Amping Your Success by Leading with Quality: Why "Shifting Left" Isn't Enough

Heather M. Wilcox

Heather.wilcox@hmhco.com

Abstract

A common topic for discussion in recent years amongst the quality community is the “Shift Left on Quality”. This is meant to express the idea that quality engineering should start as early in the development process as possible - within code, unit tests, automation, etc. But, is simply shifting left enough? What if organizations began to “Lead with Quality”? That doesn’t just mean putting QA engineers in the room early (although it is usually a smart move). Leading with Quality is a bigger idea than that. The concept is all about making smart decisions from THE BEGINNING of the planning process for a product. Regardless of whether you’re Agile, Waterfall, or some other SDLC paradigm, I contend that it is vital to have the right people in the room representing the right parts of your organization so that the right decisions can be made to enable the construction of a quality product. In my experience, many problems can be tied to a failure to lead with quality. For example, having non-technical people design a technical solution or perhaps choose the tools that the developers will utilize can lead to an array of problems during the implementation phase of a project. This paper will leverage my 25+ years of experience in the software industry as well as documented examples to delve into what it means to “Lead with Quality” and how the application of the concept can save time, money, and effort and will result in a higher quality product.

Biography

After leaving a potential career in Anthropology, Heather has spent over 25 years working and learning in the software industry, choosing to focus primarily on start-up and small companies. As a result, she has had a broad range of job descriptions which include, but are not limited to: Technical Support Engineer, IS Manager, Technical Writer, QA Engineer, QA Manager, and Configuration Management Engineer. More recently, Heather has moved into a more permanent relationship with a mid-sized educational company where she’s been for the last 12 years. This has given Heather a wide range of experiences to draw from in her current role as a Quality and Reliability Manager. In her spare time, Heather enjoys fiber arts, equestrian sports, and training donkeys.

1 Introduction

When I imagined the concept of “Leading with Quality”, I thought it was kind of a “no brainer”. It felt so obvious that I wasn’t sure it was even worth discussing. However, when the idea was pitched to my peers, I got very similar responses from everyone. Consistently, the feedback was something to the effect of, “This is a great idea! It seems obvious but no company I’ve worked with has done it, so you need to talk about it!” As part of those conversations, I realized that, after seven companies (working on widely varied technologies), I hadn’t experienced it either. Many of those organizations clearly cared about quality and put resources into building better products, but none of them actually “Led with Quality” as described in this paper. That revelation, along with the encouragement from my co-workers directly inspired this work.

In my experience, the origin of new piece of code is nearly always “A technical problem” of one form or another. However, I’ve observed numerous software organizations either ignore or not realize that the problem they are solving is fundamentally a technical one. This failure happens at the most critical point in the software development process – the beginning. The direct result of this lack of insight is that the people most qualified to solve the problem - technical people - get left out of the initial conversations and that is where Quality Failures begin. (Cherry, 2019)

I’ve witnessed versions of this scenario play out many times in my career and, over the years, have heard similar stories from colleagues:

1. A need is identified: “We need a widget that stores data from a web page!”
2. Marketing has done all the research to prove that the addition of this widget would be a valuable feature for the current product line.
3. Sales swears up and down that people have been asking for this functionality for years and they’ll be able to sell it to every customer the company has ever had and, what’s more, it will attract new clients in droves.
4. It is the PERFECT idea and all that is needed is a Dev Team to build it!

Marketing realizes the value of this widget and doesn’t want development to over-think it – nor do they want to waste engineering time since developers are expensive, so they assemble a typical “Discovery Team” (Muzyka, 2021) to design the perfect piece of software – right down to the technology that will be used to build it. And then they deliver this beautifully designed concept to the development team complete with licenses to “The Perfect Tool”. Marketing believes that there’s no reason in the world why Dev can’t use their Agile methodology to have a shipping Minimum Viable Product (MVP) in 3 months.

