Why read this article?

If you do undertake software development I would like your project to be successful. This article is my attempt to provide you with a concise summary of the best advice garnered not only from my experience but perhaps more importantly from the advice of many of my peers more experienced and lucid than myself.

Therefore I have taken the time to research and write up what I feel will be of benefit for you to know about software development. I trust the time you spend reading this article will save you from the considerable emotional and financial pain that many others have experienced.

Why? Because software development is not for the feint of heart, the impatient, the poor communicator, the unrealistically demanding, the person who has no time to properly communicate their requirements or who changes their mind every 5 minutes. If the honest truth contained herein is enough to deter you from software development it is far better done before you start rather than part way through a failed project.

Software projects rarely fail due to poor programming; it is usually poor project management.

Even an experienced climber would not even consider trying to scale Mount Everest without an experienced local guide. Let's face it, most of us wouldn't even consider scaling Mount Everest, period! Similarly you should not consider embarking on a software project without at least a list of the most common traps and pitfalls that exist and some sort of strategy on to how to avoid them. That way you can better handle your end of the project management.

The article is broken down into these sections:

Terminology

It will help your understanding if you have the same definitions of these terms as the author while you are reading this article.

Analyst
The person responsible for gathering the data from the client as to what the software is required to do and not do.

Bug
An improper functioning of the software- a defect

Business Expert
The member of the client team who totally understands the client's business and the reasons why the client's processes and procedures are the way they are. Preferably is able to authorise changes to those processes and procedures if better alternatives are discovered during the development process.

Client
Collective term for the group of persons and their functions that may reside with one individual or over a team of individuals who commission the software and with whom the developer works to create the software.

Defect
Minimally, when the software does not accept valid input, returns erroneous or no output in response to a valid request, has data entry fields not arranged sequentially. Depending on the level of quality desired may also include more stringent requirements such as rejecting erroneous or inconsistent input, allowing access to unauthorised personnel, inadequate performance, procedural inefficiencies etc.

Developer
Person or organisation responsible for designing and building the software.
Synonym:- Programmer

Functional requirement
Some thing the software is required to perform, such as compute the sum of the invoice detail lines.

Non-functional requirement
An attribute of the software that does not relate to a function. Can be a colour, speed, security, stability or other aspect.

Programmer
Person responsible for designing and building the software. Synonym:- Developer

Requirements
Items required of the software by the client, can be functional and non-functional.

Scope

A statement of what you want the software to do and what you do not want it to do. Example: The software will provide browser based access to an externally hosted, database replicated in three locations, allowing authorised users to store and retrieve names, positions, status, status date, personnel who reported the status, telephone numbers and email addresses of all staff to be contacted in the event of an emergency. It must be automatically be updated in the event of a change in the corporate database and have no financial information contained therein. It must be able to be satisfactorily viewed from a mobile phone or hand held device.

Software development

The best definition I have seen of software development is from Alistair Cockburn's book "Agile Software Development" published by Addison Wesley, a book I thoroughly recommend to all considering embarking on the process of commissioning software development. Alistair says:

"Software development is usefully viewed as a cooperative game of invention and communication with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game."

(The next game can be modifications or extensions to the initial software, a rewrite of it or the development of the next program.)

Specification
A comprehensive list of functional and non-functional requirements of the software.

Technical writer
The person responsible for creating the software documentation.

Tester
Person responsible for testing and documenting defects for the developer to fix.

Three point estimate format
A format for providing time and cost estimates for software development. Best case (shortest possible time), most likely case and worst case (longest conceivable time).

User
Member of the client's staff who does or will use the software. Preferably one who is experienced in the processes the software will be performing.

Why do software development projects fail?

Peter Simpson, a world renowned, Melbourne based management consultant, on hearing of my decision to embark on a career as a software developer remarked, “Gee, that’s a thankless task!”

When queried as to the reason for his uncharacteristically pessimistic response he replied, “Because no one is ever happy with it.”

According to research from the Standish Group apparently only 11-16% (depending on size) of software projects finish on time and under budget. Other estimates put it as low as 2%.

The Standish Group report “Unfinished Voyages” even went so far as to assign a point value to their identified success criteria:

SUCCESS CRITERIA	POINTS
1. User Involvement	19
2. Executive Management Support	16
3. Clear Statement of Requirements	15
4. Proper Planning	11
5. Realistic Expectations	10
6. Smaller Project Milestones	9
7. Competent Staff	8
8. Ownership	6
9. Clear Vision & Objectives	3
10. Hard-Working, Focused Staff	3
TOTAL	100

Some estimates put it as high as 75-80% of software projects are scrapped before or shortly after delivery as they fail to meet user requirements.

In a Software Engineering Australia article titled "Little gain from past software disasters", Mr Derrick Brown cites the reasons for software failures as being:

1. Unclear business objectives,
2. Ill-defined scope,
3. Unrealistic time-frames,
4. Poor communications,
5. Irrational senior management promises,
6. Poor business commitment,
7. 'End-date' driven projects,
8. Poor planning,
9. Lack of appropriate skills.

In another article in the same magazine Terry Wright says:

"There are six reasons that projects have scope and cost blowouts. These are:

1. Lack of user input
2. Incomplete requirements

3. Changing requirements
4. Lack of executive support
5. Technology incompetence
6. Unrealistic expectations"

The following are two statements made by Tom Seibel in an interview conducted while he headed a $2 billion a year software company that makes a CRM (Customer Relationship Management) product.

Interviewer: What kind of problems do you see most often?

Seibel:
There are software anomalies;
there are documentation anomalies;
there are project management problems;
there are change management problems;
there are operating system problems;
there are networking problems and training problems.
In projects of this size, believe me, there are always problems.
And this is not specific to CRM.
This is any large information technology project.
This is what the world's really like.
So we have a very rich support infrastructure to support our customers and get them through all these things.

Interviewer: The response I hear to the question of why software projects fail is that the customer did not plan well or didn't delegate well or made poor decisions. Is that really fair? Shouldn't the company selling a product that costs millions of dollars shoulder some responsibility for its ultimate usefulness?

Seibel: Absolutely. This is the point I'm making. We don't see it as our obligation to simply deliver bug-free software. We do whatever it takes to make sure the customer succeeds. So absolutely, I think it should be the software company's responsibility. And this is the responsibility that we take upon ourselves.

