What's the difference between a Software Project Manager? and a Football Manager?

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

During a recently started project, I was trying to explain to a new client that we would be approaching the project using a particular development methodology and to illustrate this, I used the analogy of football manager: We had a team with well defined role and responsibilities and we had a strategy to win the game, or get the project delivered on time/budget, and we would employ different tactics to address different difficulties as the project progressed.

Admittedly, a bit of a dumb analogy. However, it did get me thinking: Why are some managers very successful and able to work their magic wherever they go (by the way, this has got nothing to do with Pompey being promoted to the Premier league this year as first division Champions!)? It's not all done by throwing money at the problem (witness Jim Smith and Harry Redknapp at Pompey); neither is it done by always having better players, although this helps a great deal (witness Arsenal throwing away the Premiership). Good managers, without doubt, get the most out of their players. They help them gel as a team; they enthuse and encourage them; they make them work for each other and he gives them the confidence to take on the world. But what happens on the day – after all the talking and training have stopped – when the players take the field? Does all the planning and strategising lead to goals and match success? Clearly, not always.

For many years now, I've been looking for the best way to organise a software project team. I've also been looking for the silver bullet development methodology. Furthermore, I've been looking for the perfect project to try them all out on. You know, a project that has no time constraints, loads of budget and crystal clear requirements. The trouble is, even after nearly 15 years of waiting, I still haven't found one. Am I just a bad project manager or am I just unlucky (or, as I strongly suspect, is it time for me to wake up and smell the coffee)?

I've been teaching the Rational Unified Process since 1997. I'm a firm believer in its virtues. However, I've also seen how the real world can put such strains on its structure that the whole shape of the project can be badly damaged. A good example is that of controlling iterative development – a key feature of RUP. On one project on which I served as Process Engineer (or RUP mentor), the client was unable to plan the recruitment and use of 12 contractors because I wanted to run each iteration in series, one after the other (no overlapping). This meant that we needed the contractors for 4 weeks but couldn't use them for the next 3 weeks until the next iteration was underway. The project manager insisted on keeping all 12 contractors busy and this started us on the path of parallel iterations. Although this might seem innocuous enough, it did mean that iteration reviews appeared to be taking place at almost any point in subsequent iterations. This in turn led to suggestions for improvement being implemented far too late. In a similar situation, as Process Engineer on a different project, we found ourselves unable to mitigate any major project risks because the QA department, who were all powerful, were unable to provide the team with a "live test" environment. This resulted in all system testing being delayed for 4 months while one was constructed. This meant that most of the construction phase was completed (functionality-wise) before we discovered that we were unable to fulfill non-functional performance requirements.

In both these examples, I stuck doggedly to my RUP-guns and badgered the project manager to do all he could to return to the straight-and-narrow RUP path. Clearly, he couldn't and the project found their way

up **** creek, without RUP paddles!

Have you ever wondered what is said during a half-time talk in a match were the losing team is clearly under performing? Various fly-on-the-wall docu-soaps have tried to answer this question. How does that team, looking so bad, come out for the second half, turn the game around and win? Could it be that rather than telling the team to stick to his original plan that the wise manager changes the plan and the tactics and maybe even the team content and shape?

I'd be the first to acknowledge that RUP cannot be expected to be "the" development methodology. The recent popularity and some notable success stories are testimony to this. We know of the success of approaches such as DSDM, FDD, XP (Extreme Programming) to name but a few trendy ones. But when you look carefully at almost any methodology, you find the self-same activities. Admittedly, they have different names and different emphasis, but they share more in common than one might think. Therefore, would it not make sense to accept the fact that the planned use of particular methodology might, in practice, not be suitable for the project that it was intended to be used upon? In other words, we might have to change tactics halfway through the game.

Let's look at a couple of examples. In one project, we found that the Stakeholders were signing-off requirements in the form of use cases and subsequently radically changing their mind. To address this, we made sure that screen shots always accompanied written requirements and this helped them visualise what they were agreeing to. As the project progressed, the stakeholders focused more on the screen-shots and less and less on the written use cases. This greatly upset the guys documenting use cases. In hindsight, wouldn't we have been better off acknowledging this and instead started to use elements of FDD or XP and accept the fact that use cases weren't our solution *in this case?* Unfortunately, we didn't and the subsequent proliferation of change requests was all the evidence that was needed.

In another project, our customer was extremely focused on delivery schedule and cost but not so focused on the functional requirements. In other words, they were not sure what was needed but knew that it had to be finished by a certain date and for under a certain amount. In hindsight, the DSDM approach using fixed time-boxes into which tasks of varying priority are placed, might have given us more control. DSDM uses the MoSCoW rules to determine priority: This means features that the application Must have, Should have, Could have or Would be nice to have. No time-box may contain only Must have tasks; there must be some Shoulds, Coulds and possibly some nice to haves too. When time runs out, which it tends to, all high priority tasks have to be complete but lower priority tasks are either dropped from the project or moved into the next time-box. In hindsight, this might also have given us a better handle on the rate at which budget was being consumed.

In conclusion, I still believe that there are no silver bullets in the Development Methodology game. However, I am now beginning to believe that it is the duty of project managers to learn more tactics and to learn more ways to play different games. If we regard each project as the equivalent to playing different opponents, we ought to acknowledge that the same tactics will not work in every game. The more approaches we learn to developing software, the more likely we are to being able to change tactics at halftime and still be able to win the game by the time the whistle blows.

Play up Pompey!