Clearly this is a contrived example; nevertheless, as previously mentioned, I’ve witnessed variations of this in just about every company I’ve worked for, and I believe it happens often. In 25+ years of working in the technology industry – at least 20 of those in Quality, I can only think of a very few times – certainly less than 10, where the “right people” got in the room early and were able to have a positive effect on the final product. On the other hand, I can think of many examples where I or someone I knew, discovered a significant problem very late in the development process, that could have been avoided by bringing the right technical person in during the initial discussions. The failure to engage the right experts from the beginning led directly to a compromise in Quality before even a single line of code had been written.

2 The Beginning: Why We Don't Put the Right People in the Room and Why Those Reasons are Wrong

There are many reasons why companies don't put the right people in the room from the start. I know I've said some of them myself and heard versions of all of the following: "Engineers are expensive", "We don't want to disturb Bob because he's working on important stuff", "If you let an Engineer design software, you'll end up with a shiny UI with a million buttons that our customers won't be able to use", "QA doesn't need to be there until the code is finished", "Support doesn't know enough about software development to be helpful."

However, when you dig into these statements a bit, they fall apart quickly.

"Engineers are expensive." It is common belief that engineers are paid more than their Marketing equivalent. This is often used as a justification for waiting until later in the planning process to bring Senior Developers in to consult. The argument is that, if an engineer isn't working on actual software development tasks, their expensive time is wasted. The truth is that the "expensive engineer" is a fallacy (which will be addressed later in this paper). However, regardless of the cost of engineering time, bringing in the right technical person to consult on a project may save thousands of dollars (or perhaps millions, depending on the scope of the project). An engineer will understand the available technologies and their capabilities as well as the possibilities and limitations of existing infrastructure and software. They can make recommendations that will help the Marketing team set up the project in the best possible way and avoid early errors that become expensive problems later.

"We already have an architect on the team – we don't need an engineer." This argument stems from the notion of avoiding a duplication of effort - why have two individuals performing the same role in the room? Yet, this doesn't accurately reflect the situation. Solution Architects often view and understand software from a very high level. (Coursera, 2023) I have witnessed Architects that were very "hands off" and did not understand the nuances of a particular system or a piece of code within it that could complicate the proposed solution. Conversely, an engineer who has created code for the affected product or system will have knowledge of granular technical details that an Architect might overlook, but that may be vital to the success of the new project. This specific knowledge may be helpful in providing valuable insight into whether a proposed solution is reasonable, efficient, and functional. The software engineer may also be able to leverage their deeper knowledge of the existing code to suggest a more appropriate option.

"We don't want to disturb Bob because he's working on important stuff." In this case, the perceived problem is that engineers should be focused on project related work as close to 100% of the time as possible. Every moment of an engineer's day that is not spent coding or working on code is valuable time wasted, since there is always more to do than there are developer resources available to do the work. It's true that there are rarely enough engineers in a company and the best ones are in demand every minute of every day. However, as with the issue of expense, the investment of a few hours of their time at the very beginning, may save hundreds of hours of rework by multiple engineers later in the project. Isn't it better to spend valuable engineer time up front to prevent the waste of, potentially, much more time later in the project?

"You don't want to bring Engineers in too early in the process because, if you let an engineer design software, you'll end up with a shiny UI with a million buttons that our customers won't be able to use." This breaks down to an argument that Engineers are not UI designers so they shouldn't be involved in anything that might be UI design. My developer friends will probably kill me for this, but this statement is likely closer to the truth than most of us would care to admit. However, developers aren't being asked to design the UI, they're being asked to help design a software solution. A good solution and a good UI are not the same thing. Clearly, the User Interface drives the software, but the underlying technology is the core of the product. Good developers are more than capable of designing a technical solution that your UI experts can create a beautiful interface for.

