Developing a Software Product

Software Product Development for the Non Technical

By

Joe Walling

http://wallingis.com

Introduction

Lately, there is frequently news about another software company being acquired or going through an IPO with a valuation of over a billion dollars. Many seeing this are wondering how they can have the next big software product. This book is aimed at people that have little knowledge about software development or have tried having a software product developed and did not have a good experience.

This book is not designed to teach someone how to write code or be the software developer. Instead, it is aimed at teaching someone how to come up with an idea and then screen and hire the correct people to help you develop the product. It will cover the full lifecycle of software development in non–technical terms so the lay person can understand the concepts.

Through this book, I hope to convey what I have learned in the 30 years I have been doing software development and running my own software business. In addition to writing software myself, I have also had to hire and work with over a hundred developers over my career. As you might guess, I did not always hire the right person or write profitable products. On the other hand, I have written some very profitable software. My hope is that this book will help you learn from my mistakes and my successes.

If you are currently a programmer that is interested in creating software products, you may be interested in the chapters on coming up with an idea and how to

monetize your product, but don’t expect to find any tips about writing code in these chapters.

TABLE OF CONTENTS

Legal Notes

This book is not meant to offer any legal or financial advice and makes no claims about how profitable or successful you will be. While this book is based on processes that have worked for me, the specifics of your situation may be different so you may get different results. Be sure to check with your legal or financial professional relating to your specific situation.

Overview

Despite the fact that you may see a lot of people making money while developing software, the process is one with many ways that things can fail. It is a difficult process unless you understand how the pieces fit together and what processes to put into place to maximize your odds of success. Even skilled software developers have a high rate of failure. Conservative estimates for skilled software development teams, has the failure rate at over 50%.

At this point you may be asking “if skilled teams have such a high rate of failure, how can I stand a chance of succeeding?” Rest assured that it is possible if you follow some straightforward, repeatable processes. While these processes aren’t easy, they are something almost anyone can follow. When skilled teams fail, it is usually because they missed or failed at least one step in the process outlined here. The keys to the process are:

- Finding an appropriate product to develop
- Developing a specification document that spells out how the product will meet the needs of your target market
- Communications and project management
- Product development
- Being able to monetize the product
- Support and improve the product if you can monetize it

If you are new to software development, the biggest piece of advice I can give you is to “think simple”. Get your feet wet with a small project that can be quickly developed and taken to market. Your first effort should not be to develop an accounting system. You will make mistakes during this learning process, so isn’t it better for these mistakes to be inexpensive mistakes?

Following are several reasons why you might want to consider developing a software product

- Has a high perceived value when compared to a typical info product
- No inventory is required
- Can be relatively cheap to build when compared to building physical products.
- Not everyone can do it. Since the idea of software development scares most people, there is less competition.
- You can sell it over and over after it is developed.
- You can outsource the development process

In this document, we are going to go through all of the steps involved in the software development lifecycle. Below is a short version of what we will be covering:

1. Choosing a product to create
2. Writing the specifications
3. Finding the right software developer
4. Writing the software
5. Testing the product
6. Deploying the product

7. Supporting the product
8. Determining what needs to be improved in the product and determining if it makes sense to improve.

When you have been through all eight steps, you should have an idea of whether or not the product will be a success. At this point, you should look at what you have spent and what you have made. Then do a forecast of future sales.

If when you look at the numbers you can see that the product is going to make you money quickly, it may make sense to ramp up development. If it is not obvious whether you have hit the mark and have a product that is going to sell well, you should hold off a little while and concentrate on sales. You don't want to put a lot more money into software development for a product that is not going to break even.

It is best to figure out early that a product is not right for the market or that you just don't know how to sell into a specific market.

Certain topics in this book may get a little too technical for some people. While I have tried to simplify it so that anyone can understand it, you may not see a need for understanding some aspects. In that case, it is OK for you to skip those portions. However, you need to be sure that you are working with someone who can handle those technical issues as they are important to the software development process.

