Automated Build and Testing

Alexander Lyons
Southern Polytechnic
1100 South Marietta Pkwy
Marietta, GA 30060
alyons2@spsu.edu

ABSTRACT
In this paper, the topic of automated building and automated testing will be discussed from the standpoint of applying them to a team's active process and what impacts these two processes have on building software. This paper will also analyze a case study to see what results were gained from implementing test automation and if the impacts discussed prior are shown in the case study.

INTRODUCTION
Build and test automation are tools used in software engineering to improve the process of developing code bases by utilizing existing resources in new ways to add speed to this process. How this works is that rather than having developers spend their time compiling code and testing code (especially for unit and regression testing), computers can be instructed to perform these tasks and give results based on the tasks which are performed. There are tools available for both of these tasks, some of which actually have some cross over functionality.

BUILD AUTOMATION

What is Build Automation?
Build automation is the automatic compiling and linking of software (in parts or as a whole) using programs and scripts to perform all of the tasks necessary to build the software. This concept is that the build happens as though a person was actually performing the task; in an automated build system, the process will document failures along with their causes, and can even perform alternative flows in certain scenarios.

There are three types of automated builds; on-demand, triggered, and scheduled. On-Demand builds begin based on human input. While this could be as simple as hitting the build button on an IDE, the steps involved in a true automated build service go much further than just building the software, with activities such as archiving the build, and reporting on the build process. So, this idea is not just hitting the compile button, but having a directly human accessible trigger to cause the build process to run.

Triggered build operate from the fulfillment of some trigger condition, i.e. a submission of code to some form of repository. This method uses the idea that whenever some specified event occurs, it would require the data from a compile attempt to validate that the event was successful.

A scheduled build happens at a specific point in time, often happening on a daily or weekly repeated cycle [1].

What is the Direct Benefit of Build Automation?
Build automation saves time. When programmers are forced to compile large projects to see if they even finish the build process and what potential errors there are, they end up spending the time which the program is compiling not programming. This can work in some cases in which there is work which a programmer can do which is not tied directly to the source code, but this environment requires a process to already exists within that development team, and is more often than not non-existent. Often times, programmers may read related articles to what they were working on or even just not working on anything related to their office. In large projects, compile time can take multiple hours, which means a large loss of productive code hours. For example, if a project takes four (4) hours to compile and get data on the compilation, and there are ten (10) programmers working on the project, all of which cannot make changes to the code while it is compiling (due to something as wanting to see the results of compilation before making a decision on how to proceed), then every time the code is compiled while the programmers are working at the office, forty programmer hours (40 = 4 hours x 10 programmers) are lost. Using build automation, builds occur at night after the programmers have left the office, which means a loss of zero hours of code time.

Another benefit to automated build process is that the build process is set by configuration, which means as long as the system does not require a change to the build configuration, the system will be built correctly every time with regards to the build process. And if there are any discrepancies with the build process, they can be changed in a centralized manner, rather than having to train each developer who might have to build on the new build process. This means

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CHI'13, April 27 – May 2, 2013, Paris, France.
Copyright 2013 ACM 978-1-XXXX-XXXX-X/XX/XX...$10.00.

less bad builds which means more time working with correct projects.

How to Best Utilize Build Automation?

The best way to utilize build automation is to use a tool to manage the build automation, so that results and data can be well monitored and reported in a regular fashion.

Triggered build works best in an environment where build time is not necessarily terribly high (usually within an hour) and the team of developers is well connected (probably small), so that builds can occur without interrupting work flow too much.

Scheduled builds work best when used within teams that are large and the system build takes a long time (usually an hour or more) and the teams working on the project are more isolated.

Warnings of Build Automation?

One thing which is often mistaken for build automation, is having a computer build a project through the use of a scheduled task. This is what Microsoft recommends doing for automated builds on a Team Build project. The reason as this is not a form of build automation is that there is no relevant data gathered from the build process, which means if something happens outside of the expected results, then the developers will have no data to trace to the cause of said outlying event.

TEST AUTOMATION

What is Test Automation?

Test automation is the running of test using programs and scripts rather than people to perform tests and gather the results of such tests.

There are quite a few forms of testing which occur, most of which fall under two categories of testing; code-driven testing and graphical user interface (GUI) testing. Code-driven testing is testing based on the code base for a system and is well optimized by utilizing test automation. The two largest examples of this are unit testing and regression testing [6].

Unit testing is well met by automated testing in that, on an atomic level, code with a specific set of inputs should return a specific set of outputs, and if all of those ranges can be easily covered by a small set of test cases with predictable outcomes. This means that a computer could easily report on a binary level whether or not a unit successfully completed all of the test cases, and which test cases it failed and how.

Regression testing also works well when handled by a computer rather than a person. Regression testing being done by a script or program means that each new or updated function can be tested against the functions it interacts with and each of these test can be done rapidly with proper recording of outputs, as these outputs are data centric rather than judgment based.

Automated testing for graphical user interfacing is a bit more difficult and has come a long way from its origins. The most common method of doing graphic user interface automated testing is to use scripts representing use cases to perform tasks and gather results based on the outputs. Scripts are built in two ways. The first way is that a user's inputs are recorded as they perform a test case scenario and then that capture input is run by a program to recreate the test case.

