Top 5 tips for using agile in mission critical systems development

By definition, the cost of failure in mission-critical products is very high. Applying agile methods to the development of the software and systems that run them can help prevent the defects that lead to failure. Agile development methods can improve product quality, reduce costs and the time to market, and increase the predictability of their outcomes. However, it takes some fine-tuning of agile methods so that they make sense for these complex and rigorous projects. Agile governance, dynamic planning, test-driven development, incremental development and effective risk management are the keys to successful application of agile in mission-critical systems development.

Mission-critical systems and agile development

Mission-critical systems have a major impact on human safety and security. The cost of failure of these systems can be very high, resulting not only in financial loss, but sometimes in injury or death. Applying and adjusting agile methods to their rigorous and complex development can help prevent failure, improve quality, provide more predictable outcomes and reduce time to market and cost. Here are the top 5 tips you can use to apply agile to your mission-critical systems development.

Tip 1: Agile governance

Agile governance is about enabling development teams in an environment where teams can meet their objectives by avoiding the unnecessary hindrances that often plague traditional development efforts. The idea is for you and your teams to determine your objectives and how you will meet them. The most important step, however, is to truly make the decision to govern the project in the first place. Then, you need to understand what you’re going to measure, how you will measure it and how you will respond to those measurements.

Lots of people govern the wrong thing—things that are easy to measure. Agile is not about easy; it's about doing what makes sense to be successful and avoid failure. You need to decide the key performance indicators (KPIs) that correlate with your definitions of success and then use them continuously to monitor quality and completeness. One KPI could be project velocity, which means not only how much is getting done in terms of time, but also how much of that work is of
value. After you have your KPIs, you use them to steer the project. Transparency and visibility of progress, therefore, is crucially important.

**Tip 2: Dynamic planning**

Agile is a rigorous, disciplined approach to systems and software development. As in a waterfall approach, this takes planning. However, the difference is that for agile, the planning is dynamic. The main premises are that 1) planning must be continuously adjusted based on improved project knowledge and 2) your plans can only be as detailed as the depth of information that you currently have. Agile governance provides the "truth on the ground" and you should be prepared to make adjustments at least after every iteration, but often weekly or even daily. You will get more information as you progress and that information should be fed back into your plan.

I also recommend taking a two-tiered approach to dynamic planning. The first tier is an overall plan that is roughly based on a set of planned iterations of four to six weeks, each of which includes the build and immediate testing on the build. The second tier is the more detailed planning used for each iteration, or the description of what is going to go on in that month to month-and-a-half. At the end of each iteration, you review what has been done and compare it to the plan. If there is a discrepancy—that is, if the results don't exactly match the plan—you go back and adjust the plan to reflect reality.

**Tip 3: Test-driven development**

Removing defects from completed software requires a great deal of effort and can be costly. In fact, studies have shown that 60 percent of software development effort is typically spent in testing and defect removal. It is better to avoid putting defects in the product in the first place. You can do that with test-driven development. With test-driven development, you are continually testing in parallel with writing the software. You use nanocycles of 20 - 60 minutes, and you use the half of the nanocycle to write the code and the half to do a test that demonstrates that it works. The type of project you do determines your nanocycle. For example, I worked on a spacecraft where the nanocyles averaged 20 minutes. The result is code that is of much higher quality and has a far lower defect rate, reducing the cost of testing at the end of the project.

Another agile best practice is to use continuous integration to bring the work of the team together and demonstrate the integrated correctness with functional flows that use all components. Continuous integration is typically done daily throughout the entire project, although some organizations do it weekly. In general, the earlier you discover a defect or integration problem, the cheaper and easier it is to redress. At the end of the project, there will be none of the integration issues that you see when the more traditional approach of throwing together individual segments of software, developed in isolation, is used. Again, defect avoidance is always better than (late) defect repair.

**Tip 4: Incremental development**

Projects for mission-critical systems are inherently difficult and complex. The best way to approach them, therefore, is the same way we’re encouraged to tackle big projects in life: break them into
smaller pieces. Construct your system as a series of iterations ("sprints") that are each 4 to 6 weeks in length.

Each iteration should have a mission statement. The goal is to fulfill its mission, which focuses on the requirements to be implemented (usually captured with user stories or use cases) but also includes the platforms to be supported, architectural concepts to be included, risks to be mitigated and existing defects to be repaired. Verification and validation testing is conducted at the end of each iteration.

For my projects, we also have what I call a "party phase" at the end of each iteration. Some people call this a post-mortem, but I see it as a celebration of ongoing success rather than as an autopsy to determine what failed. We look at the mission statement, determine how well we met the statement and look for ways to improve the process. We always ask what could we have done better, faster, more efficiently?

Tip 5: Dynamic risk management

Risks are all the things you don't know in a project. In my experience of more than 300 projects, ignoring risks is the leading cause of a mission-critical system project failure. Yet organizations rarely do any kind of risk planning, and those who do create the plan once and then never look at it again. This is a recipe for disaster. You must pay attention to risks when you are building mission-critical systems. Therefore, you should use a dynamic risk management plan, also called a risk list, to manage your risks and update it frequently.

In the risk lists or risk plans, you identify risks which are a function of two values—the severity of the outcome you want to avoid and how likely that outcome is. The product of those two values, severity and likelihood, results in a quantitative measure of risk that you can track. You then execute risk mitigation activities, also called "spikes," which are planned activities that reduce risks. Because not all risks are sufficiently severe or likely to warrant special attention, risk mitigation activities are only created for risks above a threshold level. When the risk is high enough, a risk mitigation activity is planned and becomes a scheduled activity.

Conclusion

Agile methods, when properly applied to the development of mission-critical systems, can help prevent costly failures. Agile methods can also improve their quality and decrease the effort it takes to apply them. They draw attention to risks, which, when ignored, are the leading cause of failures and they dispense with the static, ballistic planning that also leads to failure. Although tools are not as important as the methods, IBM Rational software tools such as Rational Team Concert, Rational DOORS, Rational Rhapsody, Rational Quality Manager and more, along with Jazz, can help bring the automation and transparency needed to reduce error-prone processes and improve planning. Add in agile best practices, such as those found in the Harmony process, and you've got a winning combination.