When we get into some of the tool and technology chapters, you may find that some aspects are overkill for a really simple project. The hope is that after you have a few successful small projects that you will do some larger projects and then have a need for those chapters.

The most important aspect of developing a software product, or any product for that matter, is that you take action. If that action results in a product that a lot of people need and are willing and able to pay for, then it will be a success.

WHAT TO DEVELOP

In this chapter, we are going to come up with an idea of what to develop. We will go through a 3 step process to come up with a product.

1. Brainstorm
2. Evaluate each item
3. Pick an idea

The first step is brainstorming. And this stage, don't try to go into detail or evaluate the merits or feasibility of the idea. Just write down what comes to mind and move on to the next idea. To some people, this comes easily, while for others this is more difficult.

To make this easier, think in terms of problems you are having or have had. These ideas can come from problems you are seeing at work, problems common to many of your clients, or from problems or opportunities in your experience as a consumer. Also think about opportunities related to your hobbies.

The best opportunities are in areas you are familiar with. Often, people overlook all of their job or industry specific knowledge and think everybody knows what they know. That is not true. The odds are good that you have figured out things that others in your line of business have not. How can you write software to help them?

Think back to when you started the job and had a hard time figuring out how to best do your job. Is there some software that you could create that would make

your job much easier or would have made it easier when you first started? Do you have a special process that you follow that is innovative or that would be useful to others in your industry.

While you can write software for an industry you don't have experience with, your chances for success are much slimmer. Not only do you have to get the software developed, but you also have to learn a new industry with new processes. Often, getting this industry experience could take years.

If you are still having difficulty coming up with ideas, here are some other things you can do:

1. Ask others what they want to make their jobs or lives jobs easier
2. Think about the other software programs you use. Is there any functionality missing that you and others like you want to see, but the vendor won't deliver? If so, you might create a product to address those issues. These focused products generally are small and quick to develop. You will need to make sure that the product you are creating this for has a large user base.
3. Go to a book store and look at the major categories and the popular topics in the various areas. Is there any software that might solve problems for those popular topics?

You don't want to build a product in search of a problem. You want a clearly articulated problem that your application can solve. You want an application

that people immediately understand how it helps them solve the problem. You don't want one that you spend all your time trying to educate prospects how this product could be useful to them.

Once you have finished brainstorming, it is time to evaluate the items on the list. Start off removing any item from the list that is absurd or would make no sense for you to develop. Then look at the list of questions below and make several passes through the list eliminating items based on answers to the below. When you get your list down to the top 3 or 4 possible products, it is time to ask for some outside opinion. You don't want to settle on one product when you might not have thought it through completely. Getting some other opinions is helpful. Ultimately, pick the product that you think will be the best, using the advice you have received.

When evaluating the list, ask yourself the following questions.

1. Does the product meet a need?
2. Does the product require a specialized business knowledge? If so, do you have that knowledge?
3. Are there competitors?
4. What is the size of the potential user base?
5. How much would a person be willing to pay for the product?
6. Can the product be monetized?
7. How long will it take to build the product?
8. How much will it cost to build the product?

9. Do you need a salesperson to sell the product or can it be sold purely online?
10. Do you have access to a market that would buy this product?
11. How much support will be needed? Will you be able to support the product or will you need to pay for help?
12. Are there any special legal considerations?

The product you develop should meet some need for your target audience. It could be that it solves a problem that many people in that niche have. The more painful the problem, the more likely they are to buy your product.

Note that people don't always buy things that they need. In fact, they are more likely to buy something they want than something they need. For example, people that smoke know that it is bad for them and that they don't need the cigarettes, yet they continue to do so because they want to. The point of this is that if you can find a product to create that they both need and want, you will have much more luck selling the software.

