Adopting agile methodologies and best practices for validating large legacy application base

Radhika S (Test Analyst)
Honeywell Technology Solutions Lab

Abstract

In the present technology era, the technology trend is changing every day along with the increasing customer base for existing legacy applications. It requires 60% of IT budget [2]for maintaining and supporting legacy applications developed about 20 years ago. It is worth spending as these services have millions of customers connected to them. These services are stable, and therefore can be migrated to newer technologies and methodologies. It is always a challenge for large product based companies to maintain and enhance these legacy applications, mainly when the application is in production environment, and accessed by the larger customer base. Adding new features to the existing system and performing a technology refresh in an incremental way is tricky and needs effective continuous deployment and testing methodologies. However, enhancing these application bases by using new techniques such as following agile methodology, modifying existing code base to bring rich user experience, and adding new features without impacting the other

modules with ongoing customer usage is always a challenge.

This paper proposes some of the best practices for adapting the agile methodology for continuous development and deployment, planning and performing testing, and managing the regression scope based on impact analysis. Additionally, it provides some of the other best practices such as using toolsets which are used for achieving the development, defining testing scope, handling change requests, and maintaining traceability with legacy features. The paper highlights the daily challenges faced while following the above process as we still have scope for improvements.

Introduction

What is a Legacy system? Any system that has been inherited from earlier versions or an application built with a very old technology or an application that was built few years ago say 15 to 20 years ago, may be termed as a Legacy System.

In this paper, I am going to discuss of the live legacy service that we worked on to stabilize the system, to migrate it to new technology, the best practices we followed, adapting to agile methodology, testing and validation.

System background:

The service I am talking about is a vast system consisting of 9 different modules and is used by around 56 utility companies around the globe. This system mainly deals in maintaining the huge database of the customers of the utility companies. Like we have BESCOM (Bangalore Electricity Supply Company) at Bangalore, who distributes electricity to Bangalore city and we the users are its customers, likewise there are several utility companies in the US whose customers are our end users. We do reporting, rebate, demand response to these companies.

This service was built and been maintained for 16 years and has undergone a technical transition and handed over to Honeywell. Understanding such legacy services without any documentation and having the team worked on it, onsite, was a real challenge. There was no process in place and no testing team formed. Still the team could achieve the stability and gained complete control on the system in a span of 2 years. How could we achieve this with some best practices is what is in my next chapter.

Best Practices:

1. Firstly the entire team was formed here in Bangalore (Since this project was taken over from an US organization by Honeywell) based on the expertise on each technology. There was need of people with different technology like .net, power builder, WCF, etc. So we set up a team with such expertise and then moved on bringing up collaboration with the onsite team. We planned knowledge transfer sessions for each module like reports, power builder and web module on daily basis. We planned these in such a way that only the people with the appropriate skills would attend these different sessions. These KT(Knowledge Transfer) sessions were recorded for further usage for any new member in the team or to refer offline. Then with these KT sessions going we identified a tool for tracking our efforts spent on each of the task we are doing.
2. We selected JIRA as our tracking tool as this service we were working on was about solving the tickets on day to day basis and deploying the code to LIVE. Once we had this tool in place we started dividing our team in such a

way that we made one person responsible for each module or divided the team based on the utility companies we serve. This way when the JIRA tickets would flow down from the account managers (these are the people who directly talk to the users /customers to gather requirements) to the team offshore, it became easy to assign these tickets back to the engineers who actually would work on it to solve the issue.

3. Now that the team was formed and the tickets were solved on day to day basis, we observed that some requirements would not be clear to the engineer and so the issue used to go back and forth and take several days to move to the production. This is when we thought that there should be a testing team as well who would understand the requirements well before and test it appropriately thereby reducing the turn overtime.

4. As I earlier said that this was a very vast system, it involves different databases for every utility company. Therefore the number of set-ups to be done on an individual machine is also more. Any new member joining the team would face lot of difficulties in making the set-ups and the other member would spend much of his time helping the latter. Therefore

we made several documents which clearly show how to make each and every setup. A snapshot of the various documents created is shown in Fig(a)[3].