"QA doesn't need to be there until the code is finished." This statement is a leftover from the days when most companies used a Waterfall lifecycle and testing, or quality assurance work was always the last step before the public saw code. The Shift Left movement has proven that sentiment to be incorrect. (Testim, 2021) QA absolutely needs to be involved from the beginning (this is shifting WAY left). Nobody will understand how to build in quality better than a Quality Engineer. They'll help identify areas where the proposed solution might be lacking – where it might not work for all customer use cases or where there might be a confusing piece of design. They will advocate for your customers in a way that Marketing cannot because QA understands both the technology and the practical usage models. (Harris, 2015)

The final common scenario: "Support doesn't know enough about software development to be helpful." This issue is caused by a lack of appreciation for the kind of knowledge that Customer Support possesses. Regardless of a support engineer's software development expertise, they are the ones who talk to users every day. They know how the software is applied in the real world and they can speak for the user in a way that even QA cannot. The support representative can confirm whether a proposed usage model makes sense from the user's perspective. They can also point out UI functionality or features that might result in a confusion and increased customer support calls, or that may not have enough value to be worth implementing.

3 Who the "Wrong People" are and Why They are a Problem

The design portion of a project is often called "The Discovery Phase", which is how it will be referenced for the purposes of this paper. (Khalimonchuk, 2022) This phase of a project is driven by a Discovery Team that normally includes a Business Analyst, UX Designer, Solution Architect, and a Delivery Manager. (Muzyka, 2019) I've also seen Product Managers (PMs), Technical Product Managers (TPMs) and Project Managers involved in the conversation. Marketing and/or the Business Analyst represent the needs of the customer. Either Technical Marketing or the Business Analyst understands the technology at a high level as well as Competitors' offerings and can describe the strategy around how the new product will elevate the company's stake in the marketplace. The Solution Architect has knowledge of the current configuration at some (again) higher level and understands the technical possibilities. The UI team knows how to design the product to be functional for the customers. The Delivery Manager understands the schedule and knows how to drive the project team to a timely conclusion. From many perspectives, this is the "dream team" for solution design. In theory, the team contains people that understand the customers and the business as well as folks that have enough general technical knowledge to cover for the actual Developers and QA that are out in the shop doing the important work of creating software. Upper management every reason to expect that the Discovery team will be successful in assembling a quality project that will be practical to implement and will meet the needs of the customer.

Once formed, the Discovery Dream Team designs a solution that they feel makes the most sense for the customer and for the company. When they've assembled all the pieces of the project, the team formally documents the plan and delivers it to the development team complete with the tools that will be used to build the new widget.

As previously mentioned, it is not uncommon for the Discovery team to select a technology or a tool for the development team to use as part of the design process. (Khalimonchuk, 2022) This selection is an early chance for the loss of Quality.

Technology decisions made by the Discovery Team can be based on any number of reasons including:

- The availability of a tool or technology the company already owns or uses.
- Information gleaned from websites. (Puzhevich, 2022)
- Financial concerns from the finance team, who might be averse to allocating funds for new technology, potentially pressuring a team into opting for free or Open Source tools.
- Pressure from a manager associated with the project. (I once had a manager demand that I re-code a fully functional build system in JAVA because he liked the language.) (Cherry, 2019)

- A salesperson from another company promises that their product will work perfectly for the project.
- Recommendation of the Solution Architect.

However, the team that will use the technology or tool may not be involved in the selection process. This can happen for several reasons. Often the Solution Architect represents the development team in tool decisions as that person usually has a higher-level understanding of the problem and the technical skills to understand what is and is not technically possible. Other possible reasons are that the software team may not yet exist, they may be busy elsewhere, or the Discovery team may not want to disturb their work. Irrespective of the rationale, the omission of the software team from the decision-making process can lead to a loss of quality, resulting in an unsuitable or suboptimal selection.

