The Tactics of Software Development

Dr Graham Stone, Dunstan Thomas Consulting
http://consulting.dthomas.co.uk

Introduction

In a recent article, I compared and contrasted the job of a Premier League Football Manager with that of a Software Project Manager. An odd comparison? Maybe, but my point was that some football managers appear to get better results from the same team than the recently sacked manager. Why? The team's the same; the opposition are the same. I concluded that the best managers got the most from their team but were also willing and able to change not only the team but their tactics, sometimes in the middle of a game. There's probably also an element of complacency with the old manager that doesn't exist with the new, that is, because he is new, he gets listened to more. I also concluded that as software project managers, we perhaps cling to the belief that the methodology we prefer can solve all problems provided that the team, the customer and the project allow it.

I'd now like to explore this attitude further. Every football manager has a pretty good idea of the strengths and weaknesses of the opposition they are about to play (because they've studied the videos, etc.). They are therefore able to make certain assumptions of the best approach to deal with the oppositions' strengths and weaknesses. Alternatively, they may believe that there is nothing about the opposition they need to fear and choose to play their own game regardless of whom they are playing next. In the world of software development, the "opposition" is far less clear. We may be fortunate enough to be developing an application very similar to one already completed. Alternatively, we may be developing a second or subsequent application for the same client. We may also be fortunate enough to be working with the same software team. However, odds are that none of these will be true.

Let's assume the worst case scenario: New client; new project (i.e. unfamiliar business problem to solve); new technology; new team. For some, this will be the absolute norm – it certainly is for 75% of my projects. So where does the project manager start? For me, it is the development methodology because it tells me how to organise the whole project. My development methodology of choice is the Rational Unified Process (RUP). It tells me how to assign roles and responsibilities to the team, how to select the deliverable (admittedly with the help of hindsight from earlier projects), and how to plan the phases, iterations and objectives of each part of the project. Since I have completed many projects using this approach, I start with a considerable amount of pre-written documentation that only needs to be amended to reflect the demands of the new project. If the project started straight away, relying on the thoroughness of the development methodology and experiences gained using it on other projects, I believe we have adopted the stance of the football manager who refuses to have his plans or tactics affected by the opposition – we're going to play our own game.

The Standish Group (www.standishgroup.com) publish annually a report aptly called Chaos which presents statistics on how and why software projects fail. Table 1 is taken from the 1995 Chaos report (although I still believe it to be broadly current) and gives an insight into how projects tend to fail. *In my view, the contents of this table should form the essential check-list for any Risk List document created by the project manager.*

Project Challenged Factors	% of Responses
1. Lack of User Input	12.8%
2. Incomplete Requirements & Specifications	12.3%
3. Changing Requirements & Specifications	11.8%
4. Lack of Executive Support	7.5%
5. Technology Incompetence	7.0%
6. Lack of Resources	6.4%
7. Unrealistic Expectations	5.9%
8. Unclear Objectives	5.3%
9. Unrealistic Time Frames	4.3%
10. New Technology	3.7%
Other	23.0%

Table 1

If a development methodology is to help us succeed, it needs to be able to address all the items in Table 1. However, if you look behind the stats, what you will notice is that pretty-well all the factors above are human or personal issues (with the possible exceptions of 5 and 10). That is, we need to be able to adapt our game plan to take into account the fact that software development is in fact a game whose outcome is primarily determined not by software but by the characters involved in the project – namely, the customer and the development team (in Extreme Programming, the customer is always regarded as part of the team).

For the sake of argument, let's regard each of the ten points in Table 1 as being the equivalent to the opponents of our software team. We have RUP (again for argument's sake) as our strategy for dealing with them but how can we change our tactics to address them and what parts of our methodology will we need to be altered in order to minimise the changes of these issues causing our project to fail? Also, will a single development methodology address all issues or do we need to consider parts of other methodologies as well?

The following sections address each issue in Table 1 in turn.

Lack of User Input

Many companies won't let the development team talk directly to the users. With a web application, it is arguable that this is logically impossible (alternatively, we have to consider ourselves as users). Some companies choose to place Business Analysts in front of the users so that end user requirements come from them instead, that is, the Business Analysts act as surrogate users. In the best situation, the team are able to work freely with users as they wish. Therefore we have:

1. Totally free access to users
2. Access to users via Business Analysts
3. Hypothetical users

Unmodified RUP will cope with option 1 without any modification, indeed it assumes it. But what About options 2 and 3 and why are they likely to cause a problem to the project team?