If the product you are creating requires some special business knowledge, do you have that knowledge? Or will you have to learn about this niche first? It is best if it is a topic that you already have an understanding of so that you won't spend a lot of time having to learn the business. It also means you are more likely to have a feel for what is important to that business

owner and less likely to miss the mark on functionality or importance of some functionality.

Are there competitors in this niche? If not, you might want to steer clear. There is probably a reason that there is no competition. For example, the group you are targeting may not be willing to spend money. They may have very low margins or they may not see what you are offering as a problem they have.

If there are competitors, it means that there might be a market need. To know if there is definitely a need, you need to do more research as it may turn out that your competitors aren't making any money. If that is the case, then you don't want to compete with them for no money. If your market research shows that they are able to monetize their software investment, try to figure out how large the market is. How many potential sales are there? How much are they selling the product for? How much money would this make you? The larger this number the better the opportunity for you.

When looking at competitors, you want them to be small to medium sized companies. You don't want to try to compete with a company like Microsoft in an area that is their strength because they will squash you like a bug. It is advantageous if the competing products are fairly weak and would be easy to improve upon.

If you have determined that there is a market, then you have to think about your ability to reach that

market. Do you personally have a way to sell into this market? Will you need to hire a salesperson to do this? If so, you need to take this into account when you run the numbers.

Another thing to consider is the sales cycle. How long is sales cycle? Are there only certain times of year that people will be buying the product? For example, if you are selling to schools and municipalities, they have a very distinct time that they buy. If your product is getting released a couple weeks after this time, you will have to wait another year for the next opportunity. Can you afford to wait that year with no sales?

While you might not be able to estimate the cost of software development at this point, you should be able to estimate the relative development cost based on complication of the product. If this is one of your first projects, make sure you are picking simple projects. For now rule out the more complex ones. You can save the list and later may be able to do the more complex items.

Make sure the numbers work. The closer you can get to estimating all of the costs and the potential revenue, the better you will be able to estimate the success of the various projects you are considering.

Start simple? Simple means lower development cost, lower support cost, and much lower risk.

Defining Software Success

As was mentioned in an earlier chapter, a large percentage of software development projects fail. When you see these statistics, you have to understand the meaning of failure as it is different for every project and is based on who is defining the success or failure. For example, to a business, the definition might be

The project does not meet the needs of the business unit at the desired price and in the specified time frame.

How many projects do you think fail that definition? Very few projects succeed on all of those fronts – functionality, time, and money. You could in fact fail by that definition but still have many happy users. They may love the program and not know that the program cost twice as much as was budgeted. So the majority of the people may not even know that it failed. The point here is that even with happy users, you may not get more work from the company because the person paying your bill might not like the cost overruns.

All this is to say that failure is hard to define and you have to look at it many ways to see if your project is a success. As well as these external things such as happy users, getting the project done on time and on budget as promised, you must also be a winner in the deal. If you were not compensated sufficiently, then would you call it a success? When dealing with your

clients, employees, and vendors, you should always be looking for win-win scenarios. It is never good when one of the parties is a loser, or feels like they have lost.

Creating software involves trade-offs. No software is perfect. The more features you add, the longer it will be before you can release the product. Many companies spend so long in developing the product that they run out of money before they can release the product. Besides, if you release a minimal set of features and get it into the hands of your users, they will be better able to give you feedback and direct you in what features are most important for the next release.

It boils down to whether the user would prefer to have good, working software today or perfect software at some point in the future.

Know when to stop. Adding too many features can be as bad as adding too few. These added features that may rarely be used can make it more difficult for the user to get to the features used frequently and may cause confusion. The can also be a cause of bugs. Every added line of code is another opportunity to create a bug.

Listen to your client. Ask probing questions and spend at least 70% of the time truly listening to the answers they give you. They will tell you what they want.