Attaining software development success

In order for your software project to be successful you need to know the main reasons for failure, cost and time blowouts. You can then list the opposites to the failure points as targets that need to be obtained to ensure project success. Then you can focus on and direct your efforts towards attaining those targets.

The more of these success prerequisites you are in doubt of your ability to attain then the greater must be your degree of caution in attempting the project or your risk mitigation strategy.

REASONS FOR FAILURE	REQUIREMENT FOR SUCCESS
1. Unclear business objectives	Clear business objectives
2. Ill-defined scope	Tightly defined scope
3. Unrealistic time-frames	Realistic time-frames
4. Poor communications	Good communication
5. Irrational senior management promises	Rational promises
6. Poor business commitment	Strong business commitment
7. 'End-date' driven projects	Balanced deliverables
8. Poor planning	Thorough planning
9. Lack of appropriate skills	Appropriate skills held

Regarding the six reasons that projects have scope and cost blowouts.

A. Lack of user input	Early and frequent user input
B. Incomplete requirements	Complete requirements
C. Changing requirements	Static requirements
D. Lack of executive support	Strong executive support
E. Technology incompetence	Technology competence
G. Unrealistic expectations	Realistic expectations

Let's now take up each of the items found necessary for a successful project and discuss the recommendations. Most of these are actions that the developer undertakes, either internally or in our interaction with you. Some of these are actions or attitudes only you can take or maintain.

1. Clear business objectives

As the first step of the project you need to clearly identify the business objective you have in mind that prompts you to consider the software project. If you've a mind to do so you should also quantify the benefit expected from the project. How else can you make a cost\benefit decision?

2. Tightly defined scope

The Scope document should not only include what you want the software to do but also what you do not want it to do. This minimises scope creep by forcing a review of the project scope by senior management if someone wants requirements included not covered in the original scope document.

Not all projects fall into this category but what you want the software to do can sometimes be grouped into stages. Each stage becomes a deliverable - something that can be delivered and used when completed. Therefore you see an initial benefit faster and the total project is implemented progressively.

3. Realistic time-frame

The old saying, "Rome wasn't built in a day." applies just as well to high quality, well documented software. It takes time to design, consider all possibilities, construct and test software to ensure quality. Software development takes as long as it takes. It doesn't take as long as the developer predicted, it takes as long as it takes. It doesn't take as long as you: anticipated, predicted, would have liked, think it should take or budgeted for. It takes as long as it takes.

Software development is not like stamping out parts from a sheet of metal. It is a creative process that relies heavily on communication between client and developer and between developer, product and tools.

Steve McConnell, a developer and author, once said "Estimation is a Black Art."

The bigger the job and the less precise and detailed are the specifications, the less chance the developer has of giving accurate estimates of how long it will take. Even trying to use exact times on small tasks is fraught with danger. A small job can be estimated to take an hour but may end up taking two days if the developer experiences a problem with the program performing a legitimate command in certain circumstances. It is far more practical to use time frames expressed in such fashion as best case (shortest possible time), most likely case and worst case (longest conceivable time). To prevent repeating those hereafter I will refer to that as a three point estimate.

That is a far more complete and accurate answer to the question, “How long will it take?” than a single number of hours, days, weeks or months.

4. Good communication

A great deal has been written and taught on the subject of communication but the bottom line is, "The developer needs to know your business as well as or better than you do if they are going to help you automate it."

To this end there needs to be an incredibly high level of communication between you and your staff and the developer. If at any time there isn't, that needs to be remedied as a first priority. That is part of the reason for this lengthy document, to fully communicate the potential pitfalls and the processes you and the developer need to implement to stay out of them.

5. Rational senior management promises

The foundation for rational promises is built on allowing (or insisting) that the developer(s) report the anticipated time frames as per the three point estimate format and that all levels of personnel associated with the project fully understand and agree with that format and maintain their integrity in reporting and advising others that way.

6. Strong business commitment

As long ago as 1513 Machiavelli was quoted as saying:

"There is nothing more difficult to plan, more doubtful of success nor more dangerous to manage than the creation of a new system.

For the initiator has the enmity of all who would profit by the preservation of the old system and merely lukewarm defenders in those who would gain by the new one."

If the client does not have a strong business commitment to the development of the software then the first pebble in the road has the potential to totally derail the project. And whilst most developers make every effort to minimize the risks, stuff does happen.

Lack of senior management buy-in

If the boss is not prepared to back a project with his own time and energy then there is a strong chance it will not be a successful project. If he delegates the management of a project to someone else, the unspoken communication that goes with that delegation is that the project is not important enough for the boss to spend his time on it.

As a result of his non-involvement the boss has too little personal involvement with the project to know enough early enough about the true causes for the bumps in the road to correct them.

Early intervention and correction is the best way to solve problems, get the project back on track and avert a failure.

As soon as the junior executive to whom he delegated the project hits a snag with the project then the junior can recommend the project gets cancelled and the boss is most likely to accept his recommendation because he has no personally observed data to go on.

7. Balanced deliverables

End date only driven projects are less of a problem in our experience but they do crop up. When the "go-live" date is of critical import it becomes vitally important to the project success to prioritize the functionality and aim to deliver the most important functionality in the first cut and defer effort on the discretionary or less important functionality until after the initial deadline.

8. Good planning

Probably no developer has ever had a client deliver a complete set of prioritised, specifications detailing all the functionality required to be in the completed product and probably never will. It is, after all, an integral part of the project and therefore the developer's job, to Scope and Specify the functionality required. Successful software project planning requires not only an intimate knowledge of

software development but also your business processes, your priorities, your budget and time constraints as well as access to your key people and end users.

9. Appropriate skills held

Skills are not ascertained merely by reference to a piece of paper. They are best assessed by products - "the proof of the pudding is in the eating". Ask to see some of the developer's past work to see if it evidences the skills required to successfully complete your project. Sometimes your preferred developer has not completed a similar project or cannot evidence the skills but you have confidence in their ability. In this case commission them to build a working prototype for your project or some portion of it, preferably one of the more complex parts of it, so you can ascertain for far less than the cost of a failed project their ability to produce the result you seek.

As for your end users of the program, if they are currently not computer skilled, consider enrolling them on some internal training or on a course at TAFE etc.

To ensure your staff get the most out of your investment in the software decide what level of help documentation and/or on-site training is appropriate to your requirements and budget.