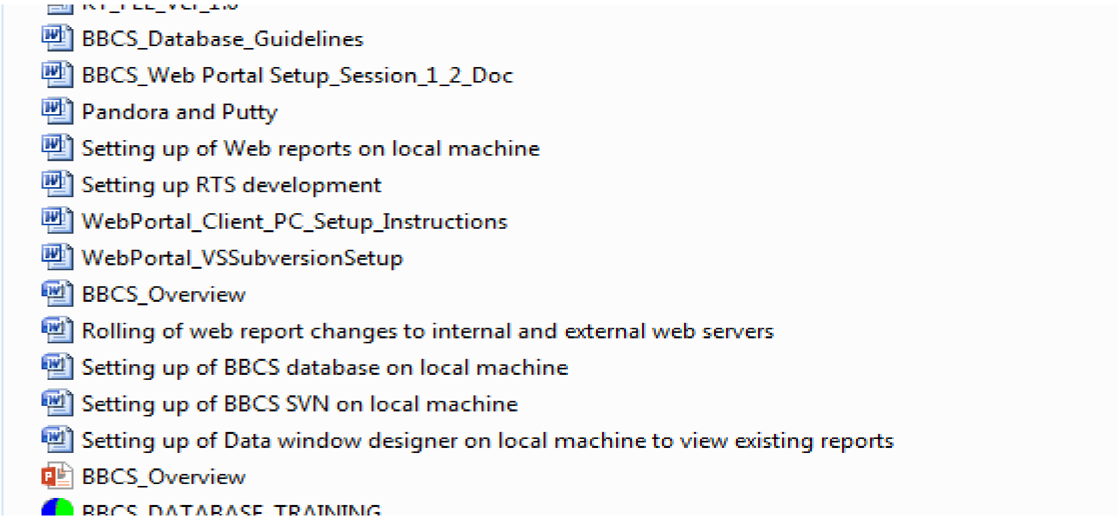


Fig: (a)

5. Next coming to the way we were deploying changes, is that every developer had access to the production. As and when the ticket was resolved, the changes were deployed to production by the individual. This was because we did not have any process in place. This called for the risk of overwriting the changes deployed by another person, if the person had also worked on the same procedure/package.

The best solution we found here is to withdraw access from everyone and give access only to the tester and the project lead, with the idea that when the tester has completed the testing, she/he is responsible of ensuring that the same changes are deployed to production. Here it is also ensured that the tickets that were not verified by the tester are not going to be deployed to production.

Process:

The potential benefits [1] that an organization can realize by using agile methods are:

- Products that better meet user needs – this is achieved through frequent interactions with users and increased adaptability to changes based on these interactions.
- Visibility into progress – short duration Sprints and delivery of completed work enable the project team and stakeholders to measure actual working software and velocity.
- Ability to release on demand – because 'completed work' is delivered at the end of each Sprint, the result is a potentially releasable product.

Due to the above 3 major benefits of Agile, we chose agile process for this service. Since this project is based on working on every day tickets and deploying the changes daily, agile could best fit here. But to adopt Agile it is required that you make the result available for the user earlier than expected. What we did to achieve this is explained again as best practices below.

Best Practices:

1. There was one problem that we could not create a build here at our end, as the control was with the onsite time and coz of the time difference there would be a delay of a day or two to get the changes to LIVE.

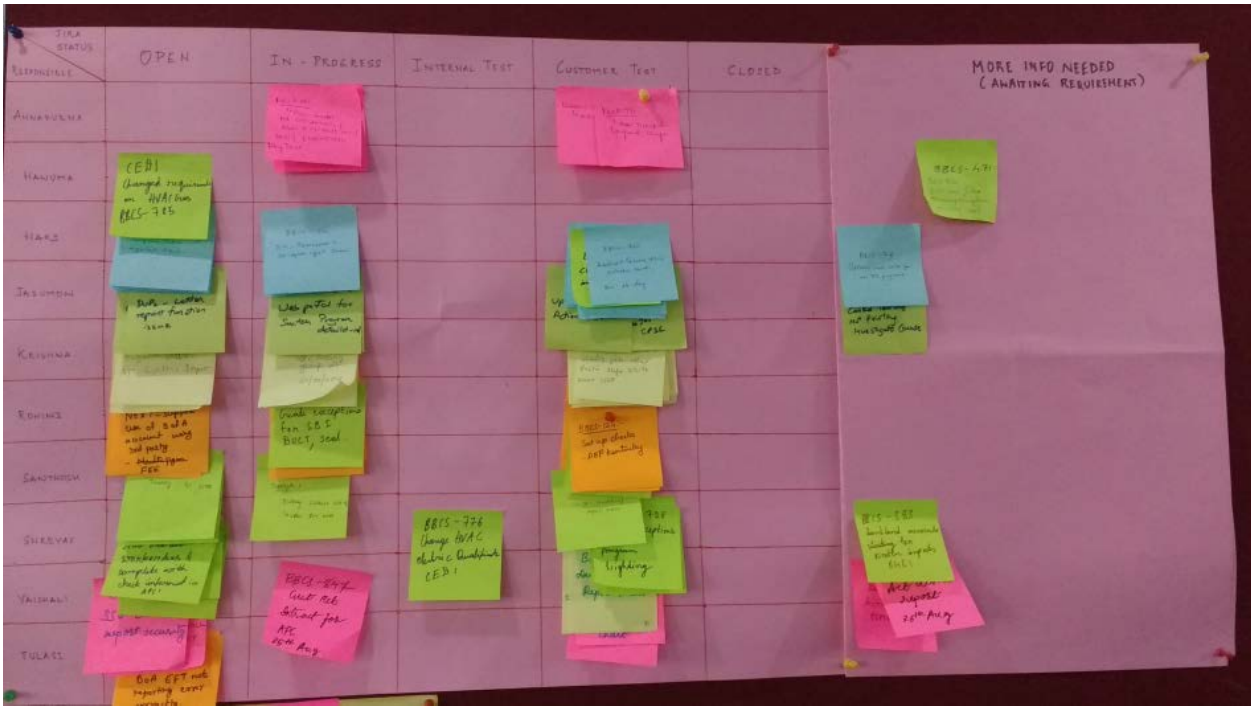
And one more issue was that we used to copy the code to a folder on the server and during the build process, this code would be picked up. This looked like a very haphazard way of deploying the code and had many security threats, overwrite of the code, etc. Therefore we then came up with SVN (Subversion) repository where the entire team both the onsite and offshore would check in their code and from here the code would be picked up by the build.

