

An End to Methodology Wars?

Dr Graham Stone, Dunstan Thomas Consulting

<http://consulting.dthomas.co.uk>

Until recently, if you were to ask most professional developers what Development Methodology they used you would probably be surprised to learn few could name one. Similarly, ask most Developers whether they write Object-Oriented code and, to a man, they probably say that they do. However, ask the same developers which modeling language they use and what type of models they produce prior to coding, and most would admit to only a scant knowledge of Object-Oriented Analysis and Design (OOAD). Fewer still would tell you that they were proficient in the use of the Unified Modelling Language (UML), now arguably the industry standard. I dare say, even fewer would admit to a complete knowledge of any Development Methodology used in conjunction with UML and OOAS

A visit to www.standishgroup.com/chaos is not for the faint hearted. The site provides chilling reading with details of how software projects fail. For example would you credit that less than one in five of all software projects will succeed in hitting deadlines or budgetary constraints? Would you believe that about a third of all projects get cancelled? As one reads these horror stories, the odd thing is the persistent feeling of déjà vu. Projects fail for dozens of different reasons: Lack of end user involvement; inadequate design activity; failure to control change requests; lack of risk management; poor definition of roles and responsibilities; etc. The question, which kept coming into my head, was "Why aren't we learning from our mistakes?" (The answer kept coming back to me: Because "History teaches us that man learns nothing from History"!). Why isn't there "The" best way to write software?

I have personally been managing software development projects ever since the first IBM XT arrived in a blaze of excitement in our mainframe-oriented world. I have managed (or managed the managers of) literally hundreds of software projects. During this time I have read about and tried any number of "Silver Bullets" - both in terms of software tools that would guarantee results or development technique that would make project failure a thing of the past. Did they work? What do you think! Furthermore, it's an irrefutable fact that software development is getting more complex every year. The need for higher quality and more resilient architectures is also imposing huge pressure on development teams. It is not at all uncommon for almost every new project on development teams. It is not at all uncommon for almost every project to involve some new technology or tools. So, is there light at the end of the tunnel? I believe there may be.

Since the end of 1998, we have been using and teaching the Rational Unified Process (RUP). It's a development methodology based on an iterative project life cycle with risk reduction being the driving force of most activities. Requirements management is achieved through Use Case analysis while analysis and design modelling are carried out using UML. Architecture is focused on the use of Component-based design patterns. None of these is unique or propriety to Rational - and that is one of its strengths.

Rational have collated and documented six best practices of software development (develop iteratively, model visually, use component based architecture, verify quality, control changes, manage requirements) and moulded around them a development process which has been delivered in HTML in the form of a fully configurable web site (everything is in HTML or Word/MS Project template format). All the activities detailed in RUP are beautifully modelled in UML and are

supported by mini tutorials (Tools Mentors) which explain how to carry out the activities using other Rational tools e.g. Rose and Requisite Pro. In addition to this, there are very well documented guidelines detailing how to run a RUP project and how to configure or customize the process. There are even Roadmaps giving detailed guidelines on how to develop certain types of applications (e.g. e-commerce). Over time, I would expect the RUP community to produce more and more of these making the adoption of RUP considerably easier.

So what is it about the process that appeals so much? Firstly, it doesn't insult my ego! In other words, it doesn't tell me that everything I was doing in the past was wrong - I know that not all of it was wrong. Instead, RUP allows me to find a use for all my previous experiences but in a better, structured and more rigorous way without trying to kid one that it is "Silver Bullet".

The key premise of RUP is that If we can drive out the risk from a project, we stand a better change of successfully completed it. RUP following an iterative lifecycle (each iteration is effectively a mini-waterfall project) where one of the more Use Cases are worked on to completion in each iteration i.e. the result of each iteration is a fully working executable which may be tested and signed off. In order to determine what goes on in each iteration, we rank the Use Cases according to risk (defined as anything which is likely to cause the project to fail). The most risky Use Cases are completed early in the project. In many cases, it is possible to remove 80% of the risk of a project by completing less than 25% of the functionality. This is completely different to many waterfall projects where we don't get to find out whether everything works until we complete the system, testing right at the very end of the project. End user involvement is also strongly encouraged with all requirements being carried out using OOAD techniques with UML as the notation. Most projects are likely to require between 3 and 9 iterations (the more risky, the more iterations). This means that the entire team gets to practice their skill over and over again. This gives it a massive advantage over projects which use classic waterfall lifecycles where skills are only ever practiced once per project since each activity (requirement, analysis, design, code, test) is only carried out once per project.

Not so long ago, the panacea for all our development work was Rapid Application Development (RAD), as typified by methodologies like DSDM. In some hands, RAD stood for Design-free Application Development whilst for other it provided the client with the perfect opportunity not to consider the entire scope of the development activity ("Just do what I want now and I will work out what else I need later on"). Whilst perfect for maintenance work, RAD sometimes suffered from lacking "the big picture" making sound architectural decisions extremely difficult owing to the lack of future planning. RUP doesn't suffer in the same way because the entire scope of the project is considered at the start of the project. Before the first development iteration is started, RUP requires that we have at least 80% of our case model completed. At this stage, the model will certainly not contain all the details necessary for the OOAD team to specify what they need. However, the model must contain sufficient detail for the entire scope subsequent iterations in which the Use Case is worked upon).

Supporting RUP are a number of other Rational products: Rose for modelling; Requisite Pro for gathering and managing requirement; ClearQuest for change request and defect management and ClearCase for configuration management. None of them is necessary to run a RUP project but their use is a bonus, the icing on the cake.

I was recently teaching RUP to a group of highly experienced project managers. One particular delegate approached me during the first morning break. "I apologise if my questions and reactions appear to be rather hostile but I must tell you this. I vocally disagreed with attending this course and cannot see what I am going to get out of it. I have managed projects successfully for 17 years and have no reason to change my approach now!" What do you say to that! As we neared the end of the course, this particular delegate had to leave the room to answer a series of very important telephone calls. I later discovered that a project, which had gone live, had hit problems (key performance criteria had failed to be met once application was testing in the live environment and with real test data). At the very end of the course, the same delegate approached me again. I drew a breath. "I have just learned that a very important project has just failed to achieve customer sign off because performance targets were not reached. I want to apologise to you because if I had run it as a RUP project, I could have mitigated that risk much earlier on and we would not be in the position we are now." At that moment, I don't know whether this is something we can easily fix or whether we're now going to have a lot of re-writing to do". The delegate went on to say that whilst I had not taught many new tricks, the RUP approach would certainly have prevented the disastrously late discovery of risk, which turned out to jeopardise the whole project. "I'm converted!" was the delegate's and the class' final comment.