Tearing Down the Walls

Embedding QA in a TDD/Pairing and Agile Environment

STEPHANIE SAVOIA, Marchex, Inc

How can quality be a responsibility taken on by an entire team?

At Marchex, Inc., we started by removing quality from a purely downstream activity. We added it upstream to requirements gathering and meetings. Then we actively injected quality "in stream", during coding, by embedding QA into the coding process on a team with test driven development and pairing.

1. INTRODUCTION

Imagine a wall. Now imagine a developer creating code and throwing it over that imagined wall. Imagine the developer moving onto another story card, task or project and putting this code far out of their mind. Sometimes it's so far out of their mind that when you have questions, they have to go back into the code or read, possibly outdated, documentation to get you the answers.

What usually happens to the code that was tossed over the wall? Well, it lands on the heads of the Quality Assurance (QA) or Test team. This is where the code is tested in various ways. If the development team does unit testing, then many of the tests QA are creating/executing are duplicative. If there are no unit tests, QA are left to test as much of the affected code as they can. Having no unit tests reduces the opportunity for QA to reduce the size of their test suites and limit QA as a bottleneck. QA could have eliminated tests that would have been executed as a unit test. In addition to the new feature testing, the QA team is responsible for regression testing. Regression ensures things that weren't broken before code changes were made, did not break as a result of the code changes.

We, as a company, wanted to change our development methodology to Agile. We wanted to have more reliability and quickly deliver value to our users. How could we become Agile, get new code tested, make sure old code didn't break, improve the quality and deliver the software quickly while retaining the goodness QA had been providing?

This paper will describe the way that Marchex, Inc., a mobile advertising technology company, evolved from Waterfall to Agile and how one team in particular also incorporates TDD and Dev/QA pairing into the way they deliver their software.

2. BACKGROUND

When I joined Marchex, in 2009, Dev and QA were divided. The division was both mental and physical, two different buildings a block apart – there was literally a physical wall (and a street) between Dev and QA. Meetings were padded with time on either end for people to walk from one building to the other. Neither group was partnering with the other group for the greater good of the company or the code.

We were using a very traditional waterfall methodology. We had documents upon documents for an entire feature that would be coded and released as a whole. Our releases were date driven and would occur every one to three months or so.

Our workflow looked like this:
Requirements were gathered by the business team and specifications were written. We had a Functional Specification Document (FSD) which would be the initial request document from the business team on what they wanted and why. The FSD would be reviewed by the Dev and QA teams and the business team would revise the FSD as necessary. Then there was an Outline Systems Design (OSD) document which was the implementation document produced by the developers in response to the FSD. This would be reviewed with the business and QA. The OSD would be revised, if necessary, by the developers. Then the QA would write a test plan using both the FSD and OSD documents. Yes, another review would happen with the business and Devs. Again, there would be revisions to the test plan as necessary. After this exhausting process, the coding would begin. The QA team would receive one to three daily builds to test against. QA saved the tests into a repository

Author's address: Stephanie Savoia, Marchex, Inc., 520 Pike Street, Suite 2000, Seattle, WA; email: ssavoia@marchex.com

for reuse in regression testing if/when the feature was coded against in the future. Additionally, there would be wikis written to describe every detail of the feature once it was ready for release.

Early in 2010, we were informed that we would be "going Agile". But there was no plan to accomplish the transition that went along with the statement. We were left to our own devices to determine what "going Agile" actually meant. Some teams embraced an Agile way. They incorporated Scrum boards. They had their daily standups. They took the time to fill their backlog with upcoming work. They developed and tested within a sprint or iterations. They finished up their iterations with a deployment to production and at the end of their iterations they had a retrospective.

On the other hand, some of the teams thought they were Agile, but were more like Agile-Fall. They would do the things that appeared to be Agile like daily stand ups, Scrum boards and iterations. The act of development though, was being executed in a shorter waterfall session. Coding would be done within the iteration but testing would seldom be completed when it needed to be.

