Making the Most of the Software Development Process

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

Organisations are under increased pressure to look at development initiatives from a return on investment perspective, and failure in software applications can potentially have dire consequences for their overall business strategy. Delivering quality IT solutions and successful projects is about consistency and requires more than just general strategy. Delivering quality IT solutions and successful projects is about consistency and requires more than just general expertise. Specific knowledge of how tasks are performed in software development environments is crucial to the success of projects – not just for current operations, but also to support future strategy.

A software development methodology should be based on industry "best practices" but should also be tailored for the project, organisation and specific team. Implementing and internationally recognised standard such as rational Unified Process (RUP), which is a complex task and there is too much in standard "out of the box" solutions to be effective in an organisation.

However, in a recent survey it was found that of those companies who claim to use a software development process, 48 per cent followed as in-house process as opposed to a branded process based on best practices such as RUP or Dynamic System Development Method (DSDM). But how can a company measure whether its in-house process is appropriate and if its software development methods and techniques are up to scratch?

There are several ways to assess the sophistication of an organisation's software development methods and techniques. Of these, perhaps the most widely used is the Capability Maturity Model or CMM. The following table provides a definition of the following table provides a definition of the various levels of sophistication and allows the measurement of one organization against another:

Level	**Description**
1.Initial	Software process is ad hoc and occasionally chaotic. Few processes are defined and success depends upon individual effort and heroics.
2. Repeatable	Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3.Defined	The software process for management and development activities is documented, standardized and integrated into the organization. All projects use an approved, tailored version of the process.
4.Managed	Detailed metrics of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
5.Optimised	Continuous process improvement is enabled by the quantitative feedback from the process and form piloting innovative ideas and technologies.

As a rough guide, most software development organisation would like to be at, or would be satisfied with level 3; fewer reach level 5. However, based on observations, most organisations that do have a process flip-flop between levels 2 and 1. Those with no process at all would undoubtedly remain at level 1, with the emphasis on individual effort and heroics for any degree of project success.

Another way to assess an organisation is to measure against standards such as those from ISO or against the key quality drivers in published processes such as RUP or XP (Extreme Programming). The comparison of   an in-house process against another process/standers enables measurement of how well that process addresses issues commonly regarded ad best practice.
RUP claims to represent ‘best practices in software engineering’. XP, on the other hand, claims to represent a faster, more efficient way to write software with fewer rules and less documentation. Since these two methodologies probably represent two ends of current process spectrum, it is appropriate to compare in-house processes with them.

Yet regardless of which software development process is used, its purpose is to increase the chances of success to enhance the likelihood of subsequent projects also being successful as a result of lessons learned. However, this is not how many people perceive development processes and the result can sometimes appear to be merely recipes for document template completion or time-table for an endless series of meetings.

To ascertain whether a development process addresses real software issued or not, it is important to first look at why a project fails. Then how the process ultimately impacts on this can be examined. A quick search on the internet for “Why do software projects fail?” leads to numerous pages of well-intentioned advice. The Standish Group (www.standishgroup.com) is one organisation that specializes in rooting out the source of software projects problems. The following sections are an extract from the Chaos Report which provides the top-ten reasons for project success and failure. It also provides definitions as follows:

- **Project success:** *The project is completed on-time and on-budget, with all features and function as initially specified*

- **Project challenged:** *The project is completed and operational but over-budget, over time estimate, and offers fewer features and functions than originally specified*

- **Project impaired:** *The project is canceled at some point during the development cycle*

Given these definitions, it is interesting to look at what the research showed:

> *"The most important aspect of the research is discovering why projects fail. To do this, The Standish Groups surveyed IT executive managers for their opinions about why projects succeed. The three major reasons that a project will succeed are user involvement, executive management support, and clear statement requirements. There are other success criteria, but with these three elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically."*

Project Success Factors	% of Responses
1. User Involvement	15.9%
2. Executive Management Support	13.9%
3. Clear Statement of Requirements	13.0%
4. Proper Planning	9.6%
5. Realistic Expectations	8.2%
6. Smaller Project Milestones	7.7%
7. Competent Staff	7.2%
8. Ownership	5.3%
9. Clear vision & Objectives	2.9%
10. Hard-Working, Focused Staff	2.4%
Other	13.9%

> *"The survey participants were also asked about the factors that cause project to be challenged."*

Project Success 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%

But why include this research from the 1994 Chaos Report here?
There are two reasons: firstly, it is clear that software projects still fail, and secondly that they are still failing for some reasons despite the fact that technology has changed and the industry is another decade older. It stands to reason, therefore, that either current software development processes are not working or they are not being used properly.