With the SVN in place, we set up our build environment here to make sure the build is created the same day and the changes are posted to LIVE for the users.

2. There was one more problem of tracking the JIRA tickets progress, which was becoming difficult day by day. We usually had tracking meeting on the start of the week in which the estimates were provided by each individual and the manger assigns the due date in the JIRA tool.

But when JIRA's were piling up, having meeting only once a week was making it go out of track of the JIRA as some were not meeting the due dates and some were awaiting for more information from the account managers.

So instead of having multiple meetings to track JIRA tickets, we started the scrum meeting with Agile in place. What we do in this meeting is explained below with the Fig (b)[5] which is self explanatory.



Fig(b)

So this picture has the team members in the first column and the other columns show the various status of a ticket in the JIRA tool. When people come to this meeting they put their post-its in appropriate column for the ticket they are working on. The benefits of this board are as below:

1. This board helps in understanding which ticket is in what status.
2. It shows how much each person is loaded.
3. It also helps in knowing where the ticket is taking much time, is it in testing or customer test or development.
4. Helps the manager for better tracking.

The More Information section is for those tickets for which the requirements are not clear, or if any more information is required on the ticket from the account managers. This Scrum meeting is something we carry out on daily basis for 30minutes.

3. Next coming to the weekly data to understand how many tickets are still open, how many were completed and how many were re-opened from the testing team in case of defects, we circulate weekly metrics on every Monday. The fig(c) [4] below shows the weekly metrics. The different columns are explained below:

i: *Completed*: Shows the number of JIRA's that were closed in the last week.

ii: *Req. Defects*: Shows any defect in the requirement.

iii:*QA Defect*: This shows the number of defects raised by the testing team before sending it for

iv: UAT defects: This shows the number of defects that were reported by the users during UAT.

Week Metrics - August 15 - August 19									
Developer	Completed	Req. Defects	QA Defects	Total QA Defects	UAT Defects	Re-Opened	Delayed	Eng. Defects	Eng. Success
Emp 1	4	0	0	0	0	0	0	0	1
Emp2	2	1	1	2	0	0	0	0	1
Emp 3	5	1	1	1	0	0	0	0	1
Emp 4	1	0	0	0	0	0	0	0	1
Emp 5	1	0	0	0	0	0	0	0	1
Emp 6	5	0	0	0	0	0	0	0	1
Emp 7	4	1	0	0	0	0	0	0	1
Total	22	3	2	3	0	0	0	0	

Fig(c)

Total Open Backlogs - ACS JIRA						
Assignee	Open	In Progress	Verified	Accepted	Completed	In Verification
Emp 1	1	0	0	0	0	0
Emp2	1	1	1	0	1	2
Emp 3	0	1	1	0	0	0
Emp 4	1	0	0	0	1	3
Emp 5	3	0	1	0	1	0
Emp 6	2	0	1	0	0	1
Emp 7	4	1	0	0	0	0
Total	12	3	4	0	3	6

Fig (d)

v: *Re-Opened*: Shows the number of tickets that were re-opened as there were defects in the JIRA.

vi: *Eng.Defects*: Shows the number of tickets that were re-opened after they were deployed to LIVE.

