Five steps for agile transformation in manufacturing

Aurimas Adomavicius

President

Evidence indicates that agile software delivery results in a higher success rate for software projects as compared to waterfall delivery. The product (I'm going to substitute Product instead of Project going forward, the agile way) is of higher quality, Quality Assurance costs go down, and more scope can be shipped to market in a shorter period of time. In fact, we've seen a tenfold decrease in costs for delivery of identical software products for manufacturers when an agile approach and toolset were used. Why then do some companies insist on the slower waterfall process?

SOFTWARE DELIVERY SUCCESS – AGILE VS. WATERFALL

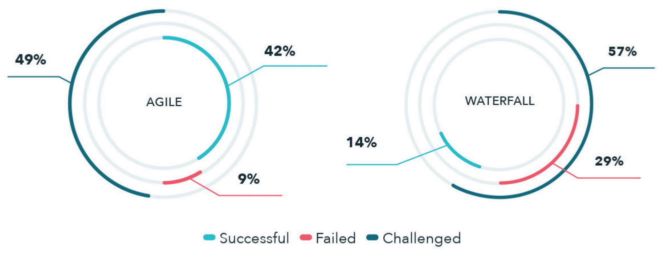


Image source: The Chaos Manifesto, The Standish Group, 2012

Blame the industry. Historically manufacturing has relied on establishing a predictable model for building a specific widget and then optimizing the process to a point where every single variable is accounted for. This results in predictable cost per widget manufactured, predictable demand for materials, distribution, and so on. Building custom software, however, is in many ways the complete opposite to building widgets. Waterfall (fixed price, fixed scope, fixed schedule) simply does not work. If a manufacturer embarks on a build vs. buy decision there is likely a good reason – the problem has not yet been solved to an adequate degree and there's a competitive advantage in using custom software. What we've come to discover, however, is that these organizations can get 80 percent of the value from Agile while changing only a fraction of their processes and practices, without, in any way, compromising on financial accountability. In fact, quite the opposite.

Following are a list of topics covered in this white paper:

Step 1: Reduce project (product) size

Step 2: Use lean requirements to jump start delivery

Step 3: Upgrade your tools to improve communication

Step 4: Invest in continuous integration and delivery

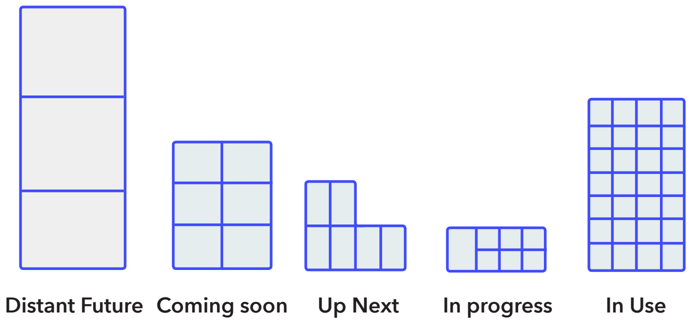
Step 5: Integrate Quality Assurance into your delivery team

Let's start with biting off a smaller piece of the apple to help with our new, Agile way of chewing.

STEP 1: REDUCE PROJECT (PRODUCT) SIZE

All large transformations start with small success stories. Starting on a large, complex project as a trial run for Agile, a process that is new to your organization, is probably not the best idea.

A divide and conquer approach when introducing Agile to the enterprise is preferred. We select an initiative from the product portfolio and distill it to the smallest functional modules, allowing for the team to iterate on sub-components of the overall product. This gives the cross-functional team the opportunity to familiarize themselves with the workflow, experience the rhythm of agile rituals, and build confidence in each other.



The team is encouraged to fail. Failing on a small piece of delivery, a sprint, endorses collaboration, helps learning from mistakes, and limits the financial impact since the scope is not significant. Long term, the team grows to understand that failure is a natural part of software delivery and that the real indication of success is shipping the right product for the users. In fact, embracing failure also aids organizations in adopting a more innovative mindset. Not all ideas are meant to succeed, but you're more likely to hit a jackpot if you allow your teams to experiment.

