Adopting Agile for Enterprise Software

Peter Melko
CA Technologies
100 Staples Drive
Framingham, MA 01702
1-508-628-8012
Peter.Melko@ca.com

Jason Breen
CA Technologies
100 Staples Drive
Framingham, MA 01702
1-508-628-8457
Jason.Breen@ca.com

Joseph Bedell
CA Technologies
100 Staples Drive
Framingham, MA 01702
1-508-628-8362
Joseph.Bedell@ca.com

ABSTRACT
In this paper, we describe the Agile Methodologies and our adaption of Scrum when developing software for CA IDMS. We will highlight the benefits we are seeing using Agile and discuss some of the hurdles that still have to be overcome.

General Terms
Human Factors

Keywords
Agile Methodologies, Scrum

1. What is Agile and Scrum?

Agile is a software development methodology based on four core principles defined in the Agile Manifesto. The Agile Manifesto states,

Individuals and Interaction over Process and Tools

Working Software over Comprehensive Documentation

Customer Collaboration over Contract Negotiation

Responding to Change over Following a Plan

That is, while there is value in the items on the right, we value the items on the left more [1]. Agile is all about taking what is most productive and enjoyable about developing software and valuing it over anything else.

The principles that are valued by Agile are all interrelated. An Agile team must be able to communicate with each other, as well as talk with customers to review features as they are developed. Teams try not to develop features that no one will use and must be able to change as customers provide feedback. Agile bases itself on trying to provide value to customers as quickly as possible while also insuring utmost quality of work.

Scrum is a framework that embodies the Agile principles. Scrum is simply a set of practices that follow the Agile principles and allows the development teams to have a set of processes to guide them. The core practices of Scrum are short focused iterations often called sprints where a team incrementally creates software that is potentially shippable at the end of the iteration. Each sprint is a time-boxed session where a team completes stories.

A story is a customer benefit that is broken down into easily consumable features that can be completed in a single iteration that will give the customer some value. An example of this is,

As a site user, I need to sign in to the website to see my subscriptions.

The benefit is only seeing the information that is relevant to the user. From this user benefit, the team could create a story for the user to sign-in. As a site user, I need to provide my username and password to access the site so that I only see my content. To complete a story the team has to complete all aspects of software development; coding, testing, documentation and integration.

Scrum teams are small self-organizing teams that are both cross-functional and contain members who can perform in any role. Since the whole goal of Agile and Scrum is to deliver quality working software as quickly as possible members of the Scrum team are often called on to work outside their primary function. Depending on the makeup of the team, coders may end up writing tests and testers could write the code for a given story. Not only does this decrease the impact of people taking vacation or being sick during a sprint, but it also helps coders and testers work together more effectively.

1.1 What Scrum is not

Scrum is not just a new name for smaller scale waterfall sometimes jokingly referred to as Scrumfall. In Scrum you want to minimize handoffs and try to do as much work on a given feature concurrently. A tester should be able to start developing tests as the coder writes the code and unit tests. In Scrum you do not have a coding iteration, a testing iteration and an integration iteration.

Although the Agile principles states, responding to change over following a plan; in Scrum once the team decides on the stories that will be worked on in a sprint scope is locked. Once the team begins working on features the project stakeholders cannot change scope on the team. If the stakeholders want to change scope they can do so after the current sprint is completed. Teams define their own sprint length, but generally two to four weeks is a standard sprint length, this allows changes to be made sooner than later while not interrupting teams while they work.

2. CA's Agile Transformation

CA started to transition the IDMS team to use Scrum about five years ago. The first few years of Agile at CA used a Scrum framework very different to how we do things today. The IDMS team has been actively improving their processes over the past five years and there is still a lot of improvement to be had. Scrum is about continuous improvement where anything can be called into question to make the team more efficient. The whole Mainframe business unit is now using Scrum for development with teams in various states of adaption.