There was a lot of intra team confusion. Do we still document our software? Well, we don't need the heaps of documentation we had been creating, but we still do need to know the business requirements and have basic acceptance criteria. We needed to remember that a story is a contract to have a conversation. Other acceptance criteria may arise while coding is occurring – and this is OK. Some wiki-based documentation is very helpful to enhance tribal knowledge; however, we did not need to write nearly as many wiki pages as we had been. We needed to learn what level of detail was important and what was wasting time.

The company did progress by consolidating from two locations to one. Additional space was leased so that we would maybe have a floor or two separating people instead of a couple of city blocks. We thought this would bring about an immediate change for the better by being in one space. It actually had the opposite effect. The mental separation between Dev and QA was more visible and it may have even increased. People were still not partnering and were cutting themselves off to conversations behind their cube walls. Think of a child closing their eyes and saying, "You can't see me." It's pretty much how a few people were behaving.

There were communication issues between business stakeholders, developers and QA. Bottlenecks, time crunches and overtime were the new reality. Because QA was seldom finished testing within an iteration, they had become the main bottleneck. QA was not adept at asking for help. Our developers had been instructed to help QA finish testing to keep iterations on track once their coding was complete but they were not being utilized as a resource.

The lack of a Dev/QA partnership was highlighted by the fact that there were now two Scrum boards leaning up against a wall, side by side, as well as separate retrospectives. We had a board with Dev story/task cards and a board with QA story/task cards. Our releases were no longer deadline or date driven, but they were inconsistent, unpredictable and low quality. We were still working in a build first and test after pattern.

What was standing in the way of that elusive partnership? Dev and QA ultimately had different goals. One group is mainly about creation and the other is trying to ensure the quality of what was created. There was also fear. Devs feared having more work to do by learning some testing principles. QA feared losing their turf if the developers learned how to do some testing.

To compound these issues was the addition of Test-driven development (TDD) to our process. This is where it started to be apparent that having the separate Dev and QA processes were entirely too duplicative. Change isn't easy for anyone, let alone trying to adjust to a vastly different way of thinking about the whole Software Development Lifecycle (SDLC).

2.1 Becoming more Agile

In 2011 we had a change in engineering leadership which led to extreme changes in how we worked. I do mean extreme changes. Extreme Programming (XP) was added to one team to handle a very large project with a pretty darn short turnaround time. A select group of developers and QA were on this team with subject matter experts loaned to them for a day or two at a time when needed. Iterations were shortened from the usual 2-week cycle to a weekly one.

I would be remiss if I didn't point out that the overall team was a large one. We started out the year with 24 people. Fifteen of them were developers and 9 were QA. The numbers changed drastically over year.

Our new Director identified roadblocks, ones that we had created ourselves, which stood directly in the way of the Dev/QA partnership that we wanted to have. We consolidated both of the Scrum boards to one. Our estimates still had two numbers on the cards to indicate Dev and QA estimates, but it was a good start. We also went from Dev and QA having their own retrospectives to having a single retro.

From the quality side, we were trying to find a way to go from quality control (QC) to performing quality assurance (QA). We may not want to admit it, but testing after build is really quality control. We brought QA into the front of the process. We were added to meetings ranging from requirements gathering to assessing risk. We have strong domain knowledge about the usage and capabilities of our systems and can keep rework costs down by make people aware of possible issues early in the process.

During this time, there was an overhaul in management roles across the company. Our QA Manager positions were eliminated. The QA staff would report into the Dev managers. No QA managers lost their jobs. They were transitioned into new roles including Software Developer in Test (SDET), Sr. QA and Technical Program Manager. Each manager was able to choose the direction they wanted to go. This led to more fear from the QA staff on our team.

Management on our team also determined that they no longer had a need for the SDET role. In this instance, we did have one person leave the company but the other three SDETs were transitioned to other departments that still had a need for the role or skill set. One of the SDETs transitioned to DevOps and ultimately became a developer.

We added a support team that does light SDET-type work creating tools for internal users and handling issue tickets for our platforms from both internal and external users that would have previously gone directly to developers to research and fix if required. This freed up our developers to concentrate on delivering features and fix the occasional bug that was too complex for the incident management team.