But, let's get back to project sizing. What's too small and what's too large for getting started with iterative delivery? In short, anything less than 1,500 man-hours is probably too small, and anything over 5,000 is risky. Here's how we get to those numbers:

A typical, small delivery team could include:

PRODUCT MANAGER - the individual responsible for building the right software for the users. The PM represents the business and establishes the minimal viable scope for the product to meet the customers' needs. In classic waterfall delivery, a subset of the PM's responsibilities is often handled by business analysts. The involvement of the PM in actual day-to-day delivery is limited, so let's say they will have a 25 percent utilization of total productive weekly hours (i.e., the PM will spend roughly eight productive hours, helping the team define what they're building on one product). These hours vary based on the size and complexity of the product managed.

TEAM LEAD - responsible for distributing work across the team, grooming the backlog with the PM, and establishing high-level technical architecture. Let's say that our team lead is fully allocated to this project and can spend 35 productive hours a week on this initiative. Some agile teams have dedicated Scrum masters, however, we've split this role between the Team Lead and the PM.

We may also need a **DESIGNER,** a quality assurance analyst (because we want to automate our testing throughout delivery), and an additional engineer for delivery bandwidth.

By the time we're finished building our cross-functional team we are looking at roughly four full-time employees. There are more members in the team, but some are only partially allocated and/or shared across several projects. A team of four burns approximately 140 hours a week, or 390 hours a sprint (2 weeks x 35 hours x 4 FTEs).

The benefits of Agile surface when we can iterate over the product, building small, functional pieces of the overall deliverable, testing with customers, and revising as we go. Let's give the team some room to fail and improve, setting our overall project length at six sprints. At 390 hours a sprint and 6 sprints, we can safely say that the smallest project that we can attempt to do with Agile is around 1,680 man-hours. It's fine to start with the estimate your team prepares for waterfall delivery and then attempt to ship it using Agile.

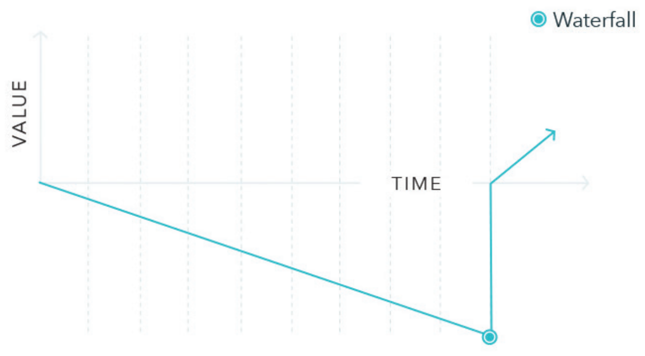
You're going to run into a lot of naysayers who will tell you that most enterprise projects are difficult to break down into smaller components. That's simply not true. Dig deep enough and each product, each technical implementation, can be distilled into sub-components that are perfect fits for iterative delivery

STEP 2: USE LEAN REQUIREMENTS TO JUMP START DELIVERY

Does this sound familiar?

A business analyst is assigned to a small-to-mid-size project and over a period of six months proceeds to create an extensive library of assets that includes the application design document (in Microsoft word), the design of web services, interfaces, and data architecture, a workflow schema, a user interface document, a high-level workflow document, and a detailed user journey that binds all of these assets together. During these six months, no business value has been created because the company is in no way closer to a functional product.

TIME TO VALUE - WATERFALL

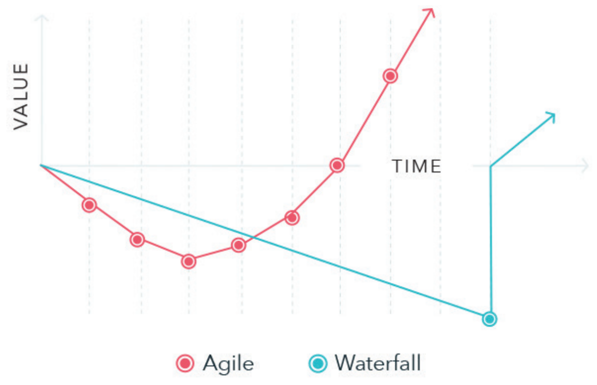


And yet the market doesn't sleep. Requirements start changing prior to the documentation being finished. Word and Excel documents are static—they do not provide support for versioning, change history and thus become difficult to maintain and keep current. Once finished, they're more than intimidating. Hundreds of pages of detailed charts, inaccessible or difficult for the team to digest. Lastly, all technical and user interface (UI) design decisions within the requirements documentation have been to-date made by a single analyst that operates without the benefit of a cross-functional team or the knowledge they bring to the table.

