More RUP Anti-patterns

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

A recent article written by Julia Filho from Black Diamond Software (www.bds.com) - .pdf available from Rational web site under White Papers - has inspired us to recount some of our own experiences and to document them as RUP Anti-patterns (alternatively entitled "Problems I have encountered applying RUP").

Introduction

Using a development methodology for the first time is scary stuff. How do you know whether you're doing it right? How do you know whether it will ensure success because, let's face it, we've probably promised someone that if we use this new process, success is a given.

We help many companies every year either to run their own RUP projects or we will act either as Project Managers (using RUP) or as the RUP Process Engineers. Furthermore, we will provide an after-training back-up service and periodically run Iteration Reviews for companies doing their own thing.

This brief article sums up some of our findings (without naming-and-shaming) and gives a light-hearted insight into some more common problems that many companies encounter.

"How does my Gantt chart look?"

I have a confession to make: I hate Gantt charts and find them to be one of the biggest wastes of software Project Managers time ever invented! You disagree? Let's see…

Without exception, every company we have recently been involved with as RUP mentors have produced Gantt charts. All of them have been produced at the very start of the project and every one of them has been laid out if the project was going to follow a Waterfall Development life-cycle. That's the first mistake.

Since the Gantt chart was produced before the project started, every time-line on the chart for every activity was arrived at by guess-work alone with no hard evidence to support the duration of each task. In simple terms, 101 guesses added together! You've now set the expectations of the Stakeholders that since this plan was obviously arrived at by the application of science and a software tool, the plan must be worth taking on face value as being accurate. That's the second mistake – you've badly set Stakeholders expectations.

I once heard someone misquote a wise man saying "projects plans are either lucky or late". How can any plan be of value if it is not based on experience? If you need to produce a Gantt chart, at least wait until after the first complete iteration has been signed off. At that point, every RUP discipline will have been completed and you will have actual figures for the length of the time either activity physically took – that's worth putting into a plan.

However, Gantt charts also take a lot of time to keep up to date, all in all probably ½ day per week when you take into account the time taken badgering developer for the latest estimates. If the project manager was to spend the same amount of time ensuring that the process was being followed and making sure that all activates

Were done properly, my guess would be that the project would be run butter, the software would be of higher quality and the project would complete in the quickest possible length of time.

So how should a project manager manage project schedule?

> Firstly, estimate must be built on hard evidence. So don't commit to them until you have done at least one full RUP iteration.
>
> Secondly, only base progress on delivered and signed-off use cases (and then make absolutely certain that no delivery code is subsequently amended in any way!)
>
> Thirdly, use the concept of Timeboxes to drive your project schedule and don't make the Timeboxes longer than 6 weeks (if the team can't make 6 week deadlines, they'll never make an 18 month deadline).
>
> Fourthly, base estimates of effort only on the previous iteration – some developers get better at estimating, other get worse. This requires that you track time for every developer on each task they do. One technique worth trying when it comes to comparing the complexity of one use case to another is to compare the number of analysis or design classes involved in the use case.

Trust (but verify!)

A cornerstone of a RUP project is the assignment of roles. The process then defines the activities and responsibilities of each role. The Development Case then provides a list of project artefacts each of which will map to a role via defined activities. Since the process defines *exactly* how and when each activity should be carried out, the naïve project manager will assume that everyone is doing the job assigned to them as the process tells them. Wishful thinking!

A little while ago, before we settled on RUP as a company standard, all our project managers ran their projects slightly differently: Some liked to get a working prototype built early; others liked plenty of design documentation; other were only happy when their team were coding! Overall, a common complaint from developers involved in failed or failing projects was that they are never given enough time for design. Once, we had in internal project to do with absolutely no external/ client pressure.
Guess how they started this project? Yep – they fired up their compilers!

On the whole, developers are never happier than when they are coding. Almost anything else is seen as administration or documentation. When you assign your most experience developer the role of Software Architect or Systems Analyst, don't be surprised if they don't go to town producing Word artefacts or UML models – given half a chance, they'll be coding! To avoid this mistake, ensure that you build into the project plans ample (brief) artefact reviews. Use the RUP artefact guidelines and checkpoint pages as the agenda for these reviews and carry them out at least once per week.

“If we do all this paperwork, we won’t have time for coding”.

On the face of it, RUP looks top-heavy with paperwork. It shouldn’t be. One rule-of-thumb we use is to regard the production of artefacts as evidence that the associated activities have been carried out. In other words, completing artefacts is not the goal – doing the activities is.

Another tip is to encourage the idea that completion of any given artefact at a single point in time is not necessarily desirable. In other words, filling out every section in a RUP Word template is unnecessary *unless* it is needed in order to complete the next associated activity.

Encourage brevity. Leave artefacts incomplete. Review this decision at the end of each iteration and if you did too little, document a bit more in the next iteration. Start off doing too little and build up if you need to.

“We can’t use RUP – our projects are too small?”

This is by far and away the most populate excuse for not adopting RUP. “RUP is massive and most of our projects don’t require that much formality” is something we hear all the time. However, asked “Do all your project succeed?”, most still reply (with a grin) “Some do!”.

