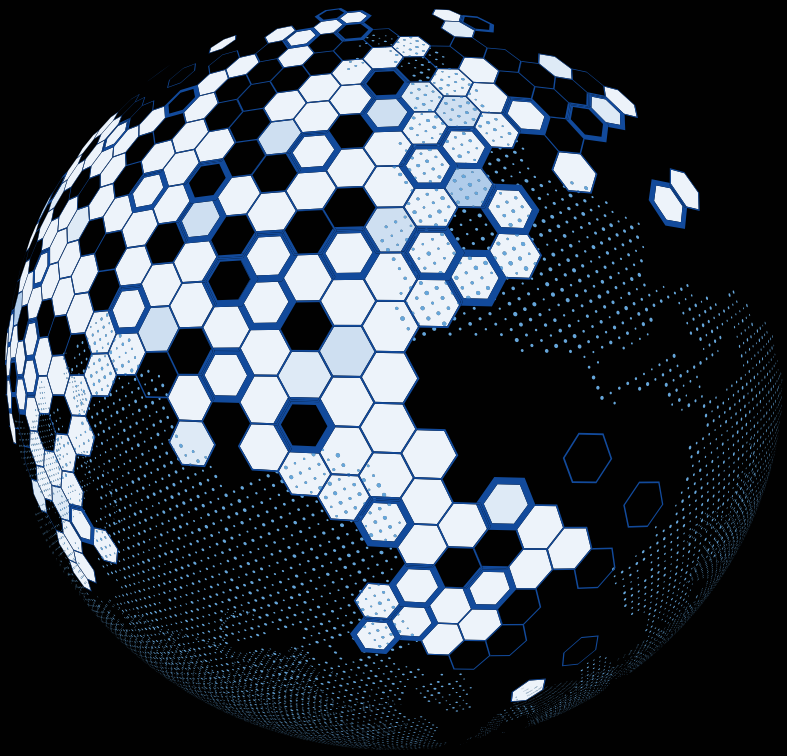
Deloitte.

Deloitte Quality and Test Engineering | White Paper



Your bad requirements are costing you money

Author: **Anjalie Rajkumar**

January 2022

MAKING AN IMPACT THAT MATTERS since 1845

On a Thanksgiving episode of Friends, Rachel famously tries her hand at making a Thanksgiving trifle. But alas, with two pages of the recipe book stuck together, Rachel unknowingly combines half a trifle and half a shepherd's pie.

Requirements are misconstrued in everyday life—in this instance it only cost Rachel her reputation as a "chef", but requirement issues on your next programme will cost a lot more.

Bad requirements are everywhere

I'm a Quality Engineer (for those of you stuck in the dark ages, that means tester). Don't click off the article just yet, I promise it's worth the read.

As a Quality Engineer, I'm used to continually finding myself in situations on programmes where my team is pushed for time, close to launch date and with a list of critical outstanding defects growing by the day. This happens frequently enough, but on a certain engagement that we found ourselves on a few years ago, frustration levels were high among the testers, clients, and leadership. We found defect after defect and were inching closer and closer to our Go-Live date. An estimated 19 hours per day were spent addressing requirement-related defects, across the 50-person team, approximately 4,800 hours per year. This factors in the time spent on activities such as raising, reviewing and resolving requirement-related defects.

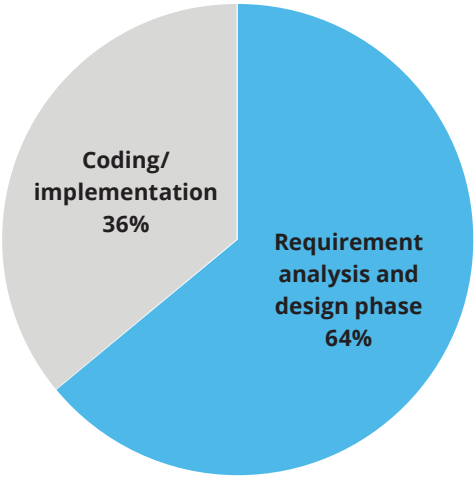
When raised with some colleagues at Deloitte Quality & Test Engineering (QTE), we observed that testers spent a significant proportion of their time clarifying requirements—an additional role on top of testing. This requirements clarification was conducted often when BAs had rolled off projects, original documentation had been archived in an unknown location, and when project timelines were at their tightest.

