

## *On Time, On Budget, All the Requested Functionality*

### *- It's Not Enough*

By Robbie Mac Iver

#### *The Traditional Success Criteria*



Success of a project is often described as being on time, on budget, and performing all the expected work. Certainly you want the road construction project in your neighborhood to be completed on time, on budget and you want the entire road constructed (one and  $\frac{3}{4}$  lanes just won't do). Most engineering efforts lend themselves to these success criteria and meeting them is a reasonable, if sometimes difficult, expectation. If you are looking for your software development efforts to be on time, on budget, and to deliver all the requested functionality however, statistics show you stand a very good chance of being disappointed.<sup>1</sup> But is that a failure? I don't think so.

*While on time, on budget, all requested functionality is a great goal; this is not a good success measurement for software development projects.*

**Why?** *You can be on time, you can be on budget, you can deliver all the requested functionality and you can still fail.*

**How?** *You can deliver the wrong thing. You deliver some very pretty software that likely works as intended, but no one actually uses it because it does nothing that anyone really needs to do. It does not help people do their jobs; it does not solve the business problem.*

Is this still a success? I don't think so. Furthermore, sadly, it was likely quite evident somewhere along the way that the business problem was not being addressed and it was either not recognized, or not addressed. We were too fixated with being on time, on budget and delivering all the requested functionality. Software development is much more dynamic than the other more traditional engineering efforts it is often compared to, and it needs a more dynamic measurement of success.

## A Better Success Measurement



Software development is not about technology; it is about business. The ultimate goal is to make the business bigger, better, or more efficient. The goal is to add value to the business. Perhaps the software is a new product to sell, perhaps the software supports your newest service offering, perhaps the software allows you to process a customer order in 2 minutes, rather than 5. Whatever it is, we must recognize, understand and evaluate the business value. If there isn't one, we should not be developing software.

Business value must also be weighed in the context of time. Delaying that new service offering because the underlying software is not ready negates the value of the software. It may also give your competition a head start; perhaps one from which you won't recover. Business needs often dictate a firm due date.

The first task of a software development manager is to understand the significance of the date. Knowing what the recipient of the software intends to do with it upon delivery is key. Is the software targeted for a demo at a trade show? Does a key customer want the software to start a pilot program? Will the software be deployed to 100k seats? What has to be delivered in each of these cases could be very different and therefore the success measurement of each is different as well. Determine what that really is and build a consensus around it early. Also be prepared for it to change as the evolving business needs do.

So what is a good software development success measurement? I would state it as:

*“Providing business value within a predictable schedule and a predictable budget”.*

This provides a dynamic measurement that can evolve as the business needs and software requirements do. Using a dynamic “value” test provides alternatives for finding success beyond “on-time, on-budget, all the requested functionality”.

## The Value Test

Software development is hard, it is risky, and it is expensive; it is not for the feint of heart. Software development organizations need to be more selective in what they choose to develop. Recognizing the need for software does not mean you should actually build it.



My company makes widgets, and a very good widget; so good that I need a better financial system to keep track of it. Should I build one? The short answer is no. My expertise is widgets, not financial systems. Sure I could figure how to build an adequate system, but why? Would my business be better off because I built a great little financial system? Is this the best way to add value to the business? Not likely. Would my business suffer because I focused all my smart people on a financial system rather than on building better widgets? Likely so. Buy this type of software from someone

who does have that expertise, but before you do, scout out the open source community for solutions first.

Now suppose I discovered a way to transform my widget into a smart-widget by adding software to it, and my new smart-widget could then significantly out perform any other widget on the market. Should I build that software? Absolutely, it makes the business better; it adds value.

*Software that involves your own intellectual property is software you should build.*

*Software that involves significant domain knowledge in your business domain is also software you should build.*

In the first case, you don't want to give away your IP. In the second case, you don't want to have to educate someone else about your business domain to enable him or her to build your software. If you are thinking about building software that does not fall into these two categories, think again. It is not a good use of your resources, and it adds little or no value to your business.

### **Constant Feedback**

Now that you are building software that provides significant value to your business, you must come to terms with some myths about software development requirements.

True or False?

- You can discover all the requirements up front.
- You can stabilize requirements changes early in the cycle.
- Once a set of requirements has been determined, you can with great precision forecast the development effort needed to implement them.

If you answered true to any of these your software development effort is going to be troubled. If you answered true to all of them, it will be doomed.

A successful business is in constant change. It assesses the market and repositions itself to its own best advantage. Just as a caterpillar does not become a butterfly by maintaining the status quo, a business is in constant metamorphosis preparing itself to compete in tomorrow's business world. There are two things you can be sure of about software development requirements in this type of environment:

- You don't know all the requirements.
- The requirements you do know will change.