The results from these organizational changes were weekly releases with much higher quality. Developers were now accountable for quality that the SDETs had previously taken on. Our SDETs were much happier on their new teams and roles but the QA Engineers that remained on the team were now even more fearful and stressed. Their fear was of change. The testers were comfortable with their familiar processes. They didn't see a problem with executing tests after coding had been done. They didn't mind having a voice at the end of the process. They didn't mind not influencing the quality sooner during and before development.

3. HOW WE EVOLVED

Towards the end of 2011, management requested the remaining QA change the way they work within our group. It was a bold request. It tested QA's ability to truly be agile. To show that they could let years of traditional QA practices go and fully adapt to a new way of thinking. In order to be successful at this, they had to have completely open minds. Change not only their thinking, but also their actions. Actually PERFORM quality assurance by building quality into their daily activities.

We were given guidelines that addressed three primary areas in the new methodology to ensure quality: Measurement, Advocacy and Using the information.

Measurement:

Quality as a whole is difficult to measure. You can count tests that pass and fail, but important characteristics of high quality software is that it has a low cost to maintain, has few issues in production, has happy customers and solves the customer's actual problems. How is this type of quality measured? We had to define both thresholds AND quantifiable measurements we could combine to give us a level of success while also ensuring the words "happy", "low", "few" and "solves" were still occurring. For us, "happy" and "few" would be measured by lower volumes of support tickets being opened for our software.

Advocacy:

We work with two types of users – Internal and External. You have to know who your users are to know who you are advocating for during development or in meetings. You could be speaking up for one or both user types. Advocacy may be done in conjunction with a Project/Program manager, a UX person or on your own. It will depend on the makeup of your team. Here are a few ways we use to advocate for our users.

Internal Customer/User advocacy:

Talking to our humans! We talk to the users of our UI and/or data. We shadow them. Understand their job functions, constraints and goals. Learn about the pain points with the software they have and the software they want. We dig in and find out what they really need and bring it back to the team. These sessions are used to gather requirements, requests and acceptance criteria. This will help solidify what "Done" is for our stories.

If you are not solving your user's actual problems, they will be the first ones to find or create their own workarounds. Their workaround may be to stop using your software. This is bad! You can make a difference by giving your users a voice and a way to propose features to be prioritized into your backlog. You want them to talk to you; you really do.

External Customer/User advocacy:

Similar to internal users, external customers know what they are looking for and what they want to get accomplished. They have workflows they prefer and they know if something is too complicated to use early on.

For this, we use Empathy interviews, with our external customers. They are a great way to know you are building something that matters to an external marketplace. There are a few techniques that can be combined in an empathy interview to gather data a few different ways.

Some companies have User Experience (UX) teams that coordinate this type of interview but not all companies have them. A UX team isn't a requirement when interviewing external users. Anyone from a Project/Program manager, a developer, QA, etc can be a part of them. There are usually two or three people from your company there for the interview. At Marchex, we have a person tasked to ask questions, a person to take notes for us to review later and one to observe the interactions. The goal of these interviews is to uncover insights into the customer's beliefs and values to help get to their core problems. We use this information to solve them.

We work directly with our business team to identify external customers that would be willing to come in to talk to us or even better, allow us to come to them for a 60 to 90 minute interview. We incentivize them by offering a perk for their time. Usually it is a cash incentive. Cash works better than a gift card – they can spend it anywhere.

A picture is worth a thousand words. Visuals are the best way to convey new product ideas to potential or existing customers. During empathy interviews though, we do not demonstrate actual software, it's pencil and good old paper for us. People will be more honest about changes they would like to see if they don't think you've already spent hours developing a prototype. We'll return to them later and demo the software that was created which included their input.

When we are able to visit their workplace, we usually learn things about our customers we didn't already know. We observe their interactions with the product, their customers, pretty much anything we can't see them do in a visit to our office or a corner coffee shop. Customers may not even realize pain points anymore if they have gotten used to them.

Using the information:

The information we gather is used in the measurement and advocacy areas in our design discussions, planning, story writing, and estimation. We keep the information handy to use anywhere we can BEFORE we begin coding and we continue to use it as coding progresses. Bringing information in after the code has been built is too late to make effective decisions. Our teams are constantly driving the code to completion. In order to make the right choices, the information must be there.