To return to the fictionalized example used in the introduction, the Discovery Team has designed a solution, purchased a tool, and handed it off to the development team. It may seem unusual that the discovery team handled the tool selection and organized the purchase, but I've witnessed this exact scenario, so I cannot believe that it hasn't happened elsewhere. It also appears to be recommended by some descriptions of the Discovery Process. (Khalimonchuk, 2022)

When the development team begins the project, the first problem they encounter is the tool that was picked. The technology was selected based on price and the assurance of the salespeople that it would absolutely work for the proposed software. Money was spent and a commitment was made. However, because developers weren't involved in the decision, there was never Proof of Concept (POC) or other deeper research done to verify the tool was the correct solution. The Architect should have done some of that work but decided that further technical discovery wasn't necessary because the software looked like a good fit, so that step was skipped. As a result, there was never a comparison of the product to POCs done with other solutions. (In an alternate version of this scenario, which I have witnessed, the developers were involved, but the Discovery Team disregarded the development team's concerns about the preferred tool.) It turns out that the purchased product doesn't quite do what is needed. But the money's been spent and there was a lot of set up required to get the tool functional. To ensure success, a third-party consulting company – "The Experts" were brought in. The decision from upper management is that enough money has been spent on tools and consultants that success should be guaranteed – there is no turning back. So, the Development team is forced to figure out how to kludge together the widget using an inappropriate tool.

The next problem is that solution has been designed as a new stand-alone tool. Had the team brought the engineering staff in on the decision, the developers would have explained that they already had something similar to the new design partially implemented as part of some work the team did in an innovation sprint 3 months ago. Or perhaps the existing software has a lot of the requirements already in place and an appropriate solution could be created in just a couple of months by leveraging the existing code. In either case, the development team is now duplicating previous efforts which is wasted time.

When the quality team looks at the solution, they are immediately frustrated. Even though it meets the needs of the market, the new design is inappropriate for the skill level of customer that will likely be using it. The Architect may have understood the technical problem, but they didn't enough knowledge about the actual user of the widget. For this product, the folks that are responsible for purchasing (customers) are much different than that the actual users of the product. The Quality Team has been aware of the problem for years since they work closely with support. As a result, the solution design, although appropriate for the people that buy the software, is completely wrong for the people who use the product. Although this seems like an unusual problem, it is not uncommon. Designated "buyers" within an organization will purchase products based on a perceived need or a set of requirements that may not accurately represent the needs of the people that will use the software. (Bird, 2019)

Unfortunately, even after all this new input and information, the plan has already been committed. Technology has been purchased, the design is in place, and a whole chain of promises have been made. (Based on a dev time guestimate from the Architect and from Engineering Managers, a delivery window is promised to Marketing. Marketing then makes promises to Sales and starts building collateral around a specific release window. Salespeople excitedly "hit the road", pitching the new widget to partners and

promising delivery during the estimated time window.) The only option left is for the team to suck it up and deliver. And then, in the best of situations, in 6 months or a year the development team can justify why a complete redesign is necessary – only this time, it will be a better design using a better tool that will allow it to better serve the users for which it was intended. Congratulations – the team just got to build the same product twice, for twice the money. And that's the best case scenario.

The worst case is that management decides that they're already too far in on the solution – too much time and money has already been spent. Because "sunk costs" are too high (Wikipedia 2023), the dev team must continue to try to make an inappropriate, under-performant solution work for the next several years until the software is discontinued or enough time has passed to justify a re-write. In the meantime, the product is the source of never-ending support calls, and costs a fortune to maintain because the solution is delicate and difficult to work on.

I've had the unfortunate privilege of watching both situations play out and can say that they are deeply frustrating – especially for the development team that must do work that they know is throwaway for all the wrong reasons. The situation is even worse for a team that knows that they're going to have to continue to struggle to support something when they believe that there is probably a better and easier solution.

4 Who Are the Right People?

Occam's razor: **a scientific and philosophical rule that entities should not be multiplied unnecessarily** which is interpreted as requiring that the simplest of competing theories be preferred to the more complex or that explanations of unknown phenomena be sought first in terms of known quantities. (Merriam-Webster, 2023)

In other words: The simplest answer is usually the right one.

In this case, it means:

- Find your domain experts.
- Ask them for guidance.
- Listen to what they say / Do what they tell you.