So, six months, little value, already outdated. Time to try lean requirements. Without diving into too much history you should know that the objective of lean is to maximize customer value while minimizing waste. It goes a way back, Toyota being the pioneer in the automotive space for transforming what Ford was doing with Model T in a new, flexible framework.

There are a couple of rather simple tools within the lean framework that are irreplaceable in our requirements toolkit when building new (or replacing old) software products.

TIME TO VALUE – AGILE VS. WATERFALL



Unlike classic requirement gathering, we start lean workshops as a group. Our primary objective is to arrive at a shared understanding of the deliverable as a team, while including stakeholders and customer needs in the exercise. During the workshop that can be as short as four hours to as long as three days, we establish personas, perform a story mapping exercise, draw user flow diagrams, and go through basic prioritization exercises for the captured scope. Let's look at each in detail.

User personas capture the nuances of users that will benefit from the software. We keep referencing our personas as we're adding functionality to the story map and think of how said functionality impacts their routines.

Story mapping is a collaborative process of capturing product requirements from the cross functional team in the workshop. The approach is simple: there are no bad ideas, and your idea must fit on a post-it note. Everyone in the room takes time writing down their thoughts and then adding them to a whiteboard. The PM meanwhile starts eliminating duplicates and grouping ideas into Epics - larger blocks of product functionality.

Once a story map has been established, the team can collaboratively evaluate the complexity of individual stories. Since each story is relatively simple (otherwise it can't fit on a Post-it note), t-shirt size estimation is used to evaluate the overall impact of each. For example, is a Facebook sign in a small, medium, or extra-large story? Stakeholders can also debate the sequencing of stories, influencing the delivery schedule and building a Release Schedule and Product Roadmap.

Lean requirements are accessible, create a shared understanding, and allow all members of the cross-functional team to contribute their ideas for the product (a team of 10 is better than doing it solo). It's also low cost, takes only a couple of days, and provides just enough information for the team to start moving and shipping functional software. A side effect of involving senior executives in workshops is that alignment and buy-in happens through participation. You no longer have a project derailment halfway through the project because your sponsor has committed to the scope and schedule in the workshop.

The beauty of Agile, or specifically dual-track Scrum, is that requirements are created just-in-time for delivery of the following sprint. The definition of acceptance criteria on each story is specified during backlog grooming and lives in Jira - a product management tool. Each time requirements or documentation changes, it can be easily updated in a centralized, living repository that everyone on the team has access to.

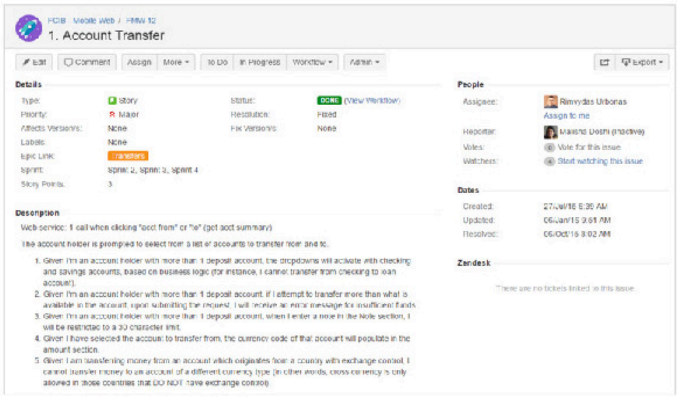
To summarize, lean requirements help us ship working software over comprehensive documentation, which is one of the cornerstones of the Agile manifesto.

STEP 3: UPGRADE YOUR TOOLS TO COORDINATE COMMUNICATION

WORST CASE SCENARIO: your project is managed in Microsoft Project, your scope is captured in static Word and Excel documents, your source code is stored in an instance of TFS, and your quality assurance team logs defects in an HP enterprise quality center. This is an ecosystem that provides zero transparency, no integration, and disconnected communication among teams. Hence one reason why projects cost twice as much and deliver half the value.