The next figure fig(d) shows the number of JIRA_id's (tickets)that are still open from last week and carried forward to the next week.

i: *Open*: No work was done on the JIRA.

ii:*In Progress*: The developer has started working on the JIRA.

iii:*Resolved*: The work on the JIRA is completed and is ready for testing by the internal testing team.

iv:*In-Verification*: Shows the number of JIRA tickets that are on UAT.

4. As we moved forward we realized that the code we are checking in is not reviewed at all as we work on tickets. But we could find many issues coming up and thought we should start reviewing the code. We then introduced the tool "CRUCIBLE" to start the code review. This helped in minimizing the defects and got absorbed as a good practice in the team.
5. Since this is a legacy system, the application is built on the power builder technology. When any ticket raised would need a change in the

UI then there would be a file to be deployed on the server. For this after the testing was completed, we would email this file to someone who would later post it on the server. This lead to one issue, that is, the DB changes would be first compiled on the server and till then the UI files were not loaded on to the server so this created lot of usability issues to the user.

And this is when we decided to create the power builder builds also at our end for the change we make for the tickets assigned to the team at Bangalore. With this build creation here, we achieved a lot of productivity savings as well as the build would be available on the same day unlike the earlier process which needs one full day for the file to go into the build and then be available at the server.

6. One another best practice we are following for the web module code is that we are maintaining a table called WHITELIST. This table contains all the procedures of a program. Whenever any procedure is executed, it will

first hit this table to check if that procedure is present. If it is present it will be executed else will fail. So therefore, whoever creates a new procedure are supposed to make an entry into this table. This helps to prevent/avoid SQL injection.

Only few members in the team are given access to this table and the new procedures are reviewed by them before inserting it into the table.

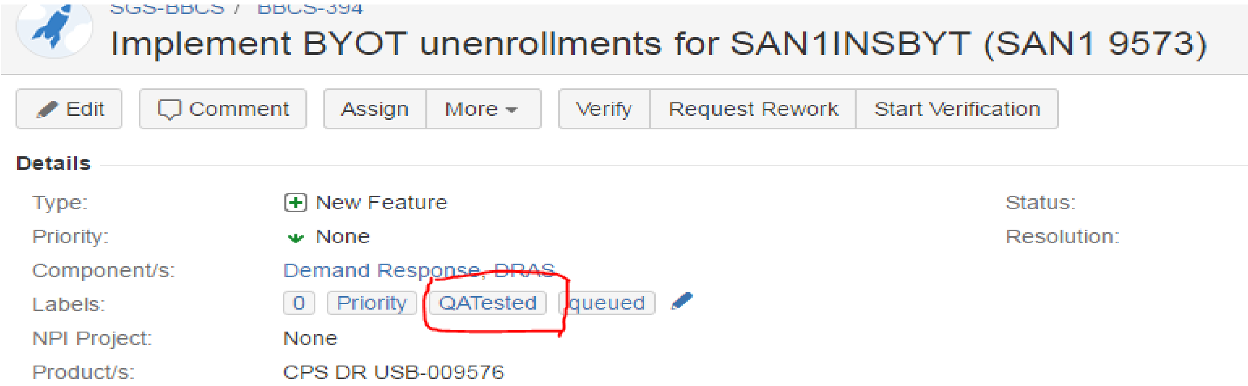
Validation:

Coming to the testing team, they work on every JIRA ticket that had to be tested and validated using SQL queries, manual testing or by automating the scripts.

If any JIRA required repeated testing then we either automate those test cases and based on the impact analysis supplied by the developer, we cover only the required test cases.

When a ticket was for a new program altogether, then we wrote test cases for each and maintained traceability to the requirements as a conventional practice. We entered efforts in a tool called **Contour** where the test metrics could be captured very thoroughly.

Once the internal testing is done and the JIRA's are posted to UAT and then closed, it sometimes happens that there would be some defect that got leaked from the internal testing we are doing. In such cases we were unable to make the tester accountable as we do not know if that JIRA was tested (as we would not test some data fixes tickets). To differentiate which JIRA's were tested and which were not, we created a new label called "QATested" as shown in fig(e) below. Any JIRA with this label means that it was internally tested before posting it to UAT. With this, the tester held all accountability if any ticket would get re-opened. He/she is supposed to do a RCA (Root Cause Analysis) on the issue that was re-opened and then come up with actions and learning's. These learning's would again be documented and checked in SVN for further use.