Let's re-run the previously described scenario, only this time the right people get brought into the room:

A need is identified: "We need a widget that stores data from a web page!" Marketing has done all the research to prove that the addition of this widget would be a valuable feature for the current product line. Sales swears up and down that people have been asking for this thing for years and they'll be able to sell it to every customer the company has ever had and, what's more, it will bring in new clients in droves. It is the PERFECT idea, and all that is needed is a Dev Team to build it!

A Discovery Team is created to research and design the new product. They bring in a Development Lead, a QA Lead, a Support Lead, as well as a Technical Product manager, UX designer, a Product Manager, and an Architect.

The Marketing folks pitch the concept to the Discovery team. The dev lead makes some suggestions about two or three tools that they might want to use to build POCs to help make a technology choice. QA immediately points out some of the problems that have been found in similar products the company has built in the past and makes suggestions for things to avoid. Support agrees with QA and, additionally, makes recommendations for features that will really help customers use the product more efficiently. The Dev lead spends 2-4 weeks building POCs and determining the best tool for the job as well as working with the Architect and the rest of the team to ensure that the design meets customer requirements. Marketing works with Sales and Support to determine the MVP requirements. At the end of the

“discovery period” the skeleton for an MVP is delivered to the development team along with an appropriate tool set. For the sake of argument, let’s say the tools were “WidgetBuilder 4.0” and a specialty plugin that allows WidgetBuilder to work seamlessly with React, which the developers are already using for their other web apps. The team is then provided the leeway to develop a full solution in a way that makes the most sense to them while still creating the product that the customer needs.

When the MVP software is delivered, it works correctly, doesn’t contain kludges, and doesn’t require a re-write in 6-12 months. Instead, the team can take feedback from the business and incorporate it into future product versions along with new features.

In the real world, we don’t know which product took longer to develop. For purposes of comparison, let’s say that both took the exact same amount of time – 3 months. The difference is that, at the end of 3 months, one team had a product that was minimally functional but needed to be re-written almost immediately and may or may not have been optimally performant. Although the other team also had a product that was minimally functional, it didn’t need to be re-worked. Instead, the team and widget were ready to move forward with new features. There may have been a little extra money spent on the front end of the project by getting technical folks involved early, but a LOT of money was saved at the end of the project because there was no need for a costly re-write.

Even though the scenarios that have been described here were clearly manufactured to prove a point, both are based in real experiences I have had or have been directly associated with. I have felt the pain of working on a project I knew was doomed from the beginning because the right people weren’t in the room during the first conversations where key decisions were made. Seemingly simple concerns like where a particular kind of data will be stored in the database or what database to use can have widespread and devastating consequences for developers and testers later on. I can say with authority that it is (and was) incredibly demoralizing to work on a project knowing that there was an easier or better way available if anyone had asked and/or listened to the right people.

From the perspective of the company, although a demoralized team is certainly a problem, the bigger issue with this type of scenario is the monetary loss. I believe I can safely claim that Shareholders and Board Members want to see a profit. Usually, they want to that profit to be as large as possible. Having the wrong people in the room can cost a lot of money.

5 The Cost of Putting the Wrong People in The Room

For the sake of argument, let’s put some numbers around the two scenarios that I’ve described. The assumption that I’ve always made (and have heard others make many times) is that the average Senior Engineer is more expensive than the average Marketing person who often ends up making the critical decisions for a project. Interestingly, a little bit of research reveals that assumption to be untrue or, at least, not necessarily true. The table below contains some salary data courtesy of Glassdoor.com. I used the salary tool at https://www.glassdoor.com/Salaries/index.htm with the title listed in that cell and the location of “Portland, OR” to find the average salary for that job. (NOTE: Glass Door is continually adding to its salary database so new research may not yield exactly the same results.) Additionally, a quick consult of Google shows that, on average, there are 2080 working hours in any given year. (Zoe, 2023) That information was used to calculate an hourly wage for each job title for ease of comparison. (Yearly Salary divided by working hours.)