We still do some testing of built code but it is more exploratory and scenario/workflow based testing and it isn't always done by QA. This is where we trust and believe in our developers. I have that trust in my team. I know that they care about the quality of the software so I can think about quality in other ways.

We don't have to focus on the immense number of tests and test variations we previously created. We are very familiar with how the internal and/or external users actually use the products from the empathy interviews, shadowing and our own use of the product. Workflow issues are apparent quickly along with issues that we find exploring the software.

3.1 Building quality into the product

A way that we built quality into our product was by maintaining automated unit testing with Test Driven Development (TDD), but we expanded it.

For those new to TDD, we 1) Run the existing unit tests to ensure no tests are currently failing. 2) Use TDD to write a failing test for the story/task. 3) Add or edit the code. 4) Re-run the new test and edit the code until the new test passes. 5) Run all of the tests to ensure none of the pre-existing tests are failing after the code changes. If some tests are failing, fix the code and repeat step 5 until all tests are green. 6) Deploy the code. The new test is now a part of our regression suite in our development and CI environments.

Our QA are now working directly with developers. We are sitting with the developers, designing the tests that are going to impact the system interactions and workflows. The developers are also creating our automated integration tests in this process.

The QA job has morphed. We are no longer writing formal test cases. We are no longer managing our tests in a separate testing repository. We are no longer creating duplication in our testing.

We found that we had a 50% duplication rate. Fifty percent of the automated tests that our SDET's had written were also in the developer's unit test suite. These tests, consisting of unit, happy path and some negative tests had already passed and did not need to be written and run again by a different team. This was waste. Waste of time and resources that could be reclaimed in our new methodology.

QA staff are gathering information in interview/shadowing sessions with users to have better acceptance criteria and further develop user stories. We are working with our end users to gather better workflow and usage scenarios. We are able to advocate for the users in planning and design meetings. We are working side by side with developers to create the unit, integration, negative tests and help with maintaining existing tests. We are exploring the software for issues not covered in the user scenarios. And we are still adapting to this new way of life.

We put this radically different way of working in to full practice the second half of 2011. Those six months exposed the QA team in a couple of ways. It exposed team members who could keep an open mind, try and adapt and those who couldn't. Some of the team moved to other divisions within the company that operated in an Agile or Agile-Fall methodology but not as forward thinking as this team, while some left the company all together.

In the first quarter of 2012 we had 13 Devs and 1 QA. This was down from the 15s Dev and 9 QAs at the beginning of 2011. There were some corporate organizational changes at the end of the first quarter that merged a couple of teams together which brought us to our more stable number of 25 Devs and 5 QAs. This group of QA was up to the challenge of the new way of ensuring quality. The staff is divided up between 5 teams. Some of the teams have QA and some of them do not. I will pause a moment for you to gather your composure back. Remember, we aren't doing traditional testing any longer. We do have 5 QAs but we all approach what we are doing a little differently. The teams that do not have their own QA share an uber QA lead that advocates for the users, advises the developers when they have questions and shadows them when they are validating their code in the testing environment.

3.2 How I was Embedded into a Pairing/XP Team

So let's dive into how one of the teams, my team, approached quality. I am embedded on our data warehouse/business intelligence team. My role is much different than the traditional test role I was used to. I'm a direct member of the pairing teams. I actively pair with developers on stories from initial conversations with business and/or system users to planning and estimation to TDD coding, integration and some exploratory testing.

Pairing gives the developers and me the opportunity to talk through the unit and integration tests that are going to be written for our continuous integration (CI) tool. I'm not a coder so I focus on asking questions as early as possible, prior to the developer actually typing. We talk over the user story and add any acceptance criteria to the story that may have been missed in the initial planning/estimation meetings.

I'm also the code reviewer member in my pairs. I'm pointing out typos during the coding. I'm inspecting tests for missing assertions and asking why we may or may not be asserting on something. I'm continually asking questions and making suggestions if something in the code seems too complex or a bit off or I just don't understand. My team refers to me as their quality conscience. There is minimal UI work on our Warehouse team but what is there is also scrutinized. The pair reviews content and display and corrections are able to be made on the spot. This saves a ridiculous amount of time for our team.