Fig(e)

When we initially started with this program, there were almost on average 50 tickets logged in weekly. This is only the defects I am talking about and not the enhancement/improvements. Below is the trend fig (f) for one of the module, which shows the issues that were logged on month on month and how the defects have slowly reduced.

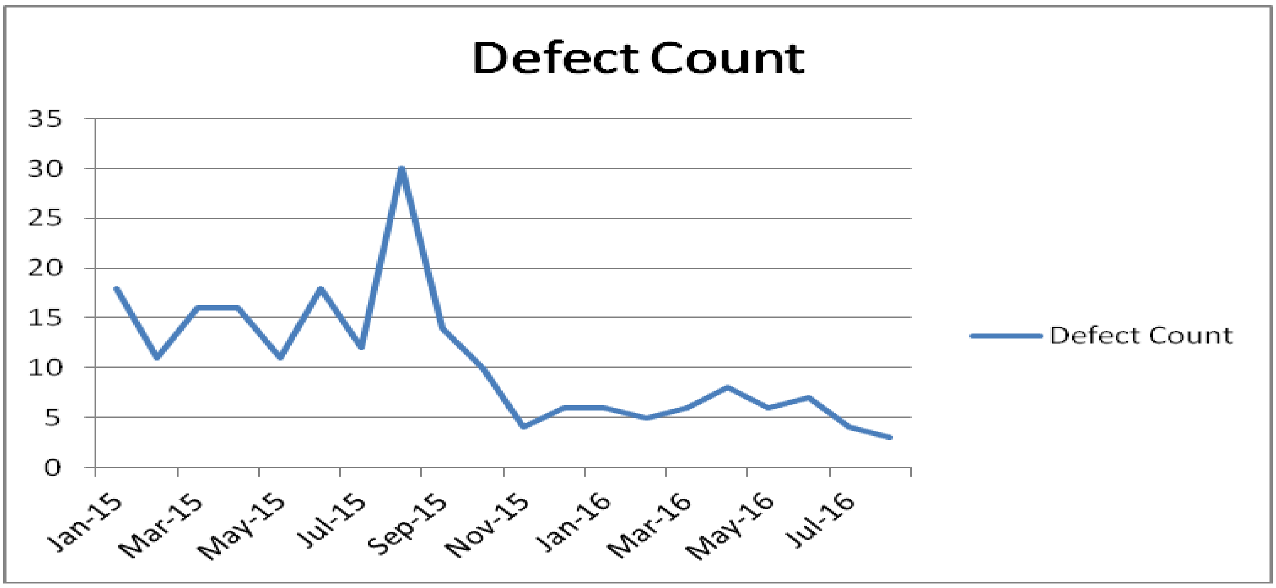


Fig (f)

We could achieve this by:

1. Interacting with the users directly or with the account managers, understand the requirements clearly look at the issue holistically how it is going to be used by the user
2. Having the team visit the team onsite to understand the actual scenario of work.
3. Having a process set to give the impacted areas with each and every fix made.(Impact Analysis Document)
4. There is a UAT (User Accepting Testing) done by the users and only after the changes are approved, they are moved to live. So chances of defects are less.

Conclusion:

This was a project which works on ticket and therefore the best practices shared may be best suited for such projects in any organization. Nevertheless some of them can be applied to any of the project whatever may be the process followed.

The key challenge is to decide on the regression test cases based on the impact analysis.

For example: IF there are 10 test cases in total and the regression test requires any 4 of them to be re-executed, there should he an automation scrip which will decide on which 4 have to be executed based on the impact analysis document or any other input factors. This is a topic open to be discussed in any forum.

References & Appendix

[1].https://corpapp.honeywell.com/sites/Exponent/V8_I2/Pages/Agile-Benefits-and-Risks.aspx

[2].http://www.netreach.com/SiteData/docs/NetReachLe/0145de4c6c210abb/NetReach-Legacy-Application-Modernization.pdf

[3].https://acssvn.honeywell.com/HBS/SGS/BBCS/trunk/Documents . This is our organization's SVN internal link.

[4]. From our organization's Internal Mailbox.

[5]. Honeywell's Scrum board from Smart Grid Solutions Business.

Author Biography

The author Radhika S is working at Honeywell technology solutions as a Test Analyst and holding experience of 6 years in functional and non functional testing. She is a B-tech graduate in Instrumentation Technology from RYMEC College, Bellary, Karnataka.

She has worked on projects which involved system testing, software testing and performance and capacity testing as well. She holds good knowledge in writing exhaustive test cases and writing test plan. She has been awarded as a best tester several times and has achieved quality excellence award for an energy program she worked on.

She is involved in innovation as well and holds 3 ideas in her name.

THANK YOU!