Title	Yearly Salary	Hourly Pay
Solution Architect	220K	105.77
Technical Product Manager	164k	78.85
Sr. Software Engineer	149k	71.63
Product Manager	130k	62.5
Sr. QA Engineer	134K	64.42
Delivery Manager	124k	59.62
Software Sales Rep	120K	57.69
Software Engineer	113K	54.33
Senior Technical Support Engineer	106K	50.96
Software QA Engineer	96K	46.15
UX Designer	96K	46.15
Business Analyst	90k	43.35

The data here is interesting. It turns out that that the old trope about the engineer being the most expensive person in the room isn't always accurate. Realistically, everyone in the room is expensive and the more time you spend with the wrong people in the room, the more money you lose. The exact amount of money wasted is a little bit academic since the hourly difference between just about anyone is minimal. With the notable exception of the Solution Architect, who is exceptionally well paid and, on the lower end of the scale, the UX Designer, Business Analyst, QA Engineer, and Support Engineer - and even they aren't inexpensive.

That brings us back to the question at hand – what is the cost when you don't get all the right people in the room?

Again, because I'm spit-balling a bit, we'll say that the "wrong" team that's designing this new project is composed of the classic representatives of a Discovery Team. That would include at least a Solution Architect, Delivery Manager, UX Designer, and Business Analyst. We will leave the Sr. Engineer, the QA and the Tech Support folks out because they're really busy and Engineers are always expensive.

Next, let's say that the project is laid out over the course of 10, 1-hour meetings. We'll also say that everyone on the team did an extra 20 hours of research on the concept. Applying basic math, we will see that getting this project put together without all the right people cost us $7646.68

Because you should always show your work, here are the numbers for the people that were included:

Title	Hourly Pay	x30 hours
Solution Architect	105.77	3173.08
Delivery Manager	59.62	1788.46
UX Designer	46.15	1384.50
Business Analyst	43.35	1300.64
Total cost per hour Included Team	330.29	7646.68

And here's the math for the folks that were excluded:

Title	Hourly Pay	x30 hours
Sr. Software Engineer	71.63	2148.90
Sr. QA Engineer	64.42	1932.60
Senior Technical Support Engineer	50.96	1528.80
Total cost per hour Excluded Team	189.01	5610.30

By excluding the Sr. Software Engineer, the Sr. QA, and the Senior Tech Support Engineer, $5610.30 was saved. But was it really?

The $7646.68 team designed the solution, complete with the tools to do the job and then handed it over to the development team to implement. The development team looked at the work and said that they felt like they could have an MVP ready in 3 months or, 520 hours times the number of people on the team. A reasonable approximation of reality would be that, even though the team worked on this for 3 months, 3 weeks of that time was spent in meetings and other activities. The more realistic number of hours is probably closer to 400. Using a standard "Two Pizza" team size (Hern, 2018), a reasonable group composition would look something like this:

Title	QTY	Hourly Pay	X400 hours
Sr. Software Engineer	1	71.63	28,652.00
Sr. QA Engineer	1	64.42	25,768.00
Software Engineer	4	54.33	86,928.00
QA Engineer	1	46.15	18,460.00
Totals	7	236.53	159,808.00

Another fair assumption for this exercise is that the Product Owner, UX, and other people are outside of the team and are not dedicated full time to the project, so they won't be included as part of the cost.

In this scenario, you spent $159,808.00 on the engineers to build and test this widget – assuming everything went according to plan. However, as laid out in section 3, it's likely that things didn't go right. There are multiple potential failure scenarios. Here are some of the more common ones that should be considered:

1. The tools didn't work as advertised and caused the team to need extra time to complete the MVP.
2. The tools weren't appropriate for the job and, although an MVP was shipped, the team ended up rebuilding the widget with a different tool set after a few months of failures in the field.
3. The solution, as designed and provided to the development team, wasn't technically appropriate and required extra work to implement.
4. All of the above scenarios.

For scenario 1, let's be kind and say that the project only slipped 2 weeks or 80 hours since the engineers were able to figure out how to manipulate the tools to do what was needed. That's an extra $31,961.60 that was spent. If the excluded engineers had been in on the design sessions, it's likely they would have caught the tool problem by taking the time to do POCs. Your $5610.30 savings turns into a $26,351.30 loss.

In Scenario 2, the best-case is that the team spends another 400 hours rebuilding the widget with a more appropriate tool. It is possible that the second attempt took a little less time, since the team had a better idea about how they wanted to build things, but there's still the added cost of another tool. It is also likely that there was some ill-will generated with customers since Widget V.1 didn't work very well and wasn't very performant. (Had the product realized expectations, there would not be a need for a re-write.) However, for the sake of simplicity, we'll say that it cost the same team 400 hours. That's another $159,808.00 minus the $5610.30 you saved by keeping the expensive engineers out of the room. When math is carefully applied to the problem, that leaves you $154,197.70 in additional costs.

Scenario 3 could be a relatively quick fix – perhaps costing as little as a week or two. However, depending on how inappropriate the proposed solution was, it could also lead to many weeks of struggle for the team. Again, using the least damaging version of events, we can re-use the numbers from the first scenario and say that the inappropriate design cost 2 weeks, or an extra $31, 961.60. Even with a fast fix, the company is out a lot of money!

It seems un-necessary to do the math for Scenario 4, since it's clear that the loss will not improve when all three situations are combined.

Obviously, these scenarios are simple and a bit contrived but, as stated previously, I've witnessed versions of these situations play out many times in both large and small companies). It cannot be stated with absolute certainty that things would have progressed more smoothly if the Engineers had been brought in early and fully included in the design and discovery sessions. However, I believe it is reasonable to say that having people in the room with deep knowledge of the existing infrastructure and software would not have made the situation any worse. The math clearly shows that a relatively small initial expenditure in engineering time during the design stage has the potential to save tens of thousands of dollars once the project moves into the development phase.

6 What the Right People Need to Succeed

Even when the "correct" people are included from the very beginning, they cannot just be turned loose with the expectation that miracles will be created. However, the opposite condition is also true – if a technical team is given a complete solution with no challenges, it is not reasonable to expect them to be excited and creative about the implementation process. The team will feel as if they've been reduced from engineers to "code monkeys".

If success is the desired outcome for the technical team, provide them with a problem and allow them to pitch potential solutions to the Business representatives. Using the earlier example, this is how the Discovery Team might frame the scenario for the development team:

- "We need a widget that stores data from a web page!"
- It needs to work with our existing website.
- Stored data needs to be accessible to both customers and internal users.
- Data should be always backed up in a secondary location.
- Request response time should be less than 1 second.

Clearly, this isn't a very complicated example, but it demonstrates the point. The desired product has been specifically described but without deep technical requirements. This is now a problem that the team can solve creatively. They may choose to build POCs with Cloud and Co-located environments so that responsiveness and cost can be compared. Once the team has accumulated some data and determined the best options, they can be presented to the Business for the final decision. However, it's the technical team's responsibility to define the potential solutions that they feel are most appropriate.

The result is that the Business gets the needed software, and the technical team is empowered with the freedom to design something that they are excited about creating and maintaining.

7 Implementation in the Real World

Any Software Development Lifecycle can Lead with Quality. For instance, in a Waterfall paradigm, the design process remains a specific stage that sits in front of the development portion of work. The difference is that a subset of the technical team is involved in the design stage instead of receiving the pre-defined solution from the business.

In a more Agile lifecycle, the development team may receive a simple skeleton of the design with the technology chosen and the initial work laid out along with a vision of where the product will eventually go. A potentially productive approach would involve the Product Owner coordinating between the business and the technical leads to lay out each new feature/solutioning challenge during the sprint before the one where implementation is needed.

Regardless of the methodology, it is critical to get the correct people involved from the very beginning of the process and, when the project is presented to the team, it must be done in such a way that the team is challenged and excited to build the product.

