An Introduction to

Business Software Development

Principles and Practice

First Edition. Published January 2016.

Book details, supporting files, and other book titles can be found at
www.antonypennbooks.com

Preface

This book is a free sample of the main book, entitled Business Software Development: Principles and Practice.

Visit www.antonypennbooks.com for details on how to obtain the book, or visit any Amazon store.

About this book

Developing software is an expensive, complex business.

This book is intended to advise everyone in the development of software for businesses, with a mix of principles and practices based on traditional wisdom and years of experience.

Although aimed at commercial businesses working on typically small to medium sized projects, either for internal or external client use, many of the principles apply to software development in general.

Unusually for a book such as this, it is not filled with every 'best practice' that you should follow in an ideal world. Instead, it takes a pragmatic approach to what is typically wrong and the essential things you really should follow, in order to get the job done. Time, finance and resource constraints are always against bespoke software development projects. This book offers practical advice to help you make the most of what you have and achieve what you need to.

A lot of the information contained may be considered common sense, but it is important to re-emphasise that knowledge to ourselves from time to time, to ensure we consider it when making decisions. The structure of this book makes it easy to just pick up occasionally and read a few random pages.

This book is organised into sections aimed at the various people involved in the development cycle. However, most of those people would benefit from reading the entire book.

A short software development story

The Big Boss at 'The Company' wants a new version of their 'Product C', to be released at the Event Show in 16 weeks time. There are no specific details yet, so a meeting is arranged between project management and marketing staff to discuss what features should be included.

The development manager is sent the draft specification of the 'version 2' requirements. In those requirements, he notes that the UI design will be coming from an external agency. He asks his main developer if the time-scale can be met, using 2 developers. He needs the answer for a meeting in 2 hours with the Big Boss. The developer says that, at a push, he can get 75% done. The development manager has a meeting with his staff and concludes that, if a third developer joins them from project B when he is finished in 8 weeks time, they can get 90% done, with the remaining 10% moved to a version 2.1 release.

A project kick-off meeting is arranged between the 3 developers, the development manager, the project manager, sales and marketing director and the marketing manager, plus the customer support manager. Due to conflicting schedules, the meeting does not take place for another week.

A project plan is drawn up, together with some milestones. The project is already starting a week later than the developer anticipated, so a few days have been trimmed off the testing phase to make the deadline fit the event date.

The developers start working on a more detailed specification whilst also designing the general structure of the system. They have a few questions that need answering, so the project manager tries to organise a meeting with the sales and marketing staff, but this takes another week to

come to fruition. In the meantime, the developers work from their assumptions and start coding.

The UI design does not arrive when expected, so the development manager asks the project manager to talk to his contact at the design company to get an ETA on the designs. A week later they arrive, but there are some issues, so the project manager organises a conference call between himself, the design agency, the development manager and the 3 programmers. This call takes 2 days to occur, but eventually the issues are sorted out and a revised UI design is sent within a few days.

One month into the project is the first milestone date. A meeting is called for everyone involved. It turns out that the project is about a week late. Some minor features are pushed out to release 2.1 in order to give the project time to catch up. A catch-up meeting is planned for 2 weeks time, but again due to conflicting diaries, it is eventually scheduled for 3 weeks time.

At the next review meeting (now 7 weeks into the project), milestone 1 has been met, but the project is still 1 week behind. It is believed that they will catch up, especially as the third developer is due to come on board from Project B next week. Another meeting is scheduled for 3 weeks time, by which time milestone 2 should be met.

At this meeting (10 weeks in) it is discovered that the project is 10 days behind. This is attributed to initial problems with the UI design and the fact that the 3rd developer did not join from Project B when expected, due to problems and delays in that project, plus the second developer keeps getting pulled back to fix bugs with Project A. It is believed the team can catch up, so another progress meeting is planned for all involved in 1 weeks time, where the development team hope to get to milestone 2.

Eleven weeks in and the development team have pulled the

deficit back to 4 days.