The second way test case scripts are generated are within some testing solutions, these solutions offer the ability to generate scripts based off of user interface objects (e.g. buttons, texts boxes, etc.) and writing out their interactions manually.

What is the Direct Benefit of Test Automation?

The direct benefit to test automation is the ability of speed and repetition. By using test automation, test can be run much more rapidly than using human input and recording. Also, each test can be run multiple time and is guaranteed to have the same input sequence as long as the script is not changed. This means that problems can be monitored across changes with the code lying behind the interface without having to utilize human resources for every set of changes.

Test automation will also allow for simple errors to get caught before testing reaches human testing, which means that the testers will not have certain errors obstructed by the simple errors such as having first name saved in the last name data slot because of copied code.

How to Best Utilize Test Automation?

Warnings of Test Automation?

The first warning about test automation is that it is largely dependent on writing good test in each scenario in which automated testing is used. This is relevant in that a computer cannot interpret the inherent meaning behind a test case, so on-the-fly test case improvements cannot be made, so a faulty test or test case can result in improper test results.

The second warning is that this is not a replacement for user interface testing. The first piece of information you will receive from a human that you cannot receive from a computer is how they react to the interface; do they like or dislike the layout, how do they feel about the aesthetic choices with color and font, what do they think could be done to improve the look of the software, etc. The second major piece of information one could derive from a user is expected interface interaction. As a computer follows a script and will have no preconceived notions of how they would interface with software, they can only act in a linear fashion. A human tester will have a background and a history with computers, and this will affect how they

approach solving a test case. Ideas like the use of common shortcuts and such cannot be tested by the machine in an expectancy way, and much can be gathered from a user reacting to the existence of an expected interface option.

BUILD AND TEST AUTOMATION TOGETHER

How both of these ideas work together is quite simple. The automated build runs itself and the automated test based on its own results of running.

For example, one could have as part of the configuration in side of CruiseControl.NET that upon successfully building a project, unit and regression test are run and return the results by sending an email to the development team. These results could be tailored to what the company wants to be returned and would allow for a development team to perhaps run builds and test both when they leave for lunch and overnight while no one is at the office.

AVAILABLE TOOLS

In this section, this paper will highlight some of the tools which would be good to utilize for build and test automation. Solutions will be noted as whether or not there is a free solution, and what the software does on a high level.

TeamCity

TeamCity is a continuous integration tool that works for many different programming languages. One of its best features is that its professional license (the license used by small teams) is completely free. It includes both build and test automation, which is something which is prevalent among many build automation solutions [5].

CruiseControl.NET

CruiseControl.NET is a product which is also free, but there is less with regards to the user interface when building and using the solution. One of its main features is integrated unit and regression testing [3].

Selenium

This is a tool for testing web based applications. It runs based off of scripts which can be recorded or manually created. It also includes a way to expand the testing across multiple environments. This however is not the most robust when doing error reporting, but will output with errors pointing to where a script failed [4].

Visual Studio Test Professional

This tool is Microsofts test tool to integrate with the Microsoft IDE Visual Studio, and will allow for the actions of a GUI to be recorded and various data on the scripts and will be translated into source code. This will allow for tests to be easily modified and recorded results to be predictable. The data which is recorded for GUI tests is the success of a test, the point at which a test fails, and even a screen shot of the test failing. This not a free solution, and actually comes with a hefty price tag of over two-thousand dollars ($2,000) [7].

STUDY OF MAGNIMBUS

On a project I previously worked on, we did not implement build or test automation. We lost much time in building and testing our software, something which is far more noticeable in a small team than a large one. Here I will discuss some of the development process and where time was spent, and looking at the tools which the team would have most likely used, formulate data to show how much time would have been saved and show the difference between hours lost on a large team versus a small team.

Project Definition

The particular section of the system we will examine was a communications system between a desktop application and a server system. This system was designed to have a relatively one to one correlation between actions performed on the desktop and actions performed on the server. The function worked as follows; the desktop (client) would create a message for the server, looking for at least a confirmation response on the message's reception. The message would be sent to a server, which would process the message, and would send it to the correct server which would call the methods to access the database and return the results of the request in a specific manner. This project had approximately two-hundred fifty (250) commands which would be accessed either by the client directly or would facilitate inter-server communication when processing commands.

Project Breakdown

What occurred during the project was all manual building, deploying, and testing of this system. Between the two developers working on the system, one building the client which will be communicating from and one building the scripts residing on the servers, there was a total of well over four-hundred (400) man hours spent building and testing this system. Writing the code to actually make the server and client connection for that method would take approximately fifteen minutes. Building and deploying the server code locally took approximately five minutes before any code could be run to test if the system worked. Testing the new command took approximately two minutes. Then, if the command worked, another build process would have to be run to upload the new code to the online servers which would run about fifteen minutes. After this upload, the code would have to be examined to make sure that in applying it to the server, nothing when wrong, which was on average three minutes. Then the client program would run a test to see if the method worked both by itself and within the parameters of the larger system which this communication was a part of. This process would take up to five minutes. This entire process does not account for changes made to code based on errors or requirements. This means that for any given command, at least one and one-half man hours

was spent building and testing the command (two developers each spending forty-five minutes working on the command on both ends). This would end with three-hundred seventy-five (375) man hours spent building the communication between the server and the client.