Software requirements will evolve as the business does. If they don't, you run a significant risk of delivering the wrong thing.



Constant feedback is needed to be certain that a development effort is still providing value. Using this feedback the development effort can be realigned with the business, expectations can be managed, and new requirements can be assessed. And yes, those expectations of what needs to be delivered by what date should change as well.

Software development processes often breakdown at this point. Traditional “analyze, design, build, deploy” life cycles (often called water-fall processes) are predicated on the notion that requirements can be discovered upfront and won't change. Feedback is received too late in the cycle to allow sufficient reaction to changes when they do occur. Working software cannot be vetted by the stakeholders until late in the build and deploy steps, and by then the business could have evolved completely away from the initial requirements set. These life cycles provide little or no opportunity to accommodate new requirements without considerable rework. There is a better way.

*Build software in small self-contained pieces; build new pieces that extend the functionality of the previous pieces.*

Before building a new piece, reassess what is really important to the business. Proponents of agile development processes will recognize this as being iterative and incremental. The broad scope of a development effort is defined at a high-level and divided up into chunks. Each chunk is addressed in a time-boxed period (called an iteration) with a finite start and end date. Iterations can be a varying length depending on the work planned but are typically between 2 and 6 weeks. At the end of the iteration an assessment is done to determine if the goals of the iteration were met and that feedback is used to plan the next iteration to build the next increment the software product.<sup>2</sup>

There are three rules for iterations:

1<sup>st</sup>

The end date does not change. If the work planned is not completed then adjustments are made to subsequent iterations or an additional iteration is added.

2<sup>nd</sup>

Requirements for the current iteration do not change in the course of the iteration. Requirements for the next iteration however, are fair game. Stakeholder feedback from one iteration is used to plan the most important (valuable) work to address in the next iteration. This frequently will not be the work originally planned, which is the exact point. Iterations allow you to address new requirements in context of the existing requirements and prioritize how to address them.

3<sup>rd</sup>

Each iteration (with the exception of the first one or two) produces production quality, executable code implementing some portion of the overall requirements.

Gathering and reacting to feedback from each iteration is critical; don't wait until the end of the development cycle to start. Develop that initial "high value" use case. Put it in front of the key users and stakeholders that benefit from it. Does it solve their problem? Does it fit the way they work? Does meet their expectations? This will generate new requirements, and we want it to. That is how we ensure we are adding value. When new requirements are identified, use the value test to weigh their relative priority to the requirements already planned and let that guide what work is done next.

Conduct frequent checkpoints with management and the key stakeholders. Examine the value that has been created, and the potential value that is represented by the remaining requirements. Is the business still being well served? Predictable, visible progress becomes evident and the discussion centers more on adding value than meeting an arbitrary date.

Sometimes success is recognizing the need to stop. You may look at the value test again, and realize, something has changed and the effort no longer provides the value it did. Stopping is not a failure; failure would be continuing with the effort unaware the business circumstances have changed. Adopt a process with frequent gates that require justification for continuing. It should be easier to stop a project than it is to continue it.

## Succeed by Changing How You Work



Achieving different results for your software development efforts requires taking different actions. Succeed by directing those actions toward three things:

- Business Value
- Constant Feedback
- Frequent Checkpoints

Software development is not about adding feature sets by a fixed date. It is about adding value to the business. First and foremost, understand what that value is. If your development effort doesn't start out with more requirements than will ever be developed, it will quickly reach that point. That is the nature of evolving requirements. A value based development process will embrace that evolution because it is exactly that evolution that will ensure value is being created. **Recognize** that requirements are changing, and manage those changes. **Understand** that what was important yesterday, may not be important today. **Adopt** a process that encompasses the broad picture of all the requirements, but focuses on only those that bring the most value today. **Succeed** by delivering value to the business within a predictable schedule and a predictable budget.

*softwareDecisions, is a Texas Limited Liability Partnership providing information technology consulting services to the greater Houston area. softwareDecisions is dedicated to making software development and integration efforts succeed by addressing the right requirements, focusing on the right priorities, and building the right relationships.*

*Robbie Mac Iver, PMP  
Owner/Principal Consultant  
President, Mac Iver Management, Inc., General Partner*

<sup>1</sup> The CHAOS research by The Standish Group International currently estimates that only one-third of software development project succeed by this measurement.

<sup>2</sup> For more information on iterative development visit see [Rational Unified Process](#) at the [IBM](#) web site.