2.1 Incremental Releases

Agile requires teams to get functioning software in the hands of customers as soon as possible. CA's old policy of creating large monolithic releases every couple of years is squarely against that Agile principle. To get software to customers faster, IDMS has adapted an incremental release schedule for our latest release. An incremental release is time-boxed for every few months and at the release boundary any features that are ready for customers get packaged up into a program temporary fix or PTF and are then released for customers to download and install at their sites. Packaging the new features into a PTF allows the customers to get the new software faster. Anytime new install media is cut thorough testing is required to make sure the installations still work.

At the start of a new release cycle the IDMS team creates a base release with install media and then every few months following the team will release a new PTF with customer features. Each incremental release is a prerequisite for the following incremental PTF. Once the IDMS team has released multiple incremental releases over the course of a few years we will then bundle up the base media, all incremental release PTFs and any support PTFs to create a new "GA Complete" install media. Once the "GA Complete" media is created the IDMS team will then start the process over again for the next release by cloning the current "GA Complete" media into the next release's base media.

There will always be customers that for some reason cannot use the incremental release strategy at their sites. For this subset of customers the "GA Complete" media would essentially be the same as the old monolithic releases that CA used to deliver. The big advantage to incremental releases is for customers who want a particular new feature quicker. They have the option to sign-up for the incremental releases and get their feature sooner than every two years. The customers who participate in the incremental release program have a larger say in the enhancements the IDMS team creates and how those enhancements work.

2.2 Customer Collaboration

Customer collaboration is a key facet to making Agile work. Without engaged and excited customers Agile development would not be possible. The teams work with customers in a few ways; first the team reviews their work with customers after each sprint. The sprint review gives the team a chance to show off and demonstrate working software to customers for feedback. Any feedback generated is noted and added into future sprints if the team thinks it is important enough to work on.

Once a feature gets to a certain point the IDMS team creates a test PTF to allow customers to try out the software on their systems. The intention of the test PTF is to have GA quality software validated by customers before the feature is released in an incremental release. In terms of waterfall, the test PTF is a mini beta. The tests against a feature may work perfectly on our systems but once the code gets exposed to differing setups and to the millions of transactions that we cannot accurately simulate problems may expose themselves.

With the adaption of Agile, CA has placed a new focus on giving customers only the features they want as there is no sense developing a feature no one will use. A Scrum role called a Product Owner is responsible for meeting with customers to understand their wants and needs and then to relate those to the development teams. Once the Product Owner has a customer request they enlist the team to help create the stories that will make up the feature. The Product Owner would then confirm with the customer that the team has the feature defined correctly before work can begin on the feature.

CA also has an online portal called the CA Communities where customers and developers interact. The Communities site allows both customers and developers to propose product enhancements. Proposals with enough customer interest get added to the product backlog and then are prioritized against all the other enhancement requests.

2.3 Project Planning

The Agile manifesto states, we value working software over comprehensive documentation. This is not to be taken out of context to mean that customers do not need documentation, rather that extensive design documents meticulously detailing the entire feature before coding is avoided. Agile development is about emergent design, to plan out just enough to get the current feature working before you design the next piece of functionality. This is done because some features originally planned may never get developed if customers are not interested. The IDMS team no longer creates detailed design documents in favor of interacting with one another to achieve the same results. The teams still create small documents detailing changes they make to modules for future reference for support cases.

In Agile, the product architect is still required they just work in a different manner then they used to. The architect can work with the Product Owner and customers to help clarify requirements for enhancement requests. The architect will not completely plan out the feature but will help determine if the enhancement is feasible. The architect will also work with the team to clarify requirements when grooming stories. The architect should generally work one to two sprints ahead of the product team so they will know if there are additional dependencies the team needs to consider.

2.4 Automated Testing

Scrum expects software to be of outstanding quality as well as complete at the end of a sprint. The IDMS team has a zero defect policy and will not close stories that contain new bugs. To achieve these goals automated testing has taken on a whole new level of importance. In the waterfall world projects were planned with months of testing after all coding was completed. In Scrum, teams do not have that luxury as code is being created all the time. Tests have to be running constantly to make sure the many different features are running at GA quality.