A. Early user input

In all but the smallest of organisations you will have senior management who have a strategic and policy focus as well as front line workers who use the program daily so are very much tactically focused on the details.

It goes (almost) without saying that the software must conform to the strategic objectives of senior management but if it is to be used it must also deliver the required functionality to the end users.

As much as senior management may feel that they know the requirements of their end users, experience has proven that no matter how low their status, the involvement of the people who will use the software is vital to the success of the project.

The sooner the key end users are involved in the "look and feel" of the application and are able to review the critical functionality, the sooner they can provide the vital input necessary to keep the project aimed squarely at the bullseye and the less developer time will be wasted going down blind alleys. That is why user feedback early on in the software development process is vitally important not only to its eventual success but also in minimising the project cost.

B. Complete requirements

It is critical to the success of the project that all major functionality requirements are communicated as early as possible and mention of them is incorporated into the Scope document and that they are detailed in the Specifications. This requires that all key concepts be known by a combination of the client's business expert and user representatives and that those are clearly communicated to the software developer's business analyst\programmers.

C. Static requirements

This is a two edged sword as too rigid a set of requirements can leave any important omissions as an Achilles heel in the acceptance of the software and too flexible a set of requirements can see it abandoned and never complete.

Why can too rigid a set of requirements be a problem? Within the world of software development there are several methods or processes by which the software can be constructed. By far the more prevalent has been what is described as the "Waterfall" method whereby the process goes through set, distinct stages in a fashion similar to an engineering process:

Gather Requirements
Design
Construct
Test
Document
Deploy
Maintain

While it sounds good and appeals to the more formal and disciplined side of human nature it has been observed that there are fatal flaws in this approach that result in 50% of projects created In this fashion never being used. Despite these risks it is still the preferred development method in certain scenarios such as replicating existing software or in creating life support systems or when adherence to a formal methodology is mandated by legislation.

While we could in detail cover the many reasons (imperfect communication between client and developer etc.) advanced for the intolerably high failure rate using the waterfall method that is not the purpose of this article. For our purposes it is probably sufficient to say if we want to gamble there are casinos, if we want to commission successful software projects then there are far less risky methods to employ.

D. Strong executive support

This is substantially the same as item 6 above however I will add one anecdote. I read in the daily press recently an interview with a CEO of a company who said he personally supervised the software implementation project as it was too important to leave to anyone else. If you are considering a mission critical software project I commend that viewpoint to you.

There are very few organisations I know of that are overstaffed. There are production demands that are urgent enough to keep personnel busy the whole day long and often into the night. Throw a software project on top of a busy schedule with no allowance and something will give. Part of strong executive support consists of ensuring that fast access to various personnel is provided to the developer. This includes senior personnel who know the high level business requirements down to front line users who can input valuable data on the user interface requirements.

E. Technology competence

There are people who learn computer related skills quickly and easily and others who learn them slowly, if at all. There are generally available courses that will help and the developer can often provide drills and specific training on how to use the software they develop.

F. Realistic expectations

There are a dozen ways to phrase the underlying rule that unrealistic estimations violate, "You can't make a silk purse out of a sow's ear", "You get what you pay for", "There ain't no such thing as a free lunch" etc. but the bottom line is that some people are more realistic than others.

There is a technique you can employ to improve how real your estimation is. The greater the number of people involved in the estimation and prediction process, given that they are allowed make independent, unbiased input, the more likely the aggregate expectations of the group will be realistic. This is because the inadequacies and inaccuracies of each individual's input are averaged out of the aggregate result.

If you can take a critical, unbiased look at your time estimation history, personnel assessment and success at attaining your goals then that will give you an idea of how well you personally should expect to fare with the reality of your estimations.

This is not to say that you should not try to do better than you or others have done in the past. A small percentage of us try to push our expectations into existence despite those expectations not being held by the majority. This is how benchmarks improve. Some people have a high rate of failure in this, some have better success.

If a person continues to bulldoze others into adopting or agreeing with their unrealistic estimation or they ignore danger signals, warning signs or outright disagreement with their estimations then they risk being very wrong and also losing the backing, trust and agreement of others.

Software development – the process

Estimation

Every client has a tall wish list and a short budget. In our experience nobody ever has as much money as they need to pay for all the bright ideas for software functionality that can be dreamed up for their business.

Before initiating a project all but the most well-heeled of clients (in our experience that's everyone) will require some sort of answer to their primary two questions, "How long will it take?" and "How much will it cost?" The only way to answer that question is to formulate some sort of an estimate of the effort required to build the software.

An estimate is just that, a best guess based on the data available to the estimator. The data gathering can be as short or informal as a five minute to two hour conversation to get a very rough ball park estimate or two months of detailed investigation and analysis by a team of specialists to give a detailed estimate.

Obviously the more time you spend detailing all the functionality required and estimating the time it will take to build it, the more accurate should be your estimate. One programmer told me that a large organisation for which he worked held as a rule of thumb that to accurately estimate how long the project would take to complete would require 50% of the development time. So a 1,000 hour project would take 500 hours of work to accurately estimate.

Whatever the size of project and regardless of the development method used there needs to be a requirements elicitation up front in order to be able to estimate the development effort and provide a three point estimate answer to the "How long and how much?" questions.

Small project sequence

Because everybody is an individual and software is as much artistic creation as it is engineering, there are probably no two software developers who do everything the same way, so what follows is by no means the definitive statement on the subject.

Having said that the major steps through which very small software projects progress are usually accomplished by the "Waterfall" method, a one pass through of:

Scope	What are the major functional objectives and where are the limits
Specification	Getting the requirements
Design	The blueprint
Construction	Building it
In-House Testing	
Documentation	
Implementation	
Training	
User Acceptance Testing	
Post Installation Modifications	

This can be expanded out to:

Define the scope of project and Cost\Functionality\Go Live Date priority
Write the functional and non-functional (look and feel, performance) specifications
Provide three point (best, most likely and worst case) cost and time range estimates
Create proof of concept, workflow, construct "look and feel" templates
Get user feedback on "look and feel" templates, workflow functionality
Modification to design templates and workflow based on user feedback
Construct the application
Alpha test the application
Install the beta version
Train the beta testers
Beta test and modify\correct the application per testing requests
Obtain user sign off that software is ready to "Go Live"
Train additional users
Import the final data
Go Live
Complete commissioned application and help documentation
Modify or add functionality as requested

Larger project sequence

Larger (longer than a month) or more risky projects or those based on cutting edge technology may require additional steps, like prototyping, proof of concept testing, progressive or staged implementation etc.