The progress meeting held at week 13 discovers that the project is now 8 days behind. The development team did make progress but one of them was sick for a few days. As the initial release date was 4 days before the Event Show, they decide that they can still be ready if they squeeze another day out of the testing and work up until the night before the show, and pull a few late nights. The development manager allows them to order pizza for late nights and gives them the keys to lock up on those nights that developers stay late.

Seven days before the show, and the development team finally ask the project manager, sales and marketing staff, plus some of the support staff to test the product. The original testing phase was 3 weeks plus another week for bug fixes. Staff do not get around to testing the product for 2 days, leaving only 5 days for testing and none for bug fixes.

A few bugs are found and the product is deemed unsuitable for release at the show. The marketing team take it to the show as a 'beta' product and offer 10% off for any pre-orders. A new plan is drawn up by the project manager and the sales and marketing director for the official version 2 release date. The whole cycle starts again.

In the above story, just about every single decision and choice was poor or incorrect, yet it is very likely you have shared many of these same steps and procedures in your own projects. Let’s look at why there are wrong and point out some principles that could and should have been used.

Release something by this date

All too often businesses have a date in mind for a project release, yet do not ask the people involved in building that project, how much time they need. Time-scales need to come from developers.

Vague requirements

Many specifications given are very vague, leading to assumptions being made by development staff. Ultimately this will cost more time to correct later, than if it was specified properly up front.

Developers writing specifications

Many specifications arrive at the development department very short on requisite detail. Development or analyst staff should not fill in the gaps themselves, as this leads to assumptions (which results in a broken system taking longer to fix). Instead, work with the client to help them provide the necessary details.

How long will it take?

Developers are often asked to give estimates instantly. This is crazy. You cannot provide a usable estimate without having the time needed to think about it.

Too many stakeholders

Too many people in meetings, are a waste of many people's time, plus cause unnecessary delays whilst trying to find a time suitable for all.

Too many meetings

You need to balance getting the work actually done, against too many meetings talking about it.

The wrong people at meetings

Having the wrong people attend meetings wastes a lot of time. For example, in the story above, there was a conference call between the project manager, external agency, development manager and three programmers,

whereas only one programmer and the external agency were required.

Outsource with care

Outsourcing can cause many problems. The first of which is time: you cannot control when they will actually deliver. The second is quality of work: it is much harder to verify quality from afar, especially if work is not delivered continuously.

Revise estimates

After 1 month, this project was 1 week late. You need to look for the reason why and re-calculate the delivery date. Chances are, it will be 4 weeks late after 4 months.

Allow unplanned time

In any project plan, add some time for 'unknown' tasks. Do not put anything in this space. Keep it just for soaking up pressure when deadlines slip. They always do.

Avoid end-to-end planning

If a project or major task finishes on Wednesday, do not plan the next one to start on Thursday. Leave a gap between them to handle overspill. Otherwise, as soon as one thing is late, you will always be late.

Avoid context switching

Programming is a complex task, involving a lot of (human) memory. If a programmer has to switch between project A and project B, he must also 'dump' one project from his mind and 'recall' the second. This is known as a context switch, and costs an awful lot of time. Restrict developers to working on one project at a time as much as possible.

The more developers, the slower

The more developers are working together on a project, the 'slower' (in terms of 'man months' of effort) progress will be, due to increased communication between them.

Plan to lose time

No project goes 100% according to plan, so expect that developers will come across some problems that stump them for a day, will be off sick, or get called to work on an urgent bug on another system. Plan to lose some time.

Stop trimming testing time

Typically, on an ever tightening deadline, the testing phase is the one to get squeezed. If this happens a lot where you are, try padding it more in the first place.

Support your developers

It is no good 'letting' them order pizza – buy it for them. They are committing a lot of their own time to the project, you should at least treat them to food. Also, do not abandon them. It is not their fault that the project is late, so when they stay late to help out, you should be there with them, showing your support.

Bug fix and re-testing

I've seen many project plans with an amount of time for testing allocated at the end, and yet none for fixing the bugs found and re-testing the result. Things rarely work 100% first time.