A BETTER SCENARIO: Take a look at Jira, the gold standard in the industry of software product delivery that helps team members stay connected and current. There are other tools, but none have the maturity and breadth of functionality offered by the Atlassian family of products (e.g., hipchat, confluence, etc).

User Story Acceptance Criteria



FCIB · Mobile Web / FMW-12

1. Account Transfer

Edit | Comment | Assign | More | To Do | In Progress | Workflow | Admin | Export

Details

Type:	Story	Status:	DONE (View Workflow)
Priority:	Major	Resolution:	Fixed
Affects Versions:	None	Fix Versions:	None
Labels:	None		
Epic Link:	Transfers		
Sprint:	Sprint 2, Sprint 3, Sprint 4		
Story Points:	3		

People

Assignee:	Rimvydas Urbonas Assign to me
Reporter:	Malisha Doshi (Inactive)
Votes:	0 Vote for this issue
Watchers:	4 Start watching this issue

Dates

Created:	27/Jul/16 5:39 AM
Updated:	06/Jan/15 9:51 AM
Resolved:	06/Oct/16 3:02 AM

Zendesk

There are no tickets linked to this issue.

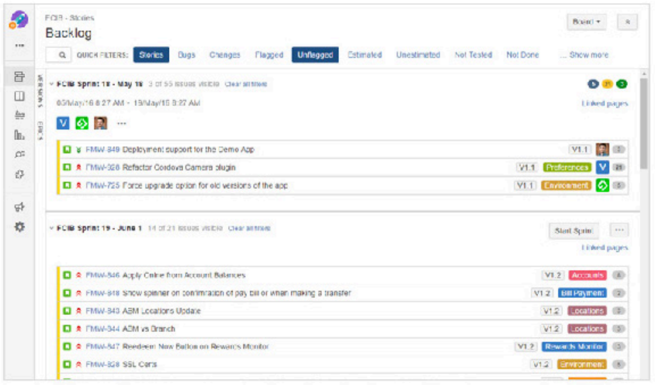
Description

Web service: 1 call when clicking "acct from" or "to" (get acct summary)

The account holder is prompted to select from a list of accounts to transfer from and to.

1. Given I'm an account holder with more than 1 deposit account, the dropdowns will activate with checking and savings accounts, based on business logic (for instance, I cannot transfer from checking to loan account).
2. Given I'm an account holder with more than 1 deposit account, if I attempt to transfer more than what is available in the account, upon submitting the request, I will receive an error message for insufficient funds.
3. Given I'm an account holder with more than 1 deposit account, when I enter a note in the Note section, I will be restricted to a 30 character limit.
4. Given I have selected the account to transfer from, the currency code of that account will populate in the amount section.
5. Given I am transferring money from an account which originates from a country with exchange control, I cannot transfer money to an account of a different currency type (in other words, cross currency is only allowed in those countries that DO NOT have exchange control)

Jira Backlog



FCIB - Stories

Backlog

QUICK FILTERS: Stories | Bugs | Changes | Flagged | Unflagged | Estimated | Unestimated | Not Tested | Not Done | ... Show more

FCIB Sprint 18 - May 18 · 3 of 55 issues visible · Clear all filters

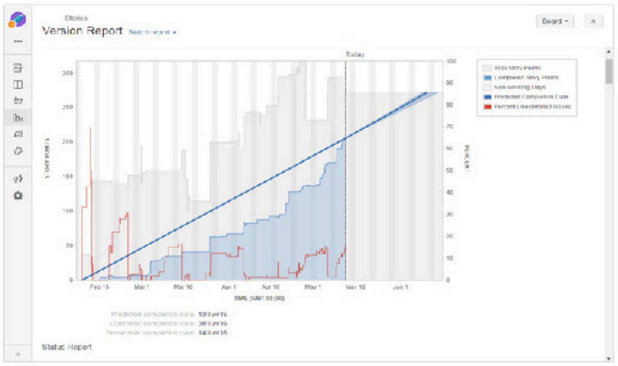
03/May/16 8:27 AM - 19/May/16 8:27 AM · Linked pages