With a little common sense, you can apply RUP to a one-week project or to maintenance activities. The key is to understand the purpose of the RUP phases. According to RUP (Version 2002.05.00), here are the purposes of each of the phases:

Inception

- *Establishing the project's software scope and boundary conditions, including an operational vision, acceptance*
  criteria and what is intended to be in the product and what is not.
- *Discriminating the critical use cases of the system, the primary scenarios of operation that will drive the major design*
  trade-offs.
- *Exhibiting, and maybe demonstrating, at least one candidate architecture against some of the primary scenarios*
- *Estimating the overall cost and schedule for the entire project (and more detailed estimates for the elaboration phase*
  that will immediately follow)
- *Estimating potential risks (the sources of unpredictability)*
- *Preparing the supporting environment for the project.*

Therefore, if the project only lasts for one week, and is technically simple enough, we could achieve these objects in a few minutes. If we have very few risks and little in the way of architectural decisions to make, this phase has practically no point at all and could be skipped. As far documentation is concerned, a brief email documenting all of the above could suffice. That’s Inception dealt with.

Here’s what RUP says about Elaboration:

Elaboration

- *To ensure that the architecture, requirements and plans are stable enough, and the risks sufficiently mitigated to be able to predictably determine the cost and schedule for the completion of the development. For most projects, passing this milestone also corresponds to the transition from a light-and-fast, low-risk operation to a high cost, high risk operation with substantial organizational inertia.*
- *To address all architecturally significant risks of the project*
- *To establish a baselined architecture derived from addressing the architecturally significant scenarios, which typically expose the top technical risks of the project.*
- *To produce an evolutionary prototype of production-quality components, as well as possibly one or more exploratory, throw-away prototypes to mitigate specific risks such as:*
- *design/requirements trade-offs*
- *component reuse*
- *product feasibility or demonstrations to investors, customers, and end-users.*
- *To demonstrate that the baselined architecture will support the requirements of the system at a reasonable cost and in a reasonable time.*
- *To establish a supporting environment.*

As above, if we don't have any risks to mitigate and no architecture to prove… that's phase over. Alternatively, we might decide to start the project in Elaboration and make a single build or prototype in this phase.

Construction

- *Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework.*
- *Achieving adequate quality as rapidly as practical*
- *Achieving useful versions (alpha, beta, and other test releases) as rapidly as practical*
- *Completing the analysis, design, development and testing of all required functionality.*
- *To iteratively and incrementally develop a complete product that is ready to transition to its user community. This implies describing the remaining use cases and other requirements, fleshing out the design, completing the implementation, and testing the software.*
- *To decide if the software, the sites, and the users are all ready for the application to be deployed.*
- *To achieve some degree of parallelism in the work of development teams.*

This is where we would carry out one (maybe two?) iteration involving all the RUP disciplines. In other words, we turn RUP into a single Waterfall project. Why not if we have few if any risks to mitigate. Finally…

Transition

- *The focus of the Transition Phase is to ensure that software is available for its end users.*

Since we have to deploy the application for end-user acceptance testing, we will always need a Transition phase – but it doesn't have to take too long; one day might suffice.
So, as you can see, if the project contains few if any risks and little by way of architecture to worry about, we could reduce RUP to a single iteration or Waterfall. Documentation will still be required but, as mentioned above, only needs to be produced as evidence of activities carried out and decisions agreed upon.
And finally, a word of caution for project managers who have "been there, done it and got the tee-shirt".

"I've finished these use cases. Do you want me to start on the ones for the next iteration?"

This is a very hard one. The received wisdom is not to run parallel iterations or iterations that overlap. However, if you have a team of specialists each with skills involving one discipline, how do you keep them busy if you do iterative development? We have found this issue to be one of the most difficult to address.

Given that each iteration involves the same disciplines that would be found in a Waterfall development lifecycle, what do you do when (for example) your requirements gathers complete the documentation of their use cases? Clearly, they could start on the next lot that might be earmarked for the next iteration. But what if errors in style or depth of detail are discovered during analysis and design? Chances are that these same mistakes are already occurring in the documentation of the next lot. One of the major benefits of iterative development is that the whole team gets to practise their skills and learn from their mistakes. This should help the whole team get better and better at their jobs as the project progresses. The more parallel iterations become, the less this becomes a benefit and the more re-work that can be expected.

There are two practical ways around this problem: Frequent reviews and Multi-skilled teams.

Frequent reviews
Given that each iteration provides an opportunity to learn from mistakes, errors arising in one iteration need to be eliminated in all subsequent iterations. If these lessons are learned before the start of the next iteration, no re-work will be necessary as the new approach can be used from the start. If the iteration has already started, some re-work may be necessary. By reviewing activities frequently, the amount of re-work can be reduced. In practise, this has to be handled just as any project risk would.

Multi-skilled teams
This is the best solution: As soon as a team member finishes one activity, they immediately start another activity in the same iteration. For example, a Requirements Specifier could start documenting Test Cases when they finish documenting use cases or a Designer could start Implementation activities. The more single skilled the team members, the more likely overlapping iterations are to be a necessity in order to keep the team fully occupied. The goal of the project manager should therefore be to increase project team skills into more than one discipline, if at all possible.

“I’ve been managing projects successfully for 17 years – I don’t need a new process!”

This was actually said to me at the first coffee break of one of the first training course I carried out on behalf of Rational by one of the delegates. She went on to say that she was forced by her company to attend the course and that she did so against their will. Having spent a day and a half tutting and shaking her head on almost every point I raised, she was often seen to answer her mobile every few minutes during the last afternoon and was frequently not in the class room. Finally, she reappeared. Sheepishly, she confessed that she had “…Just converted to RUPism!”. It turned out that the reason for all the telephone calls and her absence from the class was because 2 projects that she was managing we just going live and both had encountered last-minute problems. “I knew about these risks some time ago. However, I took the word of our senior developers that there would not be a problem – they were wrong! Had I applied RUP, I would have proved the risks or otherwise much earlier in the project and we would now be facing the problems we are. Shame – we were on time before today!!!”. Her passing shot was “I’d still manage projects the same, mind you – just in a different order!”.

Win some; loose some. ☺