But how can an organisation assess whether its process is up to scratch? Having looked at the reasons why projects fail, the next step is to examine how/if the current process helps to address these issues. To get the address the issues identified by The Standish Group. In-house processes can then be compared to examine whether the current process measures up, or if they simply generate paper.

RUP is sometimes perceived as being a bit on the heavy side, especially when compared with processes like XP. However, Rational claims that is based on Best Practices and as such it should readily attach all reasons why projects fail. The following is a list of RUP best practices and the success factors they address.

Rational Unified Process – Best Practices in software engineering?

RUP Best Practices	**Success factors addressed**
Develop iteratively	User involvement, proper planning and smaller project milestones
Manage requirements	Clear statement of requirements, realistic expectations, clear vision and objectives
Use Component architectures	
Model visually (UML)	
Continuously verify quality	Small project milestones
Manage Change	Clear statement of requirements, clear vision and objectives, realistic expectations, executive management support

Of the major success factors, the only ones not easy to ascribe to RUP are executive management support, competent staff, ownership and hard-working and focused staff. However, any project team with an existing process will benefit if these attributes are present in the project, and there is certainly no reason why a RUP project team should not have, and benefit from, such support.

The Rules and Practices of Extreme Programming *(XP)*[1].
The following provide a summary of the rules and practices of XP. Naturally, the full version of the process provides significantly more detail than space allows for here.

There are quite a few gaps in the right-hand columns where it may not be obvious which success factor is being realized in either process. Sometimes this is because the technique or practice is unique to the methodology, or because it describes a certain implementation technique or practice is unique to the methodology, or because it describes a certain implementation technique such as RUP's Component-based development or XP's pair programming. Regardless, it is pretty clear that most, if not all, the factors that influence success are enshrined in both these methodologies.

Planning	**Success factors addressed**
User stories are written.	User involvement
Release planning creates the schedule.	Proper Planning, smaller projects milestones
Make frequent small releases.	Smaller project milestones
The Project Velocity is measured.	Proper planning
The project is divided into iteration.	Smaller project milestones
Iteration planning starts each iteration.	Proper planning
A stand-up meeting starts each day.	Proper planning, ownership

[1] Copyright Don Wells 1999 (http://www.extemeprogramming.org/rules.html)

Designing	**Success factors addressed**
Simplicity!	Clear statement of requirements, clear vision and objectives
Choose a system metaphor.	
Use CRC cards for design sessions	
Create spike solutions to reduce risk	
No functionality is added early.	
Refactor whenever and wherever possible	

Coding	**Success factors addressed**
The customer is always available.	User involvement
Code must be written to agreed standards.	Competent staff
Code the unit test first.	
All production code is pair programmed.	
Only one pair integrates code at a time.	
Integrate often.	
Use collective code ownership.	Ownership
Leave optimization till last.	
No overtime.	Hard-working, focused staff

Testing	**Success factors addressed**
All code must have unit tests.	
All code must pass all unit tests before it can be released.	Ownership
When a bug is found tests are created.	Competent Staff
Acceptance tests are run often and the score is published.	

In summary, comparing like-with-like is clearly not easy when it comes to software development methodologies.

But in order to assess as in-house process the following areas should be addresses: does it address all the success factors and have an answer to all the project killers? Does it have more in common with RUP or XP? If it contains both it probably resembles RUP more closely. On the other hand, if it is less prescriptive, there may be more similarities with XP since it is clearly less formal and is inclined to be aimed at programmers rather than project managers.

One thing that is beginning to emerge from studies of this sort is that the details of the development process are probably less important than the following 'a process'-any process. In which case, an organisation's in-house process should serve it well if it is applied as intended. As long as it contains plenty of end-user involvement, has the full support of executive management and demands a clear, stable and complete definition of requirements, then the process stands every chance of resulting in a successful software project.