incremental development
- no overtime
- honest communications
- strive for feedback

Agile Development

A broader approach than XP (i.e., XP is a kind of agile development). Agile developers focus on being able to adapt to changes in the project (i.e., requirements) and eliminating waste.

Story

In XP, the term used for the functional units to be implemented. Not a Use case as they are smaller. They are a brief (in story form) description of what a customer wants to do.