I was given the choice of teams to work with after the team mergers in March of 2012 and found myself drawn to this team for the challenge of learning an entirely new process. I had been a UI tester for 10 years and I was ready to take a leap into a new realm. I had my "Ah-Ha!" moment about six weeks into the creation of our first data warehouse. We were integrating with our first source system. Once we had the fact and dimensions identified and were actually ETLing (Extract, Transform & Load) data into our tables, it was time for data validation.

In the traditional way I had been testing, I would have either been notified that data was in the tables or I would have received a new build and put data into the tables myself. I would have done the validation, found bugs, written bug reports, waited for corrected code, re-validated the data and repeated it until I was comfortable with the build and the data. In our new system, I was right there sitting with the developer and we were validating the data together. We were coding and testing what we were creating as we went. Having experience with previous data validation in our systems, I knew the process of back and forth would have taken us several months to get to a production ready pipeline. We were ready for production in six weeks. Months of time cut off of a project that I thought would be much more daunting.

Our re-work has been minimal because I've been talking to the developers about the tests that need to be written instead of waiting for code to test. We switch up pairs each time a new card is picked up so I work with all of the developers within about two weeks. Our team as a whole is now responsible for the quality of the software. I don't have that daunting feeling that it's all my fault if a bug makes it to production. I think a little more quality mindset rubs off on the Devs each time I pair with them.

We do full integration testing together in our QA environment. We test with production data sets and send that data through our QA warehouse pipeline so we can find data anomalies that may arise in production. A luxury our team has is that we can do final hardening on production hardware prior to actual production usage. We do expect to find some edge case scenarios we hadn't thought of as we pipe production data through the system; however, we can remove all of the test data in production prior to starting actual data flow. As close as your QA, staging or sandbox environments may be to production, they still aren't.

Once a code change has been deployed to production, I sit with developers to do post production verifications for each story/task to ensure the new deploy was properly put out to production and is working as we expect. Unfortunately, since much of our work is ETL, we do have to wait for the next run in order to do that verification. This can be a couple of hours or the next morning depending on the frequency of the ETL that we deployed.

If a bug does rear its little head too late in an iteration or in production, we write a story card. We talk about the risks, add initial acceptance criteria, estimate the story and the bug gets prioritized in the backlog. If an issue cannot wait for the next weekly deploy, we do have an embedded Ops person on our team, as well, and we can get our hotfixes out quickly.

Let's keep fear in this QA/Dev pairing approach to a minimum by pointing out that SDETs are not left out of this new way of injecting quality into the entire lifecycle. SDETs are active coders in a pair. SDETs are writing the test code while the developer writes the software code. The SDETs are asking the same questions that I would, but they are driving during test creation instead of the developers as in my situation although I do write ETLs in a GUI if I'm pairing on one of those cards.

Another tool we continue to use on the last day of our iteration is a retrospective. As a team we white board our takeaways from the week. What worked well and what didn't work so well. What did we learn from the good and the bad? What can we take action on? Are we noticing any patterns for things we do that can be standardized?

Since joining the team in April 2012, we have stayed consistently around four developers and me but we plan to expand. We had a developer leave the company in early 2013 and then return to our team again in early 2014. We have had developers request a transition onto to our team from within the company and have converted a contractor to an employee. We also lost a developer to leaving the industry as a whole. The churn on this team is very low which contributes to the high productivity and morale.

My learning curve has been a steep one since I was new to pairing and back end services but I never received anything but patience and encouragement from my team as well as excitement that I was there to help maintain and improve how we all thought about and accomplished quality software.

4. IMPORTANT POINTS

Like with all development teams, we are here to get value to the users. It doesn't matter if they are internal or external users. They need or want what we are creating and they want it quickly. As a data warehouse team, we are supplying and connecting disparate data for both internal and external users. Internal users are analyzing our data across platforms to make sales, make adjustments and identify future features to request while our external users are utilizing our reports to see how the money they spend with us is improving their business.