Writing Specifications

It is important to give deep thought as to what you want to accomplish with the software. If you tell a builder that you want a building and give no other amplifying information, what do you think you would get? Would you get a house, a garage, an office building, or something else? About all you can be sure you will get is a floor, roof, walls, and doors. Now suppose you tell them you want a house. With this increased specificity, in the US, you are likely to get a building with a kitchen, bathroom(s), living room, and bedroom(s). This is still far from enough information to be sure you and the builder are thinking along the same lines. You could be thinking the White House and they could be thinking a starter house. The more detail you give the builder, the more likely you are to get what you want.

There are 2 main categories of specifications. There are functional specifications that specify what the product is supposed to do. Then there is technical specifications that tell how the product does what it does. The technical specifications are much more detailed and without specific knowledge of the technology being implemented, you should avoid trying to get to this level. A technical person generally needs to write the technical specifications.

When developing your functional specification, you will need to specify the requirements. Requirements is what the application must do. You can list these out.

Use cases are a way for defining how a subset of the functionality works from the perspective of a user. For example, let's talk about the how a user logs in to a web application. Your use case could be stated as:

> *The user opens their browser and navigates to http://somedomain.com. The user clicks on login button. The login form is displayed to the user. The user enters their username into the user text box and their password into the password text box and then they click on the submit button.*

For every interaction your user will have with the system, you should have a use case. Notice that the use case specifies what the user is interacting with.

In order for the developer to have an idea of what you are expecting, you should create a prototype. This prototype is a way giving the developer what you are expecting it to look like. A prototype (also called mockup) is much cheaper than getting working software to this point and finding out that it will not meet the user's needs. A prototype could be as simple as drawing on a piece of paper or on a white board and taking a picture.

I prefer to use Balsamiq Mockups to do my prototypes because they have a wide variety of components I

can use to convey what the app should look like. You can also take screenshots of components or parts of screens that you like and assemble them to look the way you want your screen to look. Others prefer to use PowerPoint or MS Word to create their mock ups.

This is just a brief explanation of a large topic. I plan to update this document to include more detail and examples in this section. For now, if you are interested in more info, drop me an email at jwalling@wallingis and ask for more detail. I will send you a 26 page specification that I wrote for a product and will add you to the list for more information on this topic.

FINDING A DEVELOPER

When you are looking for a developer you should keep the following in mind.

- Experience
- References
- Communications skills
- Availability
- Cost

First off, the developer should be experienced. Experienced developers have already made a bunch of mistakes and hopefully learned from them. The inexperienced developer will be learning on your dime. Experience will help them make the right tradeoffs for your situation.

While the new developer may be a quick learner and may be willing and eager to do the job at what seems like a good rate, you will pay for the inexperience in the long run. The odds of the project running way behind schedule and having hard to correct costly issues after release will be much higher. You may find that your project costs you more; especially if you have to then bring in the experienced developer to fix the issues.

Don’t believe everything you see on a resume when you are gauging a developer’s experience. Sometimes the fact that someone can spell C# qualifies that for being on a resume.

So if you can't believe a resume, how can you determine their experience? The bottom line is that it is almost impossible for a non-developer to tell much of the time.

To increase your odds of finding someone with experience is to ask to see projects they have done in the past. Ask them questions related to their role so you can see if they were one person on a team that built the product. Who knows, they may have been the designer and know nothing about the back end development.

Ask them open ended questions like why they chose a certain architecture, language, or framework for this project. Then listen for their answer to see if they sound like they know what they are talking about. They may be spouting some BS, but at least you will be able to weed out those that don't have an answer and are totally clueless. You can ask the question without knowing the answer.

When looking at their resume or LinkedIn profile, get a feel for how long they have been doing development. Also look at the type of development they have been doing. While you may be OK hiring someone with little experience for a $100 WordPress plugin project, I would recommend looking for someone with at least 5 years' experience with the technology used for your project.

After you have done the above, an added step that should be done for large projects is to find a technical

person to screen the candidate for the necessary technology.

In addition to having the developer show you examples of what they have done, have them give you references. Connect with them on LinkedIn and see if they have testimonials from satisfied clients. You might also note if they have testimonials from co-workers who think highly of the developer.

Communications is probably the single most common reason for failure of larger projects. You need to be sure that you will feel comfortable talking with this person and that they can convey the technical issues to you in a way that you understand. If every time you talk to the developer your eyes glaze off because you have no clue what he is saying, it is time to find someone else.

If you are outsourcing your project to another country, you need to work extra hard on the communications part. While the costs may be lower depending on the country, they won't be as much lower as you think when you factor in the extra time and effort you spend on the communications and project management.

I have outsourced projects to lower cost countries in the past and have felt like I was being ripped off. The hourly rate was ¼ that for a local developer. However, it took 8 times as long as it would have taken me to do myself. I had no idea on whether or not the time they were spending was legitimate.

While most of my team is local, I have developed an effective process for hiring and running successful projects with developers from overseas. I'll go into that process in a while.

Next, you should determine the developer's availability. Will they be available when you need them? Do they have another job? Are they in a different time zone? When will you communicate with them? You should have answers to all of those questions and keep this availability in mind. Otherwise you could lose 2 days over something as simple as "What color should this button be?" when your schedules conflict.

While you can use a part timer on a small project, do not ever hire one for a large project. I used to frequently hire moonlighters for projects. They would tell me that they can put in 20 to 30 hours outside their day job. It was rare when one of them ever met the 20 hours. Typically I got 10 to 15 hours. It takes a long time to complete any decent sized project at this rate.

I put cost last for a reason. If you can't find an experienced developer that you can communicate with and that is available to you, it doesn't matter what the cost is. The project is destined for failure, so you might as well save your money. If you can meet these requirements, then you can worry about whether it meets your budget.

While local developers may charge more than someone in India, they are also easier to check references on. If they are readily available and will meet with you, they also have to look you in the eye and tell you how the project is going.

I'm not suggesting that you can't have a successful project working remote as even with our local projects, most of our communications is via Lync, Join.me, email, Visual Studio online, and phone. We have had many successful projects for clients not in our region, but we have to work extra hard to manage the relationship and gain the trust. If you find a developer that understands that concept, it can work.

As I mentioned earlier, I am going to cover my process for finding and hiring independent contractors. Your mileage will vary, because I have experienced local developers and use these outsourced developers to supplement the local team. In other words, I am not at the mercy of a single bad hire and have others to help me get things right when a project starts to go bad.

We use ODesk to find new developers, keep track of their time, and to pay them. I have worked with some of the other competitors, but ODesk is still my favorite.

One of the keys to success of our process is that I do not typically hire a developer for an important project on a deadline. Well in advance of a need for a developer, we will create a job posting for a small test project. This is usually a difficult project and uses the

technology related to some upcoming need. The project will be for a fixed amount, usually $100 to $200, which is big enough to have them develop something real. Often, this is something we could use internally, but can afford to have the project fail.

Let me reiterate the above point that this is not a product we have promised someone that we would deliver. This is important because I would consider the majority of these projects a failure. I do not want this project to reflect badly on us to a client. I generally give the job to 4 or 5 developers. 30% won't complete the project 30% will not do acceptable work in a timely manner, 20% will be successful, and 20% will do outstanding.

Those results are despite my screening the contractors for applicable experience, English skills, availability, and regions. There are certain regions I have better luck with than others.

When you create a fixed bid job post, you should go into great detail about what you want the software to do. Mockups and specifications make it easier for the contractor to come up with an accurate bid. At the end of the job post I specify that the applicant must start his application with some saying. For example, I might state in the post "at the start of your application include the words 'Fred Flintstone'.

While ODesk has done a great job of reducing the number of posts from companies that go and apply for every job, this will further weed them out and will also

weed out those that aren't paying attention or don't understand English well enough. Then when I am going through the list of applications and don't see 'Fred Flintstone' at the start of the application, I reject that application for not following instructions.

When you first start out on ODesk, it will take a while to get some traction and to get the best candidates to apply for your job. Just as you will be looking for candidates with a good history with good reviews, they are looking for that type of employer. The more money you spend on ODesk and the higher your rating, the more candidates you will have applying.

When you are posting the job, you will have the option of allowing all contractors to apply or to limit it to those that are invited. Initially you will want to allow everybody to apply. You can also go out and invite some that look like they have great skills.

When filling out the job posting, be sure to fill out the section on preferred qualifications. I generally choose the following qualifications:

- Independent contractor
- At least 4.5 feedback score
- Eastern Europe or North America depending on my needs
- Fluent English

A contractor is not required to meet all of these preferred qualifications to apply, but on the application you can quickly see how many they meet.

When a developer makes it through a test project and does well, I will then look at adding him to the team. I create a new project, this time I do not let everyone apply. I limit the applicants to those I invite. Then I invite the developer that just completed the successful project.

Development Contract

If you are going to work directly with a client, they can typically provide a contract. However, you need to run this by your lawyer to ensure that it is a balanced contract. By default, the copyright belongs to the software developer unless you have a contract which specifies otherwise.

If you are using ODesk, the terms of service includes wording such that upon full payment the copyright and other rights are transferred to the client. If you have a lawyer, you might have them review this.

Since I am not a lawyer, I'm going to keep this chapter short.

Communicating with your developer

Communications is key to ensuring that your project will be successful. It keeps you up to date with how the project is proceeding and also shows the developer that you care about the project and are willing to get involved to ensure its success. Fast feedback to the developer is important on keeping things on track. It is also a means for holding the developer accountable.

Communications can be done through numerous methods such as phone, Skype, email, project planning or issues tracking tools, etc… The key is that you get regular updates and that you have a central place to keep track of features and defects. I require daily status reports from the developer even if it is only a sentence in an email. If a developer is not going to work on a given day that they are normally scheduled to work, I expect a quick note that they will be unavailable so I'm not trying to reach them to get an answer.

If you are using ODesk, which was mentioned above, you can use the messaging system for passing info back and forth. However, it is not a good means for keeping track of features and defects (bugs).

For keeping track of features and defects I prefer Visual Studio Online because I use many of its other features. I will discuss some of those other features in other sections of this guide. There is a free version of

Visual Studio Online for small teams of 5 or less. For more info go to https://www.visualstudio.com/

If you aren't going to take advantage of some of the other features of Visual Studio Online, you might also look at JIRA at https://www.atlassian.com/software/jira. It is an excellent product.

For really simple projects, you do not need these project tracking systems, but if you have a more extensive program you are having written, you will need to have some means for keeping up with the features and defects other than email.

Another aspect of ODesk that I find extremely valuable is the way that it can track your developer's time and take random screenshots of them working. I look at those screenshots for 2 main reasons. The first reason is to see if new developers are working on projects other than mine during the time that I am being charged. If you see them on Facebook instead of writing code, you should bring this up.

The second reason for reviewing the screenshots is to see if they are heading down the wrong path. If they are working on something and you can see that it does not look like what you are expecting, it is best to catch it early. Otherwise they could spend more time following the wrong path and it will take longer to fix the issue. You end up paying twice.

Do not use these screenshots as a gauge of productivity or you will drive yourself and the developer crazy. Measure your developer's performance based on his ability to meet the expected targets in a cost effective manner.

As well as providing frequent feedback, make sure that you are giving positive feedback when warranted. Developers like hearing good things. All too often, all the developer hears is "this is broken" or "this doesn't work right". Try to give at least as many compliments as you give criticism. It makes for happier developers who will try even harder to please you.

If a developer does something that makes you think "this is great", as well as telling the developer that, send them a bonus. On ODesk, these bonuses get you good reviews and these good reviews help you get the top performing developers wanting to bid on your projects.