Manifesto for Agile Software Development

I have been profoundly impressed by the good sense and workability of the ideas espoused by the founding signatories of the Manifesto for Agile Software Development. I include that manifesto here so you can better understand my bias towards lean software development methodologies.

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Incremental Delivery

Once you have a Scope and Specification, projects anticipated to take longer than a month will almost certainly benefit from dividing their total functionality into stages that can be delivered progressively. This is called incremental development.

According to the Standish Group report smaller project milestones is the sixth most important factor in project success. Working, delivered software is a great way to

allow the client to see that tangible progress is being made. Incremental delivery of working software maintains client confidence that they are receiving benefit for their progress payments and helps silence the “nay-sayers”.

By contrast to the “Waterfall” method, incremental delivery looks more like this:

Scope
Specification

Deliverable 1
Design
Construction
In-House Testing
Documentation
Implementation
Training
User Acceptance Testing

Deliverable 2
Design
Construction
In-House Testing
Documentation
Implementation
Training
User Acceptance Testing

Repeat for deliverable 3 etc.

Post Installation Modifications

Iterative Development

Iterative development is where the functionality of some part of the software is built on or expanded with each incremental delivery. The key or core functionality is delivered first then the additional functionality found most needed by the users is added in successive incremental deliveries.

This is one way to get users actively involved in the process of limiting the functionality to only that which they will use rather than spending a lot of time developing a large amount of functionality, much of which will end up not being used.

A comment on iterative development as a risk reduction strategy: "...short iterations are an options-based approach to software development. They allow the system to respond to facts rather than forecasts. There are few endeavours in which it is more important to keep options open than in software development. ...options-based approaches are fundamentally risk-reduction strategies, and as counter intuitive as it may sound, you actually reduce your risk by keeping your options open rather than freezing design early." p 28 'Lean Software Development' by Mary and Tom Poppendieck

A General Comment on Software Quality

While quality as a subject may deserve its own section of this document, quality software is not accomplished that way. Higher quality is far cheaper in the long run than lower quality and is obtained by addressing quality impacting factors all the way through the development process. According to one article, on average, every $1 spent on quality control returns $10 in saved costs.

Time and again surveys show that the time you spend earlier in the project, cross-checking and verifying basic assumptions and high level design concepts is amply rewarded by lowering rework and defects.

A design error is progressively more costly the later in the project it is discovered.

Considering some estimates put rework at 40% of the development cost of a project, you can see the enormous potential to be gained here.

The Desirability of Early Feedback

No matter how formal the requirements gathering process or how meticulously performed, the analyst\developer will never get everything 100% right the first time. If that seems to you to be an unreasonable statement look at car door handle designs. It took over 70 years for recessed door handles to be widespread and how many more for air-bags?

Most users cannot conceive of and communicate to the analyst what they want and how they want it to look but they can tell what they do or don't like when shown it.

Recognising that, it makes good sense for the developer to spend as little time as possible crafting something that the user can comment on to see not if, but where, there have been gaps in the communication process.

Design drawings or non-working prototypes can be created for a fraction of the cost of working software. Ask the developer to create drawings of screen designs, report layouts and menu structures before spending a lot of time creating them. You can then show these to the potential user in order to detect early (therefore cheaply) how the design should be altered to better cater for the user's needs.

Once the drawings are accepted you can do the same thing with non-working prototypes taking the verification to the next level prior to spending the time coding the functionality.

The Value Payback of Early Feedback

The better the review and verification process early in the design and coding (the tighter the feedback loop), the higher will be the quality of the completed product, by orders of magnitude. The most effective way of improving software productivity and shortening project schedules is to reduce defect levels. Design defects are the most expensive to correct as they occur early and affect all that follows.

Obviously the law of diminishing returns dictates that there is a practical limit to the amount of time spent cross-checking and verifying but in my experience this limit is rarely approached in software development.

How to Accomplish Early Feedback

The way to accomplish the shortest possible feedback loop is to lock all parties in a room at the beginning of a project and unlock the door at the end of the project. If the client is a one man business and the developer is a one man business, that's the two of them. If the client's business is large that can mean one or more business experts and users from different departments. If the developer's business is large that can mean the analyst, developer, software tester, technical writer and trainer.

That's not going to happen but you can see that the closer you get to that ideal then the shorter you make the feedback loop. The shorter the feedback loop, the fewer design errors, the less wasted time, less rework, less defects, lower cost and better quality software obtained.

If you can ensure that all levels of staff are available on demand to the analyst and developer early in the project and the developer has a strong cross-check and feedback loop, then you stand to minimise design defects and resulting rework.

How much administration is enough?

In 1970 Winston Royce wrote that the fundamental steps of all software development are analysis and coding. "[While] many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development cost." 'Managing the Development of Large Software Systems' by Winston Royce

The more formal the administration of the software development project the more the amount of time will be spent on administration and the greater will be the cost. Therefore you need to establish up front what you will be prepared to tolerate as far as a balance between procedure and cost.

For instance if every additional feature needs to go through a formal impact analysis and approval process then it takes far more time to document that than would a quick telephone conversation between developer and client.

A project with a very high administrative overhead can cost up to ten times as much, perhaps even more, for the same functionality as a project with minimal overhead. In some organisations the productivity per developer is as low as 1,000 lines of code a year. On a top day I can write 1,000 lines of code. Might take me all the next day to debug it, but nevertheless it is truly remarkable the differences that administrative overheads make.

If you want to know in detail how each developer spent each amount of time, exactly how far through the project you are at any point in time then it will take more resources to log that data and communicate it to you, therefore the cost to you will be higher.

Factors that you may wish to take into consideration are:

Project size (a larger, longer project may need more administrative overhead so you can better see where in the cycle you are actually compared to planned),
Client size (a larger client will usually require more process and administrative overhead than a smaller one),
Legal requirements (there may be legislation mandating certain records or processes),
Financial risk or life threatening risk from software malfunction (medical equipment software will require more stringent documentation processes)

Just as a real world example on how this pans out in practice consider this story.