The above calculation does not take into account dealing with errors. Even accepting the idea that only thirty-five percent (35%) of all methods were defective and not accounting for any regression testing, that would put the necessary work hours towards building this system at over five-hundred man (500) hours to complete the system (this is assuming a one to one rework ratio on any given broken method). This also does not include the changes that were made to the way the product would function which would make previously correct methods fall into the defective category. So when an actually number was estimated to show how much time was spent on dealing with the client-server communication it was well over seven-hundred man hours of development. And this does not include the web version of the client, which could well put the man hours above one-thousand.

Applying Build and Test Automation

In this scenario, we could have applied various build and test automation to make the process very much easier.

For build automation tools, we would have used CruiseControl.NET for the desktop communications development and Hudson for the JavaScript. This would allow for the building to be done more quickly and would allow for the correct measures to be taken at each build.

Moving to testing, using a tool such as CruiseControl.NET or TestDrive, we would be able to leverage the tools to perform unit and regression testing over this system in the project (which is the only form of testing which occurred during this phase) on a much larger scale and much more quickly than we could have done. Also, this work could have all been done outside of our work hours. How this works out is that for and given command only fifteen minutes is spent coding a command. The building, deploying, and testing all happen outside of work hours and would remove thirty minutes from the work time. Assuming that just building commands would take us fifteen minutes with a thirty-five percent defective method ratio and that each corrected method would take approximately fifteen minutes to fix, the total man hours spent by the two developers come to about one-hundred seventy hours. This means that development time would have only been approximately one-third of the time spend building and correcting the software. Even including the learning and set up time for the software, which would be approximately two to three days, this is still much less time than the original development process. This means that rather than taking one and one-half months to build the communication end, it would have taken less than three weeks.

How this Effects Small Teams

In a small team working on rapidly iterating projects with quick turn-around times, ever hour is important. Any time not spent developing code or a working product is seen as wasted, especially by managers and venture capitalists (VC). This means is that if a VC cannot see the production being expedited by the process directly, then it will not be approved. This comes at a huge price (as demonstrated above) as time is spend waiting on builds and tests to finish rather than moving forward in production.

Cost is the most prevalent factor in this. The idea is that software and solutions costing two-thousand dollars ($2,000) per year will be more expensive than the hours spent working on the project. Assuming that each developer hour only cost twenty-five dollars ($25) as this was in house development, then what is the production cost under Magnimbus for the above scenario. Well just working with the number calculated in the paper then, the cost for our original work effort would be twelve-thousand five-hundred dollars ($12,500). The data which would assuming it would take three weeks across two developers puts the time at approximately six-thousand dollars ($6,000), which is half of the cost in time. Even adding approximately three-thousand dollars for software ($3,000), the total cost would be nine-thousand dollars ($9,000) of cost for the development team to VC and only three weeks of their time, rather than twelve-thousand five-hundred dollars and six weeks of their time.

What this relates to is the fact that VCs do not understand how building software works and do not understand the time and cost differences in software. There is also a lack of understanding on the part of novice developers with regards to estimating time and cost of software. On major part of the buying build and testing solutions is that they are often willing to negotiate on prices and some are even free for small enough teams. Each company is different, but TeamCity currently offers their solution for free for teams with three build agents (i.e. three developers pushing to the build servers). This means that the only cost is time in implementing the data for really small teams. These are points of data that are not looked at in the field, as these are more business oriented but are vitally important in changing the environment in which software is built, businesses.

CONCLUSION

This paper examines what automated build and testing are, how they are applicable in a business environment, common practices to follow when using these process, some pitfalls to avoid when using these processes, and some of the tools which are available to make these processes a reality.

One of the largest disparities in the field is the acknowledgement of the importance of build automation and the usefulness of the data which can be gathered from that point. Also, by using correct forms of build automation,

test automation can be made to be even more fruitful in data returns.

This paper also examines how even the smallest development teams could implement build and test automation, and looks at the idea of how much more important developer time is in a small team environment over a larger team. Many of these tools are seen as bulky and that the resources given to implement these processes will not be within an acceptable resource loss for the benefit gained.

ACKNOWLEDGEMENTS

Thank you to Robert Gross for working with me at Magnimbus LLC., and helping me to gather and understand the data used as the Magnimbus case study.

REFERENCES

1. Build Automation http://en.wikipedia.org/wiki/Build_automation
2. Chapter 9 – Setting Up Scheduled Builds with Team Build http://msdn.microsoft.com/en-us/library/bb668959.aspx
3. CruiseControl.NET http://www.cruisecontrolnet.org/
4. Selenium http://docs.seleniumhq.org/
5. TeamCity http://www.jetbrains.com/teamcity/
6. Test Automation http://en.wikipedia.org/wiki/Test_automation
7. Visual Studio Test Professional http://www.microsoft.com/visualstudio/eng/products/visual-studio-test-professional-2012