FMW-849 Deployment support for the Demo App	V1.1		
FMW-328 Refactor Cordova Camera plugin	V1.1	Preferences	28
FMW-725 Force upgrade option for old versions of the app	V1.1	Environment	5

FCIB Sprint 19 - June 1 · 14 of 21 issues visible · Clear all filters · Start Sprint · Linked pages

FMW-846 Apply Online from Account Balances	V1.2	Accounts	8
FMW-818 Show spinner on confirmation of pay bill or when making a transfer	V1.2	Bill Payment	2
FMW-343 ABM Locations Update	V1.2	Locations	[illegible]
FMW-344 ABM vs Branch	V1.2	Locations	3
FMW-847 Redeem Now Button on Rewards Monitor	V1.2	Rewards Monitor	3
FMW-828 SSL Certs	V1.2	Environment	8

Jira Reporting



You can get started by licensing a cloud instance of Jira if you don't want to get your infrastructure folks involved and that's a great way to move lean. Once you're more comfortable and need a more granular, configurable instance you can opt for the enterprise package and set up a dedicated application within your own datacenter.

Regardless of the tools you use, the objective is to promote individual communication and interaction—another cornerstone of the Agile manifesto. Jira works because it gets out of its own way and allows teams to have meaningful conversations unburdened by busywork, documentation, and disconnected sources.

Through Jira, or another tool, the team can manage backlog, define user stories, capture acceptance criteria, upload and attach visual design assets, collaborate through comments, and monitor dependencies and blockers. Added benefits include a living audit trail of all activity within the project (e.g., how did the acceptance criteria change through time on this user story?), in-depth reporting on delivery, roadmap planning, and more.

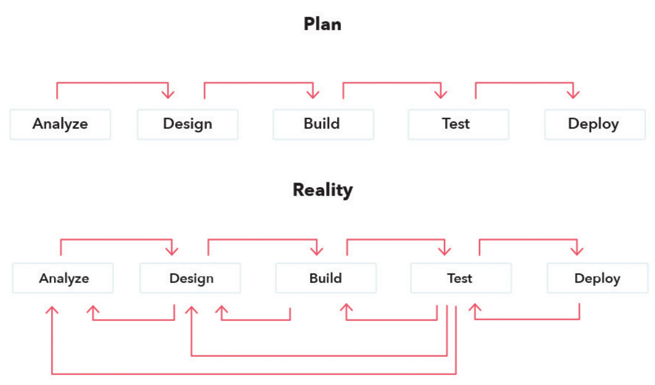
If your organization's Enterprise Delivery Framework team states that Excel documents are necessary to stay compliant, know that Jira can be customized to accept all of your custom fields that your EDF teams wants you to track. We recently went through an exercise with a Tier 1 bank, mapping its Excel-based user story document to a user story in Jira and had to make absolutely no compromises. In fact, the versioning and tracking in Jira was far superior to versioning a static spreadsheet meant to be used by accountants.

In this section, I've only skimmed the surface of what a shared toolset offers. There's reporting, transparency, delivery KPI's, and a ton of other metrics that help businesses make better, just-in-time decisions when building software.

STEP 4: INVEST IN CONTINUOUS INTEGRATION AND DELIVERY

More often than not we see very limited automation in how environments are set up, how source code is built, and how deployments are handled from environment to environment.

A sample scenario would look something like this: engineers are working on local machines and checking code into a source repository (TFS, Git, etc.). Each developer tests their code, builds the solution locally to make sure it runs as expected, ad infinitum until the project is finished and ready for testing. SIT/UAT phase kicks off and all hell breaks loose. Integrations don't work, defects come crushing down in endless Excel documents exported from the enterprise QA system, and timelines slip. Worst case, all projections of cost/timeline get thrown out the door because these problems were discovered once delivery was already finished (i.e., at the end of the project when a majority of the budget has been spent).