My lead developer and I attended a seminar on practical project estimation. Part way through the seminar the presenter had explained various methods, top down (what other similar projects have you done and how long did they take), bottom up (list everything to do and add up how long you think it will take to do it). She then said there was a simple rule of thumb you could apply very early on to provide a rough guide. You add up the number of things about which you wanted to keep data (normally stored in tables) and multiply by x hours work.

She then asked if anyone had just completed a project for which they would be willing to share the details. Having just completed a project that we went 60% over budget on and being more keen to learn than appear perfect I offered it up.

We had been asked to rebuild a DOS point of sale system in Windows for a lady who owned 3 dress shops. She said she only had $10,000 to spend and we figured if everything went well we could get it done in that time. Well everything didn't go well and it took about $16,000 worth of time. We held to our promise and only charged her the $10,000 but we were smarting from the misestimation.

When we divulged this to the group there were audible titters. The presenter then extrapolated the number of tables in the project and announced that using the guidelines it should have been a $90,000 job. It was our turn to titter.

Part of the reason we were able to do the job in less than 20% of the time allowed for in the estimating guidelines was that next to no administrative overhead was consumed. Nearly our entire effort was on producing working code.

Software quality

Quality Areas

Software quality can be assessed over many areas. There is freedom from defects, stability, reliability, security, data integrity, fitness to purpose, aesthetics, speed, the list is long.

Obviously software that controls life support equipment, nuclear reactors and financial software processing millions of dollars worth of transactions need to be of the most stringent quality with regards to accuracy, reliability and stability while a five year old child's colour\shape matching software needs to be attractive and colourful to capture and retain attention. Attractive and colourful don't even rate a mention for life support equipment and nuclear reactor software.

Ensure that you have an understanding of each of these areas relevant to your project and the desirable level of quality in each area.

Quality Levels

It has been said that there is no such thing as perfect software. You need to choose and specify up front which level of quality is appropriate to your project. That way the developer knows what your expectations are and can plan to meet them.

It is important for you to specify the level of quality that is appropriate to each area of your project for several reasons. The developer needs to know so he\she can plan appropriately to meet your requirements. Specifying too high a quality in any area incurs costs disproportional to the benefit obtained. Specifying too low a quality can result in software unusable for the purpose for which it was intended.

Resist the inclination to specify the best in everything. Determining too high a quality in an inappropriate area is a total waste of time and effort. There are still green screen, dumb terminal systems designed decades ago that are adequately performing non-trivial tasks today. They are ugly and hated for it, but no less effective in returning the requested data.

Level 1

When testing a software product, one can test it in the manner of a trained user using the programme correctly and entering the correct data in the correct locations for a limited range of transactions. This is the first level of testing and is pretty simple to programme for and relatively fast to test. So if an experienced user who only uses the program in a specified correct manner does not require all the error checking, the coding time can be as little as one twentieth of what it would be if the developer were to take into account and cater for all the levels of user competence and knowledge and testing for data entry errors in the business processes being computerised.

A program tested to only this level is an appropriate quality for non-vital software used by competent, trusted and experienced personnel. This is inappropriate for all but a very small percentage of cases.

Level 2

The next level is where you want to detect and prevent inaccurate or incomplete data from being entered. More code has to be written to test that only correct data was inserted and to display error messages and send the user back in the programme sequence when they have done the wrong thing. It is not unusual for the additional code written to test all error conditions to expand the code by 5 to 20 times.

This level of quality is appropriate for more applications demanding a higher level of accuracy and less competent or trusted users. It is far more costly to attain as the code can easily be five to ten times longer and far more complex than the lower level described above. Despite these costs it is the appropriate choice in by far the majority of cases.

Level 3

The last level is where you aim to test all possible scenarios, every combination of uses and data, where you also assume the viewpoint that a total idiot user will press buttons randomly and perform any and all functions out of their normal sequence. As there are often more than a dozen fields on a form and a great many forms in a sophisticated application, there are a great many ways to do something wrong!

You may also wish to include things like failover, access security levels and timeout lockdowns etc. as specific items at this level.

So the choice has to be made between whether you want the developer to:

1. Test it briefly so that they can get the result specified if the data is correctly entered under the most common circumstances. This will leave gaps in the error trapping and handling that will only come to light under certain circumstances.

2. Take the time necessary to test the product fully under as many circumstances as the tester can think of, catering for all possible incorrect data entry combinations and deliver it as close to perfect as we can or

The first option starts out being cheaper but a good portion of the price difference is absorbed by us adding code to do the additional things you want done and you wear the testing hat which can leave even the most patient people feeling pretty frustrated at times.

The pros and cons of the second approach are that it will take longer and therefore the cost will be greater and we may be incorporating more error trapping than is necessary in your case but you will suffer less frustration and therefore be much happier with the installed software.

Defect remediation

“In producing a working program that meets a set of requirements, defect removal inevitably is a large part, if not the largest part, of the job.” Paul W. Abrahams

People who have never programmed will sometimes ask, “Why do I have to pay you to fix the bugs?” or “But it’s your fault it has a bug, you should fix it and not charge me.”

Answer 1

My short answer to this is to ask the person,
“Are you perfect or do you still make mistakes?”

To date everyone I talk to still makes mistakes. So I ask,
“Did you dock your pay for the time you took to correct the last error you made?”

If that answers the question for you then you can skip the rest of this section and go straight to the next.

Answer 2

If not, let me say this, in my 20 years of working with computers I have never yet seen perfect software or met a programmer who wrote perfect code, me included. Therefore it follows that a necessary part of software development is finding and fixing defects. If you don’t want to pay for one part of the process then the cost of that has to be included in what the developer charges you for the other parts.

If that answers the question for you then you can skip the rest of this section and go straight to the next.

Answer 3

A person who has never programmed cannot begin to imagine how frustrating it can be for the programmer to type a command that has worked before in other scenarios and have it fail for some inexplicable reason.

Or how demoralising it can be to be demonstrating a complex program and have 300,000 lines of code look like mud because a single comma was omitted when required or inappropriately included. Because that’s all it takes, one character out of place in (300,000 lines by, let’s say at least 20 characters a line = 9 million characters) 9 million and the program crashes.

If a programmer or tester inadvertently fails to test some aspect of the functionality it is not because they are necessarily sloppy, inefficient, slack or incompetent, it is because writing code is part science, part art and is incredibly complex. Sometimes a programmer can have 4 things running around in the head and finish three but forget a line from the fourth.