KEEPING PROJECT ON TRACK

In the last chapter we covered the first step in keeping your project on track and that is good communications. You should get frequent, regular updates from the developer.

Try to have working software as early in the process as possible even if it has minimal functionality. Ask the developer for demos early in the process and if it is a web application, a version that you can play with as soon as possible.

As was discussed in the previous chapter, use feature and issue tracking software to keep the project on track. Regularly review the status and make sure the developer is in the habit of keeping the work items updated and passing them along to you for review when done.

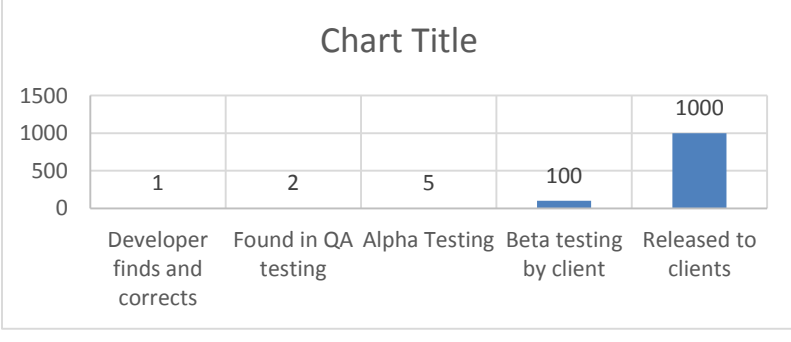
If the project is starting to fall behind, make sure to get involved early. Try to understand the issues so you can see how you can help. Is the issue that the developer really doesn't understand what he is supposed to do next. When people aren't sure what to do on something they often procrastinate and work on something else that they understand. If the issue isn't with an understanding or a technical issue, is the developer having other problems, possibly personal, that is keeping them from focusing on the application?

Building Quality Into Your Product

Needless to say, if you have a lot of bugs in the software, your customers won't be happy. Since developers are notoriously poor at testing their own code, it is up to you to ensure that the program gets tested. This might be you doing the testing or might be you hiring a QA (Quality Assurance) tester.

Key to dealing with defects is that you find and get them fixed early. The longer it takes to find a bug and the further it gets into the development life cycle, the more costly the defect.

The chart below shows the relatively cost to fix a bug at different points in the lifecycle. Note that it is not linear and the cost increases significantly the further the defect gets in the process. Think of all the people that get involved later in the cycle.



Factors related to cost

- Time for developer to fix
- Number of people involved
  - Developers
  - QA
  - Client – multiple people
  - Managers from both organizations
- Cost to correct the issues resulting from the bug. The program may have created and saved bad data which is then hard to go back and clean up.
- Interruption of normal development cycle
- Ill will generated
- Cost of lost customers

When is Software Done

In one of my jobs, there was a Vice President who we'll call Tom. Tom spent most of his day dealing with hardware projects where they would order the servers and within a month or 2 they would have the servers in and have them installed. Tom didn't have a strong grasp on how software development works; or if he did, he sure fooled me. Every time I would run into Tom he would say "when is it going to be done". At first I thought he was kidding and I responded with how far along we were on the current phase. This went on for years. Finally, when I realized he wasn't looking for a status update, my response became "it is never done, because if it is, we are done."

The meaning of the above statement is that you can't develop a piece of software and then do nothing else with it and expect it to continue making you money in perpetuity. This is a common misconception of many people who have not been in software development. When you stop developing software, others will develop products to supersede yours or the product will eventually break because your platform is no longer supported. If you create a good product that is making you money, instead of jumping to the next great idea, you should consider adding more functionality to your existing application.

On the other hand, if you are having a difficult time keeping a product profitable, then it might be time to let it go. It doesn't make good business sense to lose money on a product. Sometimes it may be hard to let

“your baby” go, but you can find another product to create and fill the void.

About The Author

Joe Walling

Find out more at http://wallingis.com

If you need help with a project, you can reach me at jwalling@wallingis.com or 864-214-2748