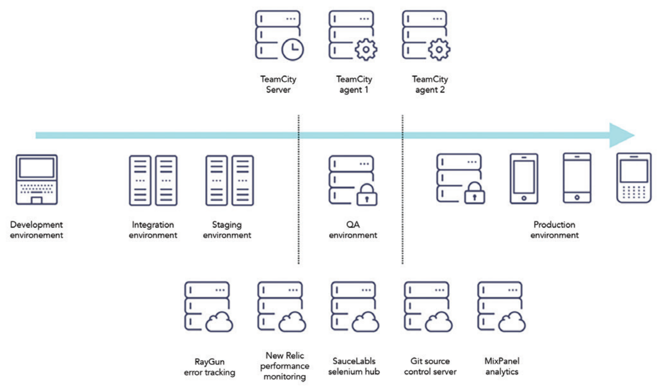


Continuous Integration (CI), a development practice and toolset that enables engineers to integrate their code into a shared repository on a daily basis, to the rescue. CI build servers run automated tests and build the product, allowing individual contributors to detect defects as soon source code is checked in.

For the sake of keeping this white paper shorter than Homer's Odyssey I will assume your applications have separated environments for development, staging, QA/UAT, and production. If not, invest in infrastructure prior to standing up a build server and implementing continuous integration. Hardware requirements for dev/staging are likely a fraction of your production instance, so costs should be reasonable.

A word of caution: setting up continuous integration involves infrastructure teams, hardware, and software licensing. It's important to note, however, that environment setup costs are very small when compared to lower QA costs, product quality, and increased delivery velocity. Let's look at a specific working example as a guideline for CI costs/configuration.

Two leading platforms are TeamCity (licensed) and Jenkins (Open source). In either scenario you will need to set up a build server, as well as agent servers for your end-points (e.g. web application, native mobile apps, etc.).



So what are the benefits?

1. **You avoid a lot of the conflicts when merging code** because everyone is using a single repository, committing, and building every day (and often several times a day).

2. **Deployment is automated.** Single click. You go to the TeamCity control panel, initiate deployment and watch as you production environment is updated to a copy of the application in your QA/UAT environment. Or isn't if there are build errors, but likely there aren't because you have been building every day, multiple times a day, across multiple machines, and running automated tests.

3. **Your delivery becomes predictable**. There are no last minute hot-fixes. No calls at night or on the weekends.

4. **Your infrastructure, support, and QA costs go down** because you get more done quicker, with less.

I touched on several important subjects in this section a very conceptual level. More detail can be found in our article *Continuous Integration, Delivery and Deployment in .NET projects*.

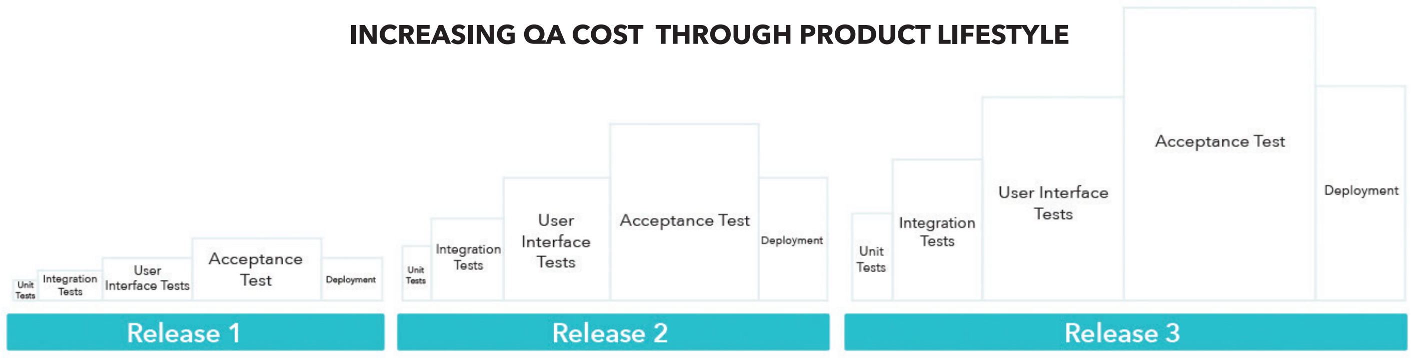
STEP 5: INTEGRATE YOUR QA AND DELIVERY TEAMS

Separate teams are often used for handling delivery and quality assurance. The intent, from my understanding, is to remove conflict of interest in instances where a delivery team may misrepresent actual quality of the deliverable or not test the application enough. It also provides an additional layer of testing and assurance for software applications that are of sensitive nature.