Sure, one could correctly argue that they need to keep better notes on the myriad of things they are doing, shouldn’t we all. Truth is, sometimes a developer will be deep in a train of thought and be distracted by a question, phone call, the need to make a bathroom visit, a bright idea that solves a problem that perplexed him yesterday, whatever, and lose track of the six step process he had in mind.

If you find yourself frustrated with your developer over bugs in software it can be beneficial to review some of the articles in the computer press. Like when Microsoft eventually released Windows 2000 it was, according to a leaked Microsoft memo, reputed to still have over 60,000 known and documented bugs or places where the code could have been improved. And that is despite having spent millions of dollars on testing tools and thousands of man hours testing and refining the code.

A software tester, be they part of the development of client team, has to be aware that just because a bug has been squashed once does not mean that it cannot reappear after changes have been made to another section of the program. This necessitates a formal testing methodology that lays out a sequence of actions to test. This testing sequence should then be performed again after a patch or update has been applied.

Here is an explanation offered to me that hopefully will make the subject better understood.

"I am not a programmer and know nothing about the subject. One day I needed to know how long some bug fixing would take. The answer I was given was "it takes as long as it takes. How long is a piece of string?" This irritated me, as I did not consider it was a real answer so I asked for a further explanation. What I was told was this.

When you write a line of code that makes a computer program, you are telling A to do BLAH. Unfortunately programming instructions are complicated and are often cross functional, so you accidentally are also telling C to do BING when you want C to do BONG. It is not until you try and enter the part of the program C that you realise telling A to do BLAH has gotten C to do BING so you fix that. Unbeknownst to the programmer in making that change he has now gotten F to do BRANG and B to do MUNG. Round and round it goes.

Basically, one line of code can undo the effects of several others or it can have everything do exactly what was asked for. It is only when someone goes into these other areas that it is realised what was changed. It isn't a matter of how trained the programmer is or how many times he has done this before (as each programming job is different). Therefore, it really does take as long as it takes.

I hope this has made it more understandable to non-programmers as it did for me." - Teal Els

Cost versus functionality versus deadline

There is an old maxim regarding software development, “There are three components – High Quality, Delivered On-Time and Cost to Budget. You can have any two.”

When it comes to software development most people have champagne tastes and a beer budget - they want more software functionality than they can afford or are prepared to pay. Over the last dozen years of software development I have determined conclusively that a developer can out produce most client's capacity to pay. This gives us our major challenge, "How do we give the client the functionality they want without sending either the client or us broke".

The attainment of that target is well nigh impossible on all but the smallest of custom software developments for small businesses. Most software developers therefore opt to produce a vertical market product that they can produce once and sell lots of copies to many people requiring the same functionality. This often means that you are forced to adopt the way the software does it as your business process rather than how you would prefer to operate.

We mostly program for small businesses for which an off-the-shelf program is an unacceptable solution. We have turned ourselves inside out trying to do the impossible (program everything the client wants at a price they want to pay) and have finally admitted defeat. While charging peanuts and delivering palaces at first sounds really attractive (from the client perspective) if it leads to the developer going broke we confess that despite our love of providing solutions that there is insufficient incentive in it for us to remain enthusiastic about the prospect and I suspect we are not alone. The end result if the developer goes broke is that it leaves the customer high and dry when they want support or additional functionality, so it ends up benefiting neither party.

I am reminded of the English clear thinking exam question,
"What happens when an irresistible force meets an immovable object?"

If you answer that the force moves or shatters the object then the response is that the object is immovable. If you answer that the object stops, absorbs or deflects the force then the response is that the force is irresistible. The phrasing of the question permits no logical answer.

In truth, there are no absolutes in the physical universe. So both statements are, in fact, a lie. The physical universe possesses neither an irresistible force nor an immovable object.

So, back to software development. One of the ways you can assist the developer is to clearly communicate your priorities so the developer is not trying to do the impossible. The three items from which you need to select a top priority before you start your job are:

Cost - Amount and payment terms
Scope\Functionality
Completion Time\Installation or "Go Live" Date

Scenario 1

If you choose cost as your number one priority the developer can program as much functionality as your budget allows. This necessitates that you set priorities on the items you wish programmed so that the most important and the most desired functionality are delivered first, within your cost restraints.

Scenario 2

If you choose functionality as your top priority then the developer can program the functionality you specify and bill you for the time taken.

Scenario 3

If you choose the "Go Live" date as your top priority the developer can deliver all the functionality possible to have operational by the "Go Live" date. In this scenario a cost or scope priority needs to be assigned as that will determine how many developers are assigned to the project.

Faster and bigger is more expensive

Just a note on that. The more people you have on a development team (whether it be to finish a job faster or because it is a larger job) the bigger will be the percentage of time spent on communication and collaboration so the higher will be the project management overhead and the greater the cost per function.

Bear this in mind if you are considering throwing more developers at a project to bring a completion date forward.

Nothing in the physical universe is perfect, software included.

There is no such thing as finished software, only software that has ceased being developed. There is always some other feature that will save time or money or add value to your business. There is nearly always a better way of doing something that only manifests after you have had a first crack at solving a problem.

"A piece of programming logic often needs to be rewritten three or four times before it can be considered an elegant, professional piece of work." from the book 'Classics of Software Engineering' by Yourdon. Therefore there will always be something you can add to the software or some improvement you can make. Full stop. Period.

Developers are not issued with crystal balls or extra sensory perceptions

The first time one can tell with 100% accuracy how long a software project will take is when the client tells the developer it is finished. Anything prior to then is an estimate. That's why a developer cannot tell you in advance how long a software project will take. They can tell you pretty much how long it will take to build what is specified and you can add an average amount for extras but that is still only an estimate.

Before the project is finished the developer can do all sorts of activities in an endeavour to give the client as good an estimate as possible. The developer can create complex models and elaborate costing calculations but these are all based on the data input (what is understand of what you want).

At the end of all these activities all the client really has is a best guess estimate, hopefully a three point range, with which to manage. Unfortunately that estimate is always different from the functionality the user wants when they start using the partially developed software. Apparently the average increase in functionality over the life of a project is something like 30%.

Where does that leave you? You need to keep in mind that you will always run out of money you want to spend on the software before you run out of bright ideas for extra functionality. At some stage you will call a halt or at least a pause in the development process while you still have items on your wish list that the software does not yet include.