8 Conclusion

A healthy company culture abhors a large meeting – and rightfully so. The bigger the meeting, the longer and more difficult it is to make significant accomplishments. It is also more costly. But, when project planning begins by excluding the people who know the software ecosystem most intimately: Software Developers, Quality Engineers, and Support - the result is likely sabotaged before the first piece of code is written. However, if even a few of these folks are included at the initiation of the planning process and the resulting plan is constructed in a way that challenges and excites the team, the final product will likely result in saved time (and money) for the project, a happier team, and better quality software. All of which mean higher profits for the company.

9 Acknowledgements

I'd like to start by thanking my editors, Bhushan Gupta and Robert Sabourin, who are both absolute gems! They kindly and mercilessly provided great feedback that helped me transform this paper from a collection of random ideas into a coherent and (hopefully) interesting work.

I would also like to thank my husband, Justin, who still doesn’t entirely understand what I do, but supports me in doing it.

And, as always, my sincere gratitude goes to my employer NWEA/HMH and my manager, who consistently support and encourage my participation with PNSQC every year.

References

Bird, Jeremy. 2019. "How to improve users lives when users & customers are not the same people" Jeremy Bird Blog, entry posted February 13, 2019, https://jeremybird276.medium.com/user-vs-customer-its-not-mutually-exclusive-b7f954c80b1d (accessed July 25, 2023).

Cherry, Denny. 2019. "Having the Correct People on a Project Is Key to Success" article posted May 16, 2019, https://orangematter.solarwinds.com/2019/05/16/having-the-correct-people-on-a-project-is-key-to-success/ (accessed July 25, 2023).

Coursera. 2023. "What Is a Solutions Architect (and How Do I Become One)?" article posted June 15, 2023, https://www.coursera.org/articles/solutions-architect (accessed July 25, 2023).

Harris, Aaron. 2015. "QA Role in Product Advocacy" Envoc Blog, entry posted April 7, 2015, https://envoc.com/think/qa-role-in-product-advocacy (accessed July 25, 2023).

Hern, Alex. 2018. "The Two-pizza Rule and the Secret of Amazon's Success." The Guardian. April 28, 2018. https://www.theguardian.com/technology/2018/apr/24/the-two-pizza-rule-and-the-secret-of-amazons-success (accessed July 25, 2023.

Kaplan, Zoe. 2023 "How Many Work Hours Are in a Year?" Forage. July 11, 2023. https://www.theforage.com/blog/basics/how-many-work-hours-year#:~:text=The%20average%20full%2Dtime%20U.S.,weeks%20in%20a%20calendar%20year.&text=On%20average%2C%20there%20are%202%2C080%20working%20hours%20a%20year (accessed June 27, 2023).

Khalimonchuk, Kateryna. 2022. "Discovery Phase in Software Development: How it Saves Time & Money" Fulcrum Rocks Blog, entry posted April 17, 2022, https://fulcrum.rocks/blog/discovery-phase-software-development#:~:text=Discovery%2C%20(or%20scoping%20phase),%26%20scope%2C%20and%20identify%20risks (accessed July 25, 2023).

Merriam-Webster.com Dictionary, s.v. "Occam's razor," https://www.merriam-webster.com/dictionary/Occam%27s%20razor (accessed June 27, 2023).

Muzyka, Bohdana. 2021. "Project Discovery Phase in Software Development (Step-by-Step Guide)" TechMagic Blog, entry posted July 29, 2021, https://www.techmagic.co/blog/project-discovery-phase-in-software-development/ (accessed July 25, 2023).

Puzhevich, Victoria. 2022. "5 Steps for Choosing a Technology Stack for Your Project" Scand Corporate Blog, entry posted May 10, 2022, https://scand.com/company/blog/choosing-a-technology-stack/ (accessed July 25, 2023).

Testim. "What Is Shift Left Testing? A Guide to Improving Your QA" Testim Blog, entry posted June 11, 2021, https://www.testim.io/blog/shift-left-testing-guide/ (accessed July 25, 2023).

Wikipedia. 2023. "Sunk Cost." Wikimedia Foundation. Last modified August 13, 2023. https://en.wikipedia.org/wiki/Sunk_cost (accessed August 14, 2023)