RUP provides the following questions to help the team find out what they are doing and why:

- Who will use this system?
- What other systems will this system send information to?
- From what other systems will we receive information?
- Who starts the system?
- Who will maintain the user information?

Naturally, if we are not talking directly to the real end users, some of the answers to these questions are going to be based on assumptions. Therefore, extra effort must be made to prove that our assumptions are correct.

Thus the tactics necessary to defeat the "**Lack of user input**" opposition is:

Take as much time as it takes to prove beyond any reasonable doubt that the information supplied by our surrogate end users accurately reflects the real end users needs – even if this means delaying the project.

Incomplete Requirements & Specifications

There are probably at least 3 basic types of software project:

1. Maintenance or enhancements to an existing application (or business problem)
2. A new application for a well understood business problem
3. A new application for a new business problem

In 1, because we have "been there; done that", our requirements should be as clear as they could ever be; let's call this 95% clarity. On the other end of the scale, we have option 3. It could easily be argued that here we can expect no better than 50% clarity on requirement which drops to about 40% once someone tries to write it down. The problem is that in recent years we have been led to believe that iterative development will help us cope with this. In RUP, for example, we are encouraged to begin an iteration with an incomplete understanding and to gain more clarity as the requirements discipline is reached. In a recently complete application for a brand new web-based business, we experienced over 150 change requests during the construction phase. Whilst the technical architecture coped well with this, the budget had to be extended to cope with the volume of re-work.

Thus the tactics necessary to defeat the "**Incomplete requirements & specifications**" opposition is:

Don't expect the requirements gathering and specification activities to be the most efficient way to define a business problem. Spend more time modelling the business needs and only when this is stable should requirements be defined and specified. In RUP terms, this would equate to more emphasis on the Business Modelling discipline.

Changing Requirements & Specifications

I have always suspected that the volume of change request (be they defects or enhancements) is a barometer of other more significant problems associated with the project. In the ideal world, customer

know exactly what they want and are able to articulate it perfectly to a project team member who is capable of documenting it perfectly. Neither of these *ever* happens for all but the most trivial project, of which we are not concerned here. Let's have a look at a few examples and try to guess what might be the root cause of the change request:

Type of Change Request	Possible Root Causes
Defects	• Inadequate testing (at all/any level) • Assumptions or ambiguity in requirements specifications • Inadequate definition of non-functional or generic • requirements • Lack of realistic test data
Functional Change Requests	• Lack of initial understanding of the business problem being addressed • Changes to Business problem being addresses • Change in budget or schedule • Inadequate review of requirements specifications • Assumptions or ambiguity in requirements specifications
Non-functional Change Requests	• Inability to predict volume/performance needs at the start of the project • Lack of "big picture" view of whole application • Poor understanding of integration with other systems

Table 2

Thus the tactics necessary to defeat the "**Changing requirements & specifications**" opposition is:

Don't consider the change requests themselves as the problem: Look for the root causes and revise/enhance the part(s) of the methodology that specifically addresses that issue – in the majority of cases this will be something to do with the detail or clarity of the requirements specifications themselves.

Lack of Executive Support

In one project in which I was recently involved, the project was cancelled half-way through the Construction phase. The reason? The Marketing Director had proved to be far too busy to review or participant in any initial project briefings or to review the Vision document that stated the purpose and business goals of the application. Despite the fact that the project was running like a dream and that the project was an enhancement of an exiting application, it turned out that it would be impossible to sell to existing clients as they would expect it to be given away free of charge as an upgrade. In another project, the customers' MD habitually re-directed all project related email message to his Deleted Items folder and failed to pick up on messages warning of a potential budget and schedule over-run until it was too late to do anything about it. In another, political problems amongst the Stakeholders resulted in important requirements being withheld until prototype reviews were conducted. In another, a personal vendetta between the project manager and the IT Director forced the project manager to publish a project schedule and budgets before the project started (NB it is often said that project estimates fall into 2 camps: Lucky or Lousy! Unfortunately, his was lousy).