Missing out on something you want never creates a happy moment. To make sure you suffer as little as possible when you come to that point you need to list all the functionality you want and get the most important functions built first.

As the project progresses much extra potential functionality will be exposed. Some of that will be more important than the functionality that was first listed in the specification. So in order to ensure the end product has the most important functionality included, you need to review the functionality that is available to be included at the beginning of every increment and prioritorise it. This makes the hard decisions as to what to include and what to exclude one heck of a lot easier.

Fixed price versus hourly rate

An article I read once said that it is unwise to pay too much but it can be a total waste of money to pay too little because if you pay too little you may not get any value from the money you do pay because the product does not do the job.

There are a variety of ways to arrange payment for custom computer programming, fixed price and hourly rate being the two most common. There are definite schools of thought on these and I would like to share my experience and viewpoint with you.

In the mythical perfect world we pervade the mind of the client, instantly duplicate their entire business experience, think briefly about what would be the perfect software solution for their business processes, draw on our infinite knowledge of all things past, present and future then instantly create the perfect software in one draft incorporating all the functionality, power and flexibility that our users need now and will want in the future.

OK, now back to the real world. Obviously it takes time to extract all the data from you, the future user of the software, it takes time to design the database and application, it takes time to figure out how to do some things, it takes time to design how a program will work, it takes time to create the forms, reports, code and menu construction, it takes time to test that what was created performs as designed, it takes time to fix the bugs, it takes time to research and test different scenarios to see if they will work faster or slower, it takes time to make changes to suit your specific aesthetic or operational preferences.

Just because an idea does not turn out feasible, practical or doable does not negate the time we have spent investigating it. What you do not see, and for which you do not pay is the many hours we put in each week learning new techniques, corresponding with fellow developers, expanding our competence so we can be even more productive for our clients.

There are a number of reasons we program on an hourly rate. To be perfectly honest the most major one and the simple truth is that I don't like being "out-exchange". You see I love delivering functionality that you need and giving you insights into your business, not arguing over whether or not it was or wasn't in the specifications. When you ask, "Can we have it do blah?" I like the hassle free operating basis of just saying, "Sure!" and giving it to you and knowing that when we send you an invoice you will drop some money in our account or send us a cheque.

Probably my worst capability is accurately predicting how long it will take to perform a task I have not done before. I don't know if it's happened to you but sometimes I have looked at something I haven't done before and predicted it to be a lot simpler and easier to do than it turned out to be. As one gets further into a project often unforseen factors appear which means that it takes more time, effort or money to achieve the desired result. Maybe this is merely an example of why there is a saying, 'The grass is always greener on the other side of the hill.'

I sometimes joke to a client, "I need to know your business better than you do to automate it!" But it's no joke, I do. You can apply all those wonderful human qualities like intelligence, judgement and discretion to the decisions you make on

running your business. A computer can't. It is a complete and utter moron. It does specifically, only and exactly what the programmer tells it to do. Get one comma out of place and it's all over red rover.

Conflict or partnership

Many in this society seem to be focussed on a win\lose operating basis. For many in business it is a case of trying to get as much as possible for as little as possible from suppliers or customers, often without regard for the survival of the other. Few have the attitude of win/win. I know I am unusual but when I write a programme for a client I mentally assume a position of partnership with my client. I find out what is needed and wanted by the client then write the programme that does that in a way I would like to work with if I were the person running the business. As I am designing and writing the programme I am continually looking for improvements to the way of doing things and how I can eliminate, streamline, automate and make more efficient the mundane, time consuming or repetitive jobs.

Also, I am continually on the lookout for legitimate shortcuts which will reduce the time taken for the client to perform a task. Many factors make it well nigh impossible to accurately estimate the amount of time it will take me to complete a given programme in advance of me starting.

Fixed price jobs

For the purpose of limiting one's expenditure one can desire to put a cap or ceiling on the expenditure and ask for a fixed price quote. Fixed price quotes are predicated in part on the principal that sometimes you win, sometimes you lose. They are (in my opinion) relevant in situations where all (or nearly all) factors are known in advance and the anticipated time to perform the work can be fairly accurately estimated. A loading for the small number of unknowns or unplanned for items can then be applied and the quoter can sleep easily knowing that sometimes he will win, sometimes he will lose but overall things will pan out.

I have a few problems with this. I am not prepared to over quote and overcharge one client to recoup from him what I didn't get from another client. I am also not prepared to work for nothing to satisfy the requirements for a known cost.

Most goods and many services are exchanged on a fixed price system where the both the quality of the product and the cost of production are known or definable in advance. This has the advantage for the client of a known cost for a specific product. With a computer programme the problem is that despite all the care in the specification of the job it is very true that the devil is in the detail. The spec is never perfect, can be subject to different interpretations and 98% of the time the totality of the job and the time taken to complete it is not known until it is completed.

It is nigh on impossible to predict at the outset all the functionality that will be required, therefore things that are "taken for granted" by the client and were not even considered by the developer can be contentious and lead to an adversarial situation.

I well recall one particular client for whom I did a small custom invoicing program. I thought I had finished and took it in to install and train him on it. We spent the next 4 hours modifying the invoice report until he was happy with it. It turned a $1,200 job into a $1,800 one. I look at that situation and on the one hand the client bitched about the price (and to this day calls me a gold digger) but on the other hand he got the product he wanted. If it were a fixed price quote it would have potentially been a bun fight because there was an undiscussed requirement that the presentation of the customer docket conform to his standard of acceptability. I didn't think to ask, he didn't think to mention it and if it had been a fixed price job I would have lost out big time.

Over the years I have lost track of the number of times the following has happened. A person commissions software development on a fixed price quote basis from another programmer. The programmer had taken longer to do the job than he had bargained for, been paid progressively and rather than spend the time necessary to complete it he had moved on to paid work leaving the client with a 90% done job. The person then called me to come in and finish off what the programmer had not completed.

Once such still sticks in my mind, I received a call from a potential client asking if I would come and complete a project for them to which they could not get the developer to come back. He volunteered, "We've paid him what we agreed on and I know he's probably put in double the time he thought it would take but there's a few things that still don't work right." The programmer had left the job undone with incomplete features and unresolved bugs. Needless to say both the client and the programmer were unhappy. The client didn't get what he wanted and the programmer didn't get the satisfaction either of completing the job or keeping the client.