The IDMS team has made great strides in improving our testing. Every night the IDMS team runs approximately 1500 tests against the supported releases. These tests are used to confirm that any new PTFs applied to the code base do not break working code. Without the nightly automated testing the IDMS team would not know if new fixes were actually creating more problems. These same 1500 regression tests are also run against new features to verify backwards compatibility.

At the end of a release cycle the IDMS team used to spend two to three months performing QA testing of the soon to be GA software. The QA tests would include all of the automated tests, manual tests and when there was install media a variety of installation test cases. Now that the release cycle for incremental releases is every two to three months taking just as long to test as develop the software was not an option. The improvements to the automated test infrastructure enable the team to complete the testing within one week. A future goal for the IDMS team is to reduce that time even further to a few days. This goal will be accomplished by further building up the automated test framework and reducing the reliance on manual testing.

2.5 Technical Debt

Technical debt is a term coined by Ward Cunningham to describe anything the team does today to accomplish their goal that will have to be revised later. The easiest way to think of technical debt is through an analogy. Technical debt is a loan the team takes out to accomplish their work faster in the short term. Sometimes you have to take out a loan to make value possible but you must be able to pay down your technical debt or else the interest grows to the point where the interest payment is more than what you bring in.

In other words technical debt is the cost of doing business; it is anything the team has to do that the customer may not care about the team completing. Some technical debt the IDMS team has to deal with includes automating legacy tests, team education, cross training and certification projects amongst other things. Since technical debt is created by the team, the teams are responsible for keeping track of the technical debt backlog and slowly whittling away the technical debt IDMS has accumulated over the years.

Each week, members of the Scrum teams meet to prepare stories on the technical debt backlog. The technical debt backlog contains stories similar to the user stories described above, but they have the express intent of paying down the technical debt interest IDMS has accrued. The team spends time creating and refactoring technical debt stories as well as grooming the highest priority stories for the next sprints. Each sprint the IDMS teams take a percentage of their capacity away from developing and supporting the product to work on technical debt.

2.5.1 Types of Technical Debt

Below we discuss in detail some of the major technical debt areas the IDMS team works on.

2.5.1.1 Refactoring Code

As the code base increases in size the complexity of the software generally increases as well. Over time fixes being added to the code might solve the immediate problem, but may make the code more difficult to read. Refactoring is the process of changing the internal structure of the code while preserving the same behavior. Refactoring is very important in older products because modules written 10-20 years ago may not be taking advantage of the latest hardware and operating system improvements. A team may also choose to refactor because there is no expert in that section of the product. Refactoring can occur across the whole module or it can be done sections at a time as a team adds new enhancements to those modules. One of the major impediments to refactoring is having enough testing to verify functionality does not change. Not only should there be adequate acceptance tests, but there should also be unit level tests for any functionality that will get updated.

Refactoring is usually reserved for modules that we’re making updates to. If we’re making a lot of changes, sometimes it makes sense to update the overall structure of the module as it wasn’t originally designed for the new functionality. This allows us to update the module to more modern standards, get new experience with what the module does, and doesn’t take much more time than the original work. We tend to avoid refactoring modules that are stable and not being changed just for the sake of refactoring them.

2.5.1.2 Fixing and Automating Legacy Tests

Any product that has been around as long as IDMS will have a massive test suite, the IDMS team has about 12,000 tests in our test repository. The simple act of maintaining all of those tests takes time. Any features that change behavior will cause tests to fail. Along with fixing tests that fail because of behavior changes, a large portion of tests in our repository are either manual or automated but do not meet our current standards. The effort to update tests to make them run nightly takes a long time but the increase in test code coverage allows the team to develop knowing the testing should identify problems before customers do.

2.5.1.3 Team Education

Any effort to train team members on the product, teach them how to work in a different role or classes to expand their technical knowledge is considered technical debt. The IDMS team counts all of these activities as technical debt since it does not bear any direct benefit to the customer.

2.5.1.4 Certifications

Every new release of the mainframe operating system z/OS or any of the other non-IDMS components IDMS interacts with requires the team to validate IDMS functions correctly. This is very important for the team to complete since customers have come to expect day one support from IDMS.

