Building a Modern Quality Program from the Ground Up

Jeff Sing
jeff.sing@iterable.com

Abstract

As startups mature, one of the biggest trends is the decision to build a Quality Program from the ground up. However, these smaller, agile, and more DevOps-centric companies aren't looking to hire a horde of testing specialists who act as a safety net—waiting for code to be tossed over the proverbial wall by developers to perform validation before a release is completed. Instead, engineering leaders are looking for a modern Quality Program that is significantly smaller and more specialized. They want Quality Leaders that rely on technology, push a concept of "Quality Culture," and are Quality Program Managers to ensure that engineering organizations successfully deliver value to their customers.

Biography

Jeff Sing is a Quality Leader who has been in the testing industry for over 15 years. During this time, he has built test automation frameworks, tested strategies, and executed quality initiatives for fields such as medical devices, infrastructure security, web identification, marketing tech, and experimentation and progressive delivery.

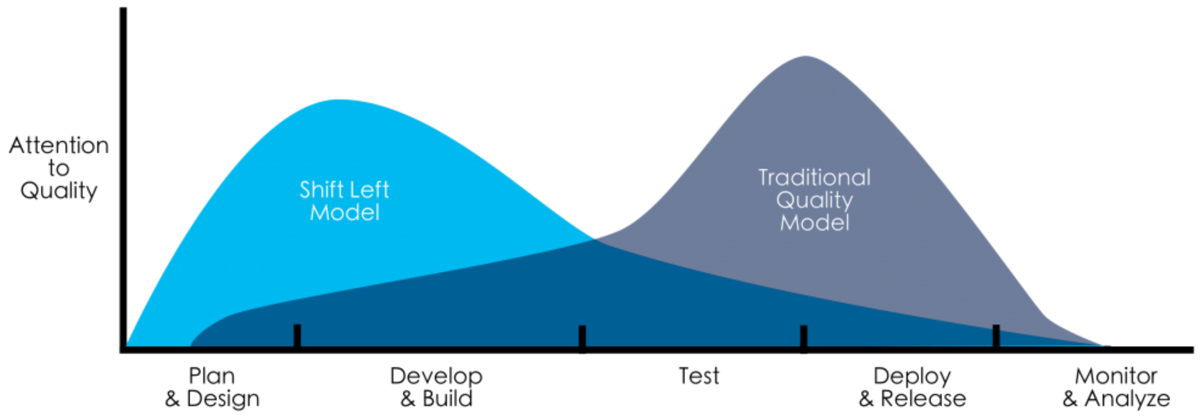
Jeff is currently a Sr. Engineering Manager at Iterable where he has both built and leads the Quality Engineering and Engineering Operations teams. Prior to joining Iterable, Jeff built the Quality Engineering team at Optimizely where he was both the Software QE Manager and Chief of Staff to the CTO.

Jeff currently resides in Redwood City, California along with his wife and three children. Jeff studied Computer Science at UCSD, and to this day is still a passionate Chargers fan.

1. Introduction

One of the more interesting trends I have noticed perusing job boards during the last five years was the proliferation of companies looking for new QA Managers or QA Leads who would either build their "Quality Engineