developer.*
The Independent Magazine for Software Professionals

Automating Software Development Processes

by Tim Kitchens

Automating repetitive procedures can provide real value to software development projects. In this article, we will explore the value of and barriers to automation and provide some guidance for automating aspects of the development process.

Although few experienced developers and project managers would argue the merits of automating development and testing procedures, when push comes to shove many teams place a low priority on implementing automated processes. The result is usually that, if automation is considered at all, it is given lip service early in the project life cycle, but falls quickly by the wayside.

Experience teaches us over and over again that trying to run a “simple” project by implementing a series of “simple” manual protocols, backed by “simple” written (sometimes, even just verbal) instructions, just doesn’t work well. Even so, many of us still tend to allow ourselves to start the next project with the thought that the manual, protocol-based method will “do just fine.”

After all, aren’t we all professionals? Can’t we read a set of simple instructions and just be disciplined enough to follow those instructions when the time is right? Isn’t it a waste of time and money to invest in automating procedures for such a small project? The development team is only a half-dozen people after all—and the argument against automation goes on and on.

If you’re a developer or tester who enjoys spending your time actually adding value to your project, rather than repeating the same routine tasks over and over, you’ll want to consider advocating the concept of automation to your team (especially, to your project manager). If you’re a project manager who’s committed to maximizing the talents and time of the members of your technical team, as well as minimizing the risk of your project failing to deliver on time and on quality, you will want to encourage your team to invest the necessary time and effort required to automate the types of tasks that will be identified in this article.

Why Should I Automate?

You may already be familiar with many of the benefits of automating development processes. Some of the more commonly cited ones are:

Repeatability. Scripts can be repeated, and, unless your computer is having a particularly bad day, you can be reasonably certain that the same instructions will be executed in the same order each time the same script is run.

Reliability. Scripts reduce chances for human error.

Efficiency. Automated tasks will often be faster than the same task performed manually. (Some people might question whether gains in efficiency are typical, noting that they have worked on projects where, in their view, trying to automate tasks actually cost the project more time than it saved. Depending on the situation, this may be a real concern. In addition, automation might have been implemented poorly or carried too far on some projects—but keep reading for more on what to automate, and when.)

Testing. Scripted processes undergo testing throughout the development cycle, in much the same way the system code does. This greatly improves chances for successful process execution as the project progresses. Automated scripts eventually represent a mature, proven set of repeatable processes.

Versioning. Scripts are artifacts that can be placed under version control. With manual processes, the only artifacts that can be versioned and tracked are procedure documents. Versioning of human beings—the other factor in the manual process equation—is unfortunately not supported by your typical source control system.

Leverage. Another big benefit to automating is that developers and testers can focus on the areas where they add real value to a project—developing and testing new code and features—instead of worrying about the underlying development infrastructure issues.

For example, instead of requiring everyone to become intimately familiar with all the little nuances of the build procedure, you can have one person focus on automating the build and have that person provide the team with a greatly simplified method of building, hopefully as simple as running a command or two. Less time spent on builds leaves more time for the tasks that add the most value to the project.

What Should I Automate?

If you're convinced that automating your development processes is a good idea, the next logical question is which processes should be automated. While the answer, to some extent, is different for every project, there are some obvious ones, as well as some general guidelines that I'd like to offer. Some of the typical targets for automation are:

- Build and deployment of the system under design.
- Unit test execution and report generation.
- Code coverage report generation.
- Functional test execution and report generation.
- Load test execution and report generation.
- Code quality metrics report generation.
- Coding conventions report generation.

Deleted: s

The above list is obviously not exhaustive, and every project has its own unique characteristics. Here's a general, and perhaps obvious, rule of thumb to help identify any process that should be considered for automation: Consider automating processes that you expect to be repeated frequently throughout a system's life cycle. The more often the procedures will be repeated, the higher the value of automating them.

Once a process has been identified, spend a little time investigating how you might be able to automate the process, including researching tools that could assist with automation, and estimating the level of effort required to implement the automation versus the total cost and risk of requiring team members to manually perform the procedures. As with any other business decision, it really should come down to a cost versus benefit analysis.

You probably noticed that the term "report generation" appears in the above list of automation candidates. The repetition points out another important aspect of automating development processes: the end result of every automated process execution should be a report that is easily interpreted by the team. Ideally, such reports will focus on making anomalous conditions (for example, test failures) obvious at a glance. Also, these reports should be readily accessible to the appropriate team members.

In many situations, it's even a good idea to "push" reports to the team (perhaps via email or RSS), instead of requiring people to remember to go out and search for them. The basic idea is that the development team should be notified as soon as possible when problems are introduced to the system or environment. A side benefit is that management is provided a more tangible real-time view into the progress and health of the project than is possible with team status reports alone.

When Should I Automate?

Consider automation as soon as you realize that team members are starting to execute a process that meets the criteria discussed in the previous section. For example, automating the build process when the project is nearly over provides very little benefit. It can, however, save dozens, if not hundreds of hours when automated as soon as development begins.

However, before you go off and start automating everything, one caution: be reasonably certain that the procedure will be required for your project, that it will be repeated more than two or three times during the project's life cycle, and that you understand the steps that need to be executed for the procedure. There are some things you just don't do very often, and in these cases the costs may outweigh the benefits.

Even for processes where automation is warranted, if you're truly guessing at the steps involved, there's a high likelihood that you'll end up re-writing the whole thing. Re-writing is much different than fine-tuning or refactoring the automated scripts. Refactoring is expected, but scrapping and starting over again means you tried to automate before you understood the process you were trying to automate.

Another danger is allowing your project to become the "proving ground" for automation for the entire organization. If your organization doesn't already have the infrastructure in place to support development automation, you should provide what makes sense for your project, but try not to allow your team to lose sight of your project's goals. A project team whose goal is to deliver working software is in no position to develop enterprise-wide strategies and tools. Trying to do so is asking for trouble.

If outside entities begin to get involved in these efforts, I'd suggest that you recommend that a separate project be undertaken to work out the automation infrastructure for the enterprise. Members of this "automation project" team are free to review your project's implementation for ideas or for use as a launching point. Once this infrastructure is in place, the question of *When to automate?* is easier to answer for future projects, and less costly.

Obstacles to Automation

If there are so many benefits to automation, why don't we see more of it on software projects? All software development teams struggle to balance the need to show immediate results with the long-term goals of the project. There are many obstacles to implementation of development process automation. Here are a few of the more common ones:

- Stakeholder pressure. Faced with the desire to show early and rapid progress, teams often overlook or choose to ignore practices that do not seem, at first glance, to directly contribute to getting working code up and running as quickly as possible.
- Focus on core requirements. The team has an overwhelming desire to begin producing working code and tests as soon as possible. Writing code that can be demonstrated is much more satisfying to developers than writing automation scripts.
- Lack of management support. Management may not have a good understanding of automation—how it works and/or the costs and benefits.
- Lack of organizational support. There is no enterprise-wide policy or infrastructure for development automation (standards, tools, expertise, etc.)
- Lack of follow through. Even with the best of intentions, teams can quickly lose their commitment to plans for implementing automated processes.

As with any fundamental change, there must be someone who's both committed to the concept and who has the authority to make sure that teams follow through with the plan. Not following through is much worse than never having taken the time to investigate and discuss automation. First of all, there are probably other problems that you could have successfully solved using the time and resources and, second, if you fail to implement any automation, when the idea surfaces again people in the organization will point out that it didn't work the "last time we tried it".

Selling Automation

As implied in the preceding section, the primary obstacle to automation is short-term thinking. Therefore, your primary task is to get your team and management to pause to think about overall project costs and schedule, instead of focusing solely on meeting next week's promised deliverables. The message should be clear. Meeting milestones is critical, but it's very possible that you could meet that next milestone to the detriment of the project. But before you try to convince people who are under a great deal of pressure that you it sometimes makes sense to slow down momentarily in order to improve the chances for success, you'd better have more than a set of maxims at hand.

This is where Return On Investment (ROI) can help. Most people in the business world, regardless of field, have at least a basic understanding of ROI and agree that, when based on valid assumptions, it can be one of the best mechanisms for evaluating whether or not to take action or adopt a particular solution. The process of calculating ROI is well beyond the scope of this series of articles. However, the Suggested Reading section at the end of the article provides a link to an outstanding article from Edward Adams that describes how to do this, as well as some additional advice on selling the case for test automation, in a very straightforward and easy to understand way.

Although many managers really like to see quantifiable benefits, such as ROI, before making major decisions about adding tasks to the project schedule, others have sufficient experience with software development and are just as comfortable with a common-sense explanation of automation's benefits. With that in mind, rather than going into a full-blown ROI calculation, let's just take a look at one scenario that most projects will face at some point to see, from a common sense perspective, the likely outcome of a purely manual process versus a process that had been automated early on in the project.