2.5.1.5 Process Improvements

Process improvement is one of the largest buckets of technical debt the IDMS team works on. Process improvements are anything that makes developing and supporting the software easier and faster. The team spends a lot of our technical debt time implementing enhancements to our current processes as well as researching new avenues for improvement. Our current process improvement is decreasing the time it takes to run all of our testing. As mentioned above the team’s regression testing runs overnight, we are in the process of implementing a framework to decrease the testing time dramatically.

2.6 Pair Programming

Team members interacting with each other is key for Scrum to work. One way the IDMS team accomplishes this is through pair programming. Pair programming is a technique where a couple of team members work together at one PC. One person is the driver who writes the code and the other is the observer who reviews each line of code as it is written. Pair programming works very well in our environment as it is a method we use to transfer knowledge. Oftentimes a senior developer and a junior developer will get together to work on the new features. The junior developer will drive and the senior developer will observe the code changes and offer advice and insight. Alternatively, two junior developers will get together to help each other learn the module while making the changes together. Pair programming is a very powerful tool for training because you are getting individualized attention while still delivering customer value.

Pair programming also affects software quality. Overall there are fewer defects introduced when pair programming because two developers are looking at each change to the software. Not only are there less defects, but the flow and structure of code can be better when pair programming. If a QA engineer pairs up with a developer, the QA engineer can write the acceptance and new features tests as the developer writes the code. Being in such a tight feedback loop allows the QA engineer to write better more directed tests and gives the developer a target to which he can make changes as well as providing the developer a more holistic view of the feature being developed.

2.7 Merging Development with Level 2 Support

In an effort to streamline our processes the IDMS team has recently combined the level two support team with the development teams. Now IDMS has four Scrum teams instead of two and each team focuses on one area of IDMS. Combining the development and support teams fits Scrum practices because we want each team to have collective ownership of what they are working on. When a new bug is found, developers are brought into the loop much faster compared to before when they would only be involved in a support case if the support team could not create a fix. Combined teams also increase the teams' opportunity for customer collaboration. The development teams used to be sheltered from customer interaction. Now that they will be responding to support cases they can start to build relationships with customers and can see how the software is used.

Performing the support role allows everyone to develop T shaped skills. The teams get specialized knowledge in their product areas while still being able to perform generally across the whole product. Support cases are usually tricky to diagnose and require a deep knowledge of the problematic modules.

Each new change is not without its own challenges. The IDMS team is having difficulty prioritizing support and development work together. IDMS is a business critical application for many businesses, when customers find defects in IDMS it can easily cost the customer clients and revenue. Because of this, IDMS issues are higher priority and the issues cannot wait for the team to prioritize them with innovation stories and then wait for them to be considered in the next sprint. Another problem the teams are facing is the irregular intervals support issues are created. Since we do not know how many issues will be created in a given sprint, it becomes difficult to plan a team's capacity. We have tried using the past sprint as a starting place in the next sprint, and found decent success with that so far.

3. Conclusion

Agile and Scrum have been welcome changes at CA. Although they are often associated with lightweight distributed products we have been able to adapt them to our product and environment. If a team is willing to put in the effort it's possible to perform Agile development on an Enterprise product and reap the rewards of Agile. Scrum has allowed our teams to take collective ownership of their products and feel empowered when planning new projects. At the same time, it has allowed us to get customers the features they want even faster. While it takes effort to get started, a team that commits to Agile and constantly tries to improve will find it a powerful framework to enable customer satisfaction.

4. REFERENCES

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunnigham, M. Fowler, J. Grenning, A. Hunt, R. Jeffries, J. Kern, B. Marick,

R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, "Manifesto for Agile Software Development," 2001. [Online]. Available: http://agilemanifesto.org/. [Accessed 14 April 2015].

[2] I. Goldstein, Scrum Shortcuts Without Cutting Corners, Boston: Addison-Wesley, 2014.

[3] M. Cohn, Agile Estimating and Planning, Boston, MA: Prentice Hall, 2006.

[4] M. Cohn, Succeeding With Agile, Boston: Addison-Wesley, 2010.