Does this sound familiar to you? Bear with me here, this isn't another shift left paper, we aren't arguing that testing earlier will fix this, we are proposing something else.

It's very common on programmes to see Requirements/User stories as the root cause for a significant proportion of defects. According to Crosstalk, the Journal of Defense Software Engineering, "most failures in software products are due to errors in the requirements and design phases—as high as 64 percent of total defect costs" (Figure 1). This could be attributed to ambiguous, stale/out of date requirements and/or technical and business requirement mismatches.

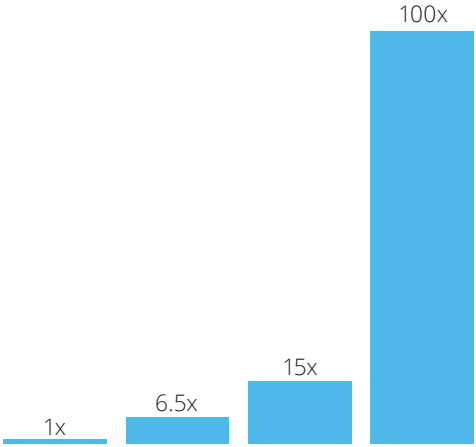
Origin of software defects
(Source: Crosstalk, the Journal of defence software engineering)

Figure 1



Relative costs to fix software defects
(Source: IBM systems sciences institute)

Figure 2



Phase/stage of the S/W development in which the defect is found

So, we know that requirements defects are prevalent on projects, but does that matter?

Bad requirements cost money

In the example above, we estimated that the project could have saved 10% of their annual testing spend through eliminating defects posed by poor or dated requirements. We're not alone in believing that significant money is wasted on fixing defects later in the software development lifecycle.

According to the Systems Sciences Institute at IBM, it is 15 times more expensive to fix a defect in Testing than in Design (Figure 2).

Furthermore, a publication from NASA highlighted the potential for cost escalation throughout the project life cycle, citing "the cost of fixing a requirements error discovered during the requirements phase is defined to be 1 unit… at the integration and test phase, the cost to fix the error becomes 21—78 units" (Haskins et al., 2004).

There is a clear consensus that finding Requirements defects later in the project lifecycle leads to notable costs (though the exact order of magnitude is up for debate), not to mention wasted time.

How do we fix that?

Prevention is better than cure

There are various approaches to improving the overall delivery approach. We see many concentrating on "reviewing" each phase of the lifecycle, with a view to increase efficiency. Even with programmes using an Agile delivery approach, utilising sessions like "3 amigos" without proper frameworks, we still see significant defects.

Author, Dr Joe Marasco, cites that once a formal requirements document has been produced, of any kind, stakeholders lose engagement and won't read/update when changes need to be made (see Figure 3). This isn't limited to a standard waterfall Requirements Document, this also includes formalising user stories as tickets on JIRA, for example. It applies to Agile ways of working too.

Figure 3

Therefore, it becomes apparent that the right time to intervene for maximum impact is during the Requirements Elicitation phase.

So, what do we know so far?

1. We know Requirements contribute to a large proportion of defects on a programme.
2. We know that the cost of finding defects increases exponentially as we progress through the project lifecycle.
3. We know the best time to engage parties for Requirement clarification is during the elicitation phase.
4. We know principles of Shift Left static analysis are helpful, but don't offer a complete solution.

Introducing the Requirements Engineer

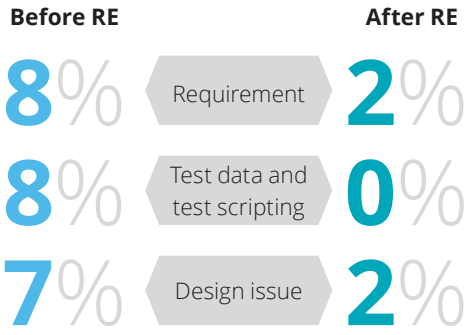
We hypothesised that bringing on a tester to "quality assure" requirements/user stories through the requirements gathering/discovery phase, would enable testing to have an input into user stories much earlier in the process and would allow us to consider, from the earliest stages, the testability of said user stories.

It was a nice theory, but would it work? How would we measure the benefit? How would it work with various delivery models?

So, we took it upon ourselves to try it. We found ourselves a willing Public Sector (Agile) project and we conducted an as-is analysis to understand the proportion of defects attributed to poor requirements. Here are the stats:

Proportion of requirement-related defects by Root Cause
(before and after a Requirements Engineer)

Figure 4



The statistics were compelling.