We have a highly motivated and passionate team that gets things done. By having our entire team be responsible for the quality, we eliminate a large testing roadblock. Ownership is everywhere on my team. I'm still that different-minded person on the team who ensures we are testing more than the Happy Path. We aren't forgetting that we really do want to provide a quality product to the end users. I still act as a voice of reason and user advocate to remind us why we are creating this software in the first place.

4.1 Trust and Leadership

A very important part of this approach is trust in engineering and leadership. I touched on this briefly earlier. Trust is not a word to take lightly. It was also something I didn't have on previous teams both within Marchex and in previous companies. You have to trust your developers. In this new way of working, you have to trust

that the developers really do care about the software they are creating. You have to trust that the entire team has bought in to this type of software lifecycle. You have to agree that quality is not just QA's job, and trust that the software being delivered by your team has been written with quality in the forefront of everyone's mind. You are there to question and remind, but you must also trust.

I had to trust in my manager and not take each and every bug we found in production personally. I suggest that you go ahead and share feelings of annoyance for having a production issue with the rest of the team. They have ownership too. Conversely, leadership has to trust engineering too. Leadership has to trust in the engineering teams.

Our team excels because we have leadership that truly believes in what we are doing. I can't tell you how important buy in from leadership is. It doesn't matter how enthusiastic a team is to try **any** type of methodology. If your leadership doesn't care about it as well, it will not succeed.

Our team is also predictable with the amount of work we can complete during an iteration. Predictability is used for forecasting what the team can deliver in the future. With our points structure, we know we can deliver approximately 20 points in a one week iteration. Our program manager and Sr. manager can use this data to predict how many features in our backlog and roadmap we can get done in the next month or even the next half of the year.

5. HAS OUR TRANSITION TO AGILE AND DEV/QA PAIRING WORKED?

In a word, yes. As with any major change at our company, it was painful in the beginning. Some of the pain was due to a large portion of our team being new to fact and dimension tables and data warehousing in general. We had a large amount of failing fast early in the project. Failing fast is OK though. We prefer to fail fast so changes can be made as early in the process as possible. We recognized design issues and were able to learn, adapt and rework the design early and quickly. Rather than delivering an entire product, all at once, with flawed designs and providing workarounds, we delivered incremental value and received feedback much sooner from our stakeholders.

We now have internal analysts that rely on our data. They can connect data from disparate systems through our warehouse. The analysts come right to us when they see something they think may be off or when they find a trend or optimization they didn't previously know about from the old spreadsheet magic they had been using previously. It's great for us to hear the good coming from our data. It's also great for us to hear the business questions they are asking of our data and what they are trying to learn from it.

External users are using our data now as well, even though they don't realize it. We seamlessly replaced the way the UI's they were using consumed data. UI teams have replaced certain queries against their own systems with API calls to populate reports from the warehouse data instead. They are seeing the same data they had been looking at previously; only we are delivering it more efficiently now.

Our code quality has improved and our test coverage gives us a comfort from both unit and system level perspectives thanks to the work we are doing with automated TDD and integration testing. To be clear, we don't watch a test coverage metric. Instead, our approach to testing with Dev/QA pairing, the trust we have in each other, feedback from users, and the performance of the product in production means that we know we have the coverage we need.

I consider myself very fortunate to be on this team. Morale is quite high and it's a great feeling to be on a truly happy team. I also know our product even better than products I have worked on in the past with the over-the-wall processes. I use that knowledge to keep the quality of our features and accuracy of our data always on the minds of our team.

6. ACKNOWLEDGEMENTS

I'd like to thank my Sr. Director, Joe Blotner and Manager, Clay Colburn for pushing us to try this new way of working. I'd also like to thank my team members, Chris Zazzi, Michael Kunugiyama, John Aegard, Justin Rudd and Kyleen MacGugan for being open to working this way and embracing a team philosophy to caring about the quality of our products. Finally, a huge thank you to Joseph Yoder, who shepherded my paper and Rebecca Wirfs-Brock of the Agile Alliance, I couldn't have done it without your help and guidance!