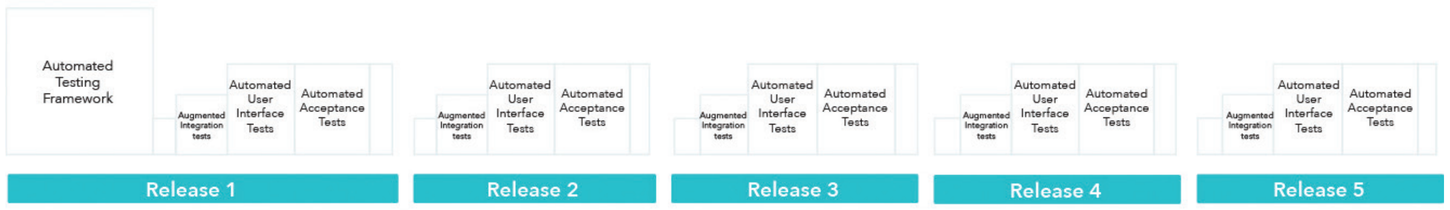
More often than not, a project plan will have a User Acceptance Testing (UAT) and System Integration Testing (SIT) phases somewhere at the end of delivery. During UAT, testers run the product through specific scenarios that mimic production use (e.g., a user needs to check their machine status by tapping on "Report Status," a user stops a job and receives an SMS confirmation, etc.). SIT deals with the technical integrations and looks at the product as a whole, the depen- dencies, the integrations, authentication handlers, underlying API's, etc.

The problem with this approach is that a project manager that plans this waterfall exercise in pursuit of better quality has absolutely no control over the outcome. Let's pretend for a minute that the delivery team did a mediocre job. QA spins up its defect Gatling guns and the delivery team is unable to

INCREASING QA COST THROUGH PRODUCT LIFESTYLE



INVESTMENT INTO QA AUTOMATION SAVES COSTS



resolve all of the defects in time for the original launch. Too little, too late to be on time and on budget. Once the issues are resolved they need to be tested again. Now the SIT cycle got pushed. The never-ending cycle continues. Also known as the "most stressful part of the year when we migrate software to production," which is typically prior to the start of the subsequent fiscal year.

SIT has its own share of issues that are predictable. It is likely that environment access is unavailable when the product is being created. Therefore, the team has to rely on service stubs (essentially mock interfaces for the services that will become available once the product goes into the QA environment), which is ineffective.

Stubs, by definition, imply that requirements for all web service calls are defined at the beginning of the project and are locked in with detailed documentation on data types, field lengths, and data formatting. Detailed requirements are evil and scope and requirements change throughout delivery. By the time you're half-way through the project, stubs are useless. Furthermore, stubs don't return real data, data that may impact how the application responds, and thus prevent you from building useful integration tests.

There are several tiers of integration you can bring to your delivery strategy. Pick one that is easiest to pull off within your organization.

Additionally, integrate UAT into your delivery sprints. Have the QA team integrate into the sprint-based delivery schedule and execute tests on fully functional software. While this may seem like more work, it actually reduces churn significantly since issues are resolved quickly and are included within acceptance of the sprint. Using the same toolset (such as Jira) also allows the team to quickly map defects to user stories and validate the acceptance criteria used, avoiding the disconnected practice of tracking defects in an enterprise QA system that is virtually invisible to the delivery team.

Use development and staging environments with access to all dependencies. Similar to the above, there may be an initial cost to set up the infrastructure, but the QA cost savings are massive over the lifecycle of the product. Use stubs only when a service is being created in parallel to the product and even then make the service available as soon as it has functional calls.

Introduce unit tests and inclusive code coverage. A unit test is a test that validates the smallest testable part of an application. It independently scrutinizes small, functional blocks of the product for proper operation, reducing the quantity of manual testing long term. An automated testing strategy is another area that will introduce up-front costs into the project, yet severely decreases manual QA involvement throughout delivery. Unit tests are written to satisfy or fail based on the acceptance criteria within the user story in your backlog. Each time code is checked into a centralized repository, the build server (e.g., TeamCity) performs the build and executes the tests. If tests fail, the engineer is not allowed to check in the changes, automatically preventing defects.