Manual testing approach. Your team is in the final month of a twelve-month project, and the system test team is starting to perform regression testing by manually re-running the entire set of test cases. Some test cases haven't been executed in several months, because, frankly, the testers have just been spread too thin, and, due to perceived pressure to deliver, management chose to move testers onto other tasks as soon as they were finished wrapping up a set of test cases.

Regular regression testing was put into the schedule early on, but repeating the same tests over and over throughout the project was (whether or not anyone wants to admit it) eventually considered a “nice to have” and definitely much less critical than other, more pressing tasks. Now some of the regression tests are failing and the test team is logging defect reports.

The development team decides to manually re-run the entire unit test suite (which also hasn’t been executed in a few months, because it wasn’t automated) to hopefully zero in on the source of the problem. The developers find two problems as a result of the unit test run: first, apparently there were never any unit tests written for the parts of the system in question; and second, there are other, seemingly non-related unit tests that are failing. To make things worse, the components that are failing were developed by someone who’s no longer with the team. Even if he were available, he’d probably require significant spin-up time to re-acquaint himself with the failing code, since he hasn’t even looked at it for months.

Automated testing approach: Early on in the project, the decision was made to automate at least the following: unit testing, code coverage, and system testing. The same bug that caused the situation above was introduced in month three of the project. The code coverage report that’s automatically generated as part of the daily build clearly indicated to the development team that there were no unit tests for the related components as soon as those components were placed under source control. Therefore, the developer went back and immediately implemented the unit tests.

After implementing the unit tests and working out the bugs that were initially revealed as a result, the developer checks the unit tests into source control. The daily build automatically picks up the new unit tests, the coverage report reflects that the components are now being tested (as well as to what extent they’re being tested), and the unit test report indicates test success or failure each day. As soon as any change is made to the system and placed under source control, the team will become aware of the impact that change has on the overall system, while the changes are still fresh in the minds of the implementer.

The system tester who’s responsible for testing that same set of features now designs his test case, implements the tests, reports any defects he discovers, etc. until he feels that the code and test case are operating as expected. He then uses the test automation tools to record the execution steps for the test case and checks the resulting scripts and files into source control. The next day, the automated test runner picks up his new test case for

execution and the results become available to the test team as part of the daily test run. The development and test teams are leveraging the automated tools and processes for proactive detection as problems are introduced.

Without performing a detailed ROI analysis, which of the two above scenarios is intuitively more desirable? Assuming that the exact same bug is introduced into the system at the same time—month three of the project—in each case, which approach do you think will result in the least expense and least risk? Which will result in the least stress to the team and to management?

Start Small

Unless you're lucky enough to have a team with multiple members who've had a good deal of experience automating development processes, you won't want to try to take the idea of automation to the extreme. Trying to do so with an inexperienced team or a team that's not completely sold on the idea of automation will likely result in failure, as both expectations and barriers are set high from the start. Pick one or two areas where you believe there is both high potential value for automation and a low risk that the team will fail implementing automation. You might ask yourself the following questions:

- Which processes will likely be exercised the most frequently?
- Does someone on the team have the experience and/or skill set to implement the automated process?

It's better to succeed in a small way than to fail in a big way. Your successes just might build on themselves, and when you suggest additional areas for automation, either later in the project or on your next project, you may face little or no resistance at all. To reach the ideal takes a long-term plan, management commitment, adequate tools and training, and of course time. If it were easy, everyone would already be doing it.

Conclusion

I'm well aware that there may be other factors that affect the decision of whether and what to automate than those we've addressed in this article. For instance, you may find that the only automation tools your organization will approve for your team's use are beyond your budget or that, due to your project's special needs, there are more important factors. Maybe you're a vendor and your marketing department has determined that time-to-market for your first release is more critical than the quality or the long-term success of the current product.

I'm a big believer in the "it's all about trade offs" principle. To gain one advantage, you typically accept other disadvantages, and in the end you should try to make decisions that tip the scales most in favor of the "advantages." I would say, however, that if your project is truly one of those projects that is better off not automating, you're in the minority category—the exception, not the rule. If the primary obstacle to adoption of automation for your organization is lack or expense of tools, there are several free open source tools that can go a long way towards solving that problem. The time you spend investigating automation strategies and tools could very well make the difference in whether your projects successfully meet schedule and quality goals.

Suggested Reading

Edward Adams, "The Business Argument for Investing in Test Automation."

http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/dec02/TestAutomation_TheRationalEdge_Dec2002.pdf

###

Tim Kitchens has over 10 years of experience in the software industry. He has served in roles as developer, architect and project manager. His primary professional areas of interest and expertise are software architecture and implementing effective development processes. Find out more about Tim in the "Contributors" section at www.developerdotstar.com.