So, what can we say about the tactics necessary to defeat the "**Lack of Executive Support**"?:

Give up! You don't stand a prayer!!

Technology Incompetence

It is the purpose and goal of every developer to fill their CV with as many languages, techniques, methodologies and TLA's as they can! Ask any developer what the most appropriate technology to solve a particular business problem and I will bet you my last Rolo that it happens to be the latest software on the market! Try this: Whenever you hear a developer use the word "Should", replace it with the words "Is unlikely to…" and see how that changes things. For example:

Project Manager: "What's the most appropriate technology for this new project?"
Developer: "Well, looking at the spec, I'd say .Net"
PM: "Have you ever used it?"
Developer: "No, but it shouldn't be that difficult to pick up"
PM: "How long will that take?"
Developer: "Shouldn't take too long at all"
PM: "Will it save us time in the long run?"
Developer: "Yeh, should do…"

Productivity will suffer if the tools being used are not mastered by those using them. So, what's the tactics necessary to defeat the "**Technology Incompetence**" issue?:
Avoid using new technology on a projects that have any sort of constraint on time or budget – regardless of the methodology being employed.

Lack of Resources

It ought to be considered common sense that starting a project with insufficient skills or personnel is a recipe for disaster, particularly if the project has schedule constraints. Similarly, swapping or removing team member could also be expected to reduce productivity. The only way I know to minimise the "**Lack of Resources**" issue is:

Strive to have as many multi-skilled team members as possible – regardless of the methodology being employed. At very least, team members should be capable of eliciting and documenting requirements as well as writing code and unit testing.

Unrealistic Expectations

To make sense of this item, I believe we need to break it down a bit. Software development projects involve:

1. Expectations that the software will solve a specific business problems
2. Expectations that the software will be completed within a given (often fixed) budget
3. Expectations that the software will be completed by a certain date

If the requirements have be accurately identified and documented, expectation (1) ought to be routinely achieved, eventually. However, we often under-achieve in terms of (2) and (3). Why? And why so often? The Standish Group tell us that less than 30% of all projects finish on time and on budget. That's appalling by any standard. But is this due to unrealistic expectation on behalf of the customers or do the development team set the expectation unrealistically? In most cases, I think the latter is true. After all, the usual question is "We've got this software project to start. Can you tell me how much it

will cost and when it will be finished?"

My personal belief is that most projects fail to finish on time or on budget because the estimates of both tend to made way too early in the project. In other words, I believe that most projects are doomed to failure from the start because someone put the estimate together before a complete understanding of the project size, complexity or even the requirements were understood. I have personally been asked on more than one occasion to provide an estimate of budget and completion date for a project *before* the requirements have been defined – in fact, before anything but the project name had been decided!

There is an increasing mountain of software estimation articles, books, techniques and software on the market. However, most techniques and certainly most tools rely on either the number of lines of code as an input parameter or function point analysis (a crude measurement of gross size and complexity). However, even if you spend 10's of 1000's of pounds on estimating software, they still only give +/- 40% estimates at the ***end*** of the requirements gathering activity (and that is assuming a thorough Waterfall lifecycle). The "cart before the horse" problem is that in order to come up with these input values, it is often necessary to have this information at the point when either a) the project has not been started at all or b) when no historical information is available as an alternative.

To a customer, estimates are needed before the start of the project. These estimate then need to be confirmed and updated as the project progresses. From the developers' perspective, we often have little idea of how long something will take to do until we have almost done it.

As far as I am aware, no popular development methodology (including Waterfall, RUP, XP, FDD, DSDM, etc.) is able to address this estimation issue. Sure, an iterative development life-cycle gives one ample opportunity to revise the estimate but that doesn't help to address the real issue – namely unrealistic expectations. The best that can be hoped for is that as the project progresses, we build up a picture of how long *this project team / customer combination* takes to define and create software. If I was to select one methodology that goes further to address this issue it is probably DSDM (dynamic system development method). It uses the concept of time boxes into which we cram as many features as we believe we can develop in a fixed period of time. At the end of this time box, low priority tasks are either dropped from the project or added into the next time box. At least this gives us a measure of real productivity. The problem with DSDM is that is does less to address overall project risks concentrating as it does on short-term objectives.

Thus, best advice for handling the "**Unrealistic Expectation**" issue is:

Always under-promise and over-deliver. Don't commit to detailed estimates of time or budget until tangible evidence to support them is available – in other words, the wet finger in the air just won't do. Methodologies that stress short term objectives like XP or DSDM can provide this evidence – but don't forget the big picture either. Perhaps conduct an extended RUP Inception phase or an XP Spike as a proof of concept or a short DSDM Time box containing a more difficult feature. In any case, initial estimates should always reflect the lack of precision of the estimate i.e. first quarter 2004 or between £50K-£75K (the upper limit should always be a comfortable figure that the team are confident of not exceeding). Don't be bullied into refining this – it will be used against you later! As the project progresses, refine the estimate accordingly. I would not advice the use of MS Project to derive estimates as it often gives a false impression of accuracy when the reality is that a Gantt Chart is typically 1001 guesses added together.

Unclear Objectives

When I first came across RUP, the importance of the Vision document was stressed time and again. I know appreciate why. Without going into detail, its purpose is to crystallize the business objectives of the project and how the software will help to achieve them. It doesn't contain any "compu-jargon" at all. It will contain a list of features but no design or implementation details.

I mentioned earlier a project that I was involved with whose Marketing Director realised too late what the project team were doing – he should have read the 4 page Vision and all would have been clear. *I cannot stress enough how important it is to start a project with a crystal clear understanding what the aim of the project is.*

My feeling (or more accurately, my prejudice) is that the more "engineering"-type methodologies like Waterfall and RUP, being more formal, are better at addressing this issue than methodologies like XP and DSDM, being more implementation-centric. XP in particular is more popular with programmers (I deliberately didn't write developers) than it is with project managers. I strongly suspect that this is partly because planning is something not naturally done by programmers but is a way of life for project managers. As such, objective setting is arguably a bigger part of the more formal methodologies and hence more suitable for mitigating this type of risk.

So, best advice for handling the "**Unclear Objectives**" issue is:

Write them down and make everyone involved with the project, Stakeholder in particular, memorise them. This is a natural part of the more formal methodologies but could easily be incorporated in all project.

Unrealistic Time Frames and New Technology

I am happy that I have covered these in sections above. I've added them again here in order to tell you 2 more stories: The time you will find most unrealistic time frames is likely to be once the change requests start rolling in. It is unlikely that many customers will willingly allow you to extend the project deadline just because they given you another 4 months-worth of change requests to complete! The wise project manager gets into the habit early in the project of telling customers how long and how much every change request will cost. At the same time, an extension to the time limit should be asked for i.e. if the change request is estimated at 2 days, ask for 2 more days to be added to the deadline – regardless of how far away it is.

Regarding new technology, prepare yourself for a fight with the developers (particularly the senior ones). Whenever possible, try to find a cost/benefit reason for using new technology. I need not remind you that it is not unheard of to find a software manufacturer whose claims for the potential of their new product don't always translate into reality. Some few years ago now, we make the bold statement (internally only) that all new projects would use browser-based technology. Shortly after that, we embarked on a new project and agreed to make it browser-based even though it was not planned to be a web application. We began the Elaboration phase trying to address the riskiest use case which happened to contain a data entry grid of some 14 columns and up to about 300 rows, each requiring field-level and record-level validation. After about 3 weeks, we gave up and instead chose a Windows user interface. Even though we had to throw away much of what was coded, all other

artefacts were still appropriate. It took less than 3 days to complete the implementation of the Windows version.

Conclusion

Most software projects get into trouble because of what the people on the project do or don't do and not because of the methodology being employed. Development methodologies are put there to provide a framework to avoid such problems. As I mentioned above, I have made the contents of Table 1 the focus of my Risk List artefact, the document in which I record all potential problems likely to jeopardise the project. I wouldn't claim to have problem-free projects – far from it. However, I feel that I now experience few last-minute issues than I used to and I feel better prepared for the future knowing that I have included the bad experiences of others in my risk management strategy.

I also feel less constrained by the rigidity of methodologies like RUP when I know that I can always change tactics at half-time if I feel it would help the project and further reduce project risk. After all, nobody has ever claimed to have *the* software development methodology but all claim something useful in the ones that make it to public awareness. Therefore, study their claims as use with discretion, a bit at a time if necessary.