NEXT STEPS

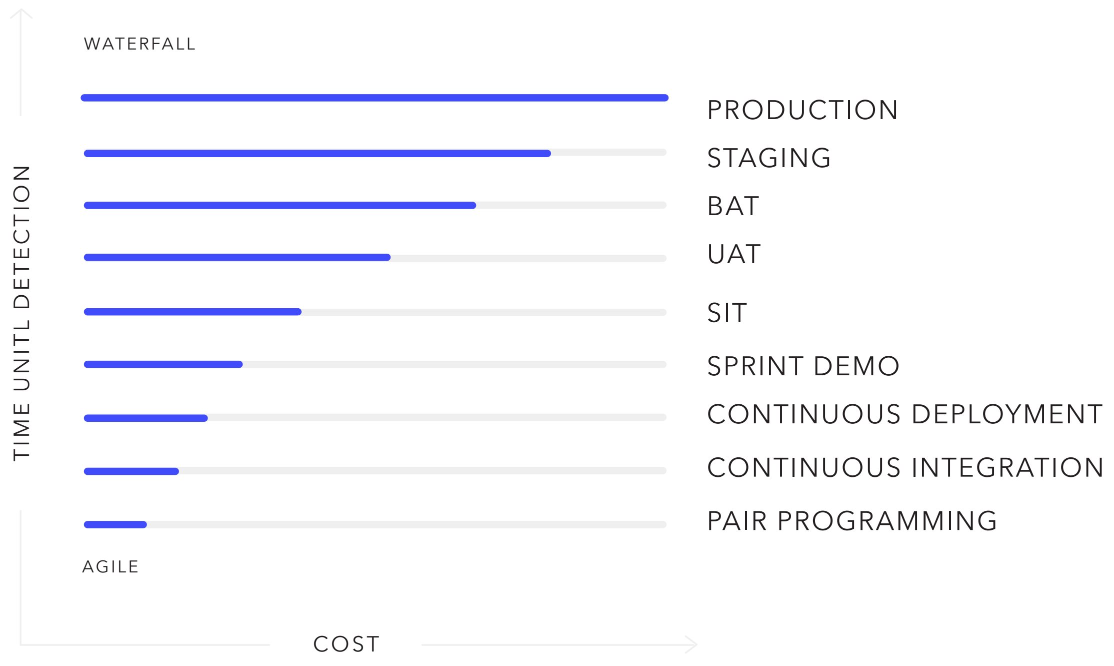
An agile transformation for a large, globally distributed manufacturer with an established enterprise delivery framework will take time and grit to execute. You're turning an aircraft carrier around, after all.

There are several critical topics that I did not cover in this white paper, such as the effect of culture in team collaboration, transition of roles from waterfall to Agile, product budgeting practices, and delivery metrics. Our experience has shown, however, that you have to start somewhere and these five mechanical, process-based changes can have a massive impact on lowering costs and increasing delivery velocity.

Select initiatives that are not part of the ERP core. Pick initiatives that are customer facing, products that you can quickly iterate on without the need for deep architectural changes within a manufacturers core platforms. I've seen this called the "Agile Edge"—a special group within the organiza- tion dedicated to innovation and rapid delivery.

A revolution needs its muscle from the bottom up, yet tactical support from the top down. Consider picking team members that are not politically tied to the "way things used to be" and provide senior executive sponsorship to push the agenda forward on environment setups, tool adoption, and change of process. The team will run into a lot of resistance and nay saying, but the results will speak for themselves.

COST OF ESCAPED DEFECT – AGILE VS. WATERFALL



Our experienced teams deliver software 4x faster than the industry average.

ABOUT DEVBRIDGE

Devbridge builds custom enterprise applications that deliver measurable results for Global 2000 organizations and their customers across all industries. Our cross-functional teams use deep expertise across product, design, and engineering to solve some of the most complex business challenges with elegant software. When building digital products, the team matters. We take ownership over results and ship mission critical, user-centric software fast.

ABOUT THE AUTHOR

Aurimas Adomavicius, President and co-founder of Devbridge. Founded in 2008, Devbridge revitalizes the largest of enterprises with custom software. When not in the trenches working with clients, Aurimas is an active speaker and writer on product design and engineering best practices.

DEVBRIDGE®

Faster than you’re used to.