In 6 months, the Requirements Engineer had implemented processes to ensure the information passed to the test team was clear and testable. In turn, this enabled test scripts to be generated efficiently and improved the quality of the solution.

Could BAs not do this themselves?

BAs and Testers play two different roles in a delivery programme. They are two different types of people for a reason; it's not necessarily reasonable that BAs should anticipate everything a tester might need for a successful test, and like with any message, things get lost in translation.

However, that doesn't mean we can't invest in setting up processes between BAs, Devs and Test teams to clearly communicate expectations of, and quality check, requirements, with a view to remove the Requirements Engineer once their work is done.

Alternatively, we can retain the Requirements Engineer for the duration of the programme, whilst the BAs roll off/move onto other projects—this ensures continuity without ringfencing BAs for the entire lifecycle.

Tried and Tested

Testers by nature, we sought another project, this time, in Financial Services (specifically, Banking). We used a similar approach to the Public Sector project, varying slightly based on team size and platform complexity. We successfully reduced testers' time spent on clarifying requirements from 30% to 10% of their day.

After our initial success, we decided to embed ourselves further into the project. We managed to streamline the 3 amigos process, establishing what a successful 3 amigos looks like so all parties know what the expected standard of a requirement should be. As result of this, we improved the communications, traceability and overall efficiency of requirement analysis.

Since then, we have also used Requirements Engineering on another Public Sector project, this time for two months to implement processes to enable better communication and traceability between teams. In addition, we are also showcasing our practitioners' subject matter expertise on another Financial Services project, this time Wealth Management. The benefit of bringing on a Requirements Engineer with industry/technology expertise at project conception ensures robust requirements and continuity right to the point of implementation.

What this means for you

Let's return to our first example—the (Waterfall) transformation programme. The poor/out of date Requirements cost the programme 10% of their annual Test spend. For a similar engagement, net the cost of two Requirements Engineers, the programme can save approximately 6% of the annual test spend per year if we commit to reducing (not eliminating) said requirements by just 50%. In some cases, this could amount to a six-figure saving. That feels like a worthwhile investment.

For those wanting to invest in improving processes in the long term, a Requirements Engineer can be integrated into the programme for the duration of the project lifecycle to assess, implement and enforce better practices. Whilst this would require the biggest investment, it would ultimately lead to the largest cost saving.

For those who are looking to reap benefits in the shorter term, a Requirements Engineer can be brought onto the programme for a negotiable fixed period, conducting a review and implementing process improvements to then hand over to the programme team to enforce.

Lastly, for those who want a light touch approach, a Requirements Engineer can provide coaching and a checklist for a limited period, providing you with a toolkit to implement better practices within your programmes.

Beyond FTE savings, there are subsequent positive externalities of introducing a Requirements Engineer. For example, the consequential savings of reducing the timeline of a project. In addition, Requirements Engineering can also help bring Environmental, Social and Governance (ESG) considerations to the forefront of your delivery. Investing in a Requirements Engineer will set up Requirements to be in a fit state prior to the Testing phase, thereby reducing the amount of time required in Test Environments. Reduced time in Environments leads to a lower carbon footprint and more sustainable working practices.

Ultimately, Requirements Engineering will improve the E2E efficiency of the delivery lifecycle. By quality assuring Requirements, we will reduce cost, increase quality, and cut down time to market. In a world where so much of our inefficiency is attributed to poor requirements, Requirements Engineering is the next logical step in improving all our delivery models.

Thanks for sticking with me, if you want to explore more feel free to reach out to me or anyone in our leadership team (we love a good debate):

Nicholas Yap
Mimi Taylor
Anjalie Rajkumar

References:

Segue Technologies. (2014). The Rising Costs of Defects. [online] Available at: https://www.seguetech.com/rising-costs-defects/ [Accessed 1 Nov. 2021].

Haskins, B., Stecklein, J., Dick, B., Moroney, G., Lovell, R. and Dabney, J., 2004. 8.4.2 Error Cost Escalation Through the Project Life Cycle. Nasa Technical Reports—*INCOSE International Symposium*, 14(1), pp.1723-1737.

StickyMinds. (n.d.). What Is the Cost of a Requirement Error? [online] Available at: https://www.stickyminds.com/article/what-cost-requirement-error [Accessed 1 Nov. 2021].

Deloitte.