I made the decision at the time to not undertake fixed price jobs. On the rare occasions I have broken that policy and done fixed price jobs I have ended up delivering more hours than I budgeted for the job. I have come to realize that this is almost inevitable for two reasons.

Number one is that I would not personally feel comfortable getting paid for more hours than I contributed. This of course totally undermines the basic principal of fixed price quoting as it means I would never enjoy any 'wins' to compensate for the 'losses'.

Number two is that it is my very strong desire to deliver the product that does what the client wants, and as the project progresses and the client better understands the detail he naturally thinks of more and more little details that I can justify doing as they 'only take a minute' to add but which all add up.

As mentioned earlier, the industry standard is that a very small percentage of programming projects finish on time and under budget. Them's pretty bad odds that I will ever come out a winner on a fixed price job. With a fixed price quote the possible result is one of three alternatives:

1. The programmer and the client agree perfectly on the job to be done, the programmer correctly estimates the time it will take to do the job and he finishes

the program as anticipated by producing exactly what was needed and wanted in the hours expected.

2. If it takes the programmer more time than estimated to do the job then the client is getting more than he paid for or an unethical programmer will not put in some of the functionality originally specified.

3. The programmer builds into his price a safety margin or over estimates the time to do the job, finishes early and wins at the client's expense. If he completes the job in less time than the safety margin allows for, then the cost of the safety margin is how much too much the client has paid for the job.

Both scenario 2 and 3 are to me out-exchange. In both circumstances one side is getting something for nothing. I am a strong believer in the principle of delivering exchange in abundance. I like to deliver to a client more and better than what was expected. On the times I have worked on a fixed price job and it took less hours to complete than anticipated I've spend the unused hours doing additional work on the programme to enhance it.

Likewise if it took longer to complete than I was being paid for I have completed the job despite it being uneconomical to do so. So as a result of my attitude, I am either not winning or I am losing.

I have no desire to win at my client's expense, I believe in doing a better than expected job where ever possible but I can't work for nothing as that wouldn't be doing the job my family expect of my in providing for their needs.

On an hourly rate the client is in the driver's seat. The client can always say, "Yes! Brilliant suggestion, add it in." or "No, we would get minimal additional benefit from that functionality, leave it out." Also some responsibility is being borne by the client to make sure that they are requesting only those things that genuinely make a difference, after all, they are paying for it.

It reminds me of what my late father used to ask clients who asked him if they should spend less money on the slightly torn stamp or more on the better one. My father would respond, "The quality is remembered long after the price is forgotten."

This saying is probably never more relevant than in software as the cost to develop good software is borne only once at the beginning whereas the time taken to operate a kludgy program is a cost borne for as long as you operate it.

Expanding job specifications

As I work with a client designing an application for them I educate the client as to what is possible and the options which are available. As a result of this the client's perception of what is possible expands and so does the list of features, reports and automated functions which are considered desirable. As mentioned above I also think of worthwhile additions to the original specs.

With a fixed price quote the alternatives are:

1. having to say 'No.' to requests for additional functions as they were not in the original design specifications for a fixed price quote, (I HATE having to say 'No.' to a client.)

2. spend administrative time in negotiating and writing contract variations, (I want to minimise the amount of time I spend on paperwork and administration as it falls into the category of work items that do not directly deliver working software.)

3. do the extra programming for no charge. (I am not in a position where I can work for nothing.)

Not one of the above appeals to me.

If a client wants a valid and worthwhile addition to the programme not in the original specs I want to be able to include what they need and want in the application without the hassle and non-productive time spent negotiating contract variations.

This complies with our purpose, which is to deliver tools that will enhance and improve the productivity, accuracy and job satisfaction of the client and staff.

I also want to be able to suggest and, if agreed to, incorporate good ideas to enhance a programme despite their not being in the original specifications.

Hourly rate

In order to avoid the above non-optimum situations Just For You Software works on hourly rates.

From a personal perspective, when I am working with a client I mentally go into partnership with them rather than adopting an adversarial role. I would much rather be wondering, "How can I make this better for the client?" or "How can I deliver the most value for the least money?" rather than, "How can I satisfy the design specifications with the least amount of work?" or "What can I leave out that the client hasn't asked for so I can get the job done in the least amount of time?"

All the above has led me to the preference to operate on an hourly rate rather than a fixed price quote and give the client the option of what they say 'yes' to in response to my suggestions. It all boils down to, "You get what you pay for." or "There's no such thing as a free lunch. Somewhere, somehow, someone pays for it."

If a client has a definite budget and wants to stick with that as the highest priority so that their costs are limited then it’s only fair we do the same and limit our costs by providing a known number of hours for that amount of money.

We will insist that we establish a strictly prioritorised list of desired features. We will then work on delivering the features from the top of the list down until the hours consumed equals the budgetary constraints. This way the client gets the most important functionality before they run out of money and both of us are in-exchange with each other.

In conclusion

As you can see from the above, there are far too many ways for a software development project to go off the rails. A great many of them are totally under the control of the client. No developer can realistically warrant that they are perfect or will get it right first time, every time. From my experience, most try to do their best and if you work with them and take half the responsibility they take to ensure the success of your project then it will be a success. Not without some bumps in the road but a success nevertheless.

If you have any suggestions as to how your developer could improve their level of service to you or improve the quality of their products or documentation, bring that to their attention. Most will appreciate it.

If you have a suggestion as to how to improve this article please drop me an email.

Congratulations for completing this document and here's to your successful project.

Warmest regards,

Tom Grimshaw
Managing Director

31st July 2006

Bibliography

Additional data can be gleaned from the Standish Group 1994 CHAOS Report:
http://www.standishgroup.com/sample_research/chaos_1994_1.php
and The Standish Group report "Unfinished Voyages" can be read at:
http://www.standishgroup.com/sample_research/unfinished_voyages_1.php

The Agile Manifesto and related articles can be viewed at:
http://agilemanifesto.org/

Recommended reading:
'Agile Software Development' by Alistair Cockburn
'Classics of Software Engineering' by Yourdon
'Lean Software Development' by Mary and Tom Poppendieck
'Managing the Development of Large Software Systems' by Winston Royce
'Waltzing with Bears Managing Risk on Software Projects' by Tom DeMarco and Timothy Lister