Users hate testing

Testing is not a fun job for most, and users are often reticent to commit time to this important task. Get them to test early

and often.

The following chapters dig deeper into these and other principles, which should help everyone involved to create better software products more effectively.

The paid version of the book goes on to detail 89 principles divided in to five target audiences: clients, developers, testers, project management and staff management, followed by an acronym dictionary and glossary of terms typically found in the field of software development.

What follows is a small sample of those principles.

1.1 Be prepared to work for it

A successful project requires a **lot** of input from the client. Many clients naively produce a basic (and usually incomplete) specification and expect to receive a fully working system in return. Many clients (especially those that are external to the development company) are not experienced in producing such complex, detailed documents. They need to work closely with the business analysts/developers in the first instance to make sure that the specification is complete and then throughout the project to assist with any questions or missing details.

Furthermore, the client should be engaging in testing prototypes, new modules or iterative features as often as possible. In short, the client needs to devote substantial time to the project during its lifetime in order to improve its chances of success.

2.20 Build throwaway prototypes

Sometimes there is a particular problem that is difficult to solve or the requirements are vague. In such cases you can build a prototype to demonstrate and test the theory of the design that has been provided or created. In these cases it should be developed as quickly as possible, using any language and tools that you fancy (i.e. that allow you to complete it the quickest). This is where 'quick and dirty' wins. The objective is purely to test that your hypothesis, the design, algorithm, UI or whatever it is, does in fact work as expected. You can then throw the prototype away and code it properly.

3.6 White box testing

White box testing, by contrast, involves exploiting knowledge of a module's internal code structure. For example, if you know that a particular parameter should be an integer with a range from 1 to 10 inclusive, test all the edge cases for that variable (0, 1, 2, 9, 10, 11) as well as some out of bounds values (-1, 0.9, ABC, 23467823, etc.)

It might also be, for example, that certain inputs within a certain range, cause additional code to be executed. Without internal knowledge like this, you may miss testing parts of the functionality.

A code coverage report (see 3.9) will also (and more accurately) identify whether or not your tests have covered such conditional code.

4.7 Use pair programming for complex logic

Pair programming is the practice of having two developers sit at one computer and write code. Normally there is one developer in the 'hot seat' and another observing. They may take it in turns to do the actual coding.

Pair programming is clearly expensive, but the cost is marginalised when used sparingly in certain circumstances. The first is when developing a complex piece of logic. The second set of eyes helps catch bugs as each line of code is written, plus both of them will interact and come up with more solutions and ideas when tackling complex obstacles. In these cases, the cost of pair programming is offset against the time spent debugging the resulting code.

Another use for pair programming is knowledge transfer. It is an ideal way to educate less experienced programmers, and is also ideal for new employees to help get up to speed on your code base and form friendships with their new colleagues.

5.7 Hire intelligence, not skill set

Many managers, when hiring programming staff, generally present a 'shopping list' of skills that they want prospective staff to have. However, this is the wrong approach.

The programming landscape is in constant flux. New languages, libraries, frameworks and even methodologies are popping up almost weekly. But these things are just 'tools' used by programmers, much the same as tools are used by a carpenter. A carpenter needs to learn his craft first, and then he can happily switch to using whatever tools become available in due course. No one tool will make him a better carpenter.

And so it is with programmers. If a programmer is intelligent, they will learn their craft, and will be able to adapt to new languages and technologies as they arise. But that fundamental skill of programming, being able to break down problems in to mathematical and logical blocks that a computer can interpret, transcends the languages, libraries, and tools available. Without that skill, any programmer is limited.

So the next time you are looking to hire developers, do not rely on a ‘shopping list’ of skills you would like, together with an arbitrary ‘number of years experience’ you feel appropriate for each. Look principally for intelligence, programming, and problem solving aptitude, ability to communicate clearly, and the ability to describe and discuss technical ideas in layman’s terms.

This book is a free sample of the main book, entitled Business Software Development: Principles and Practice.

Visit www.antonypennbooks.com for details on how to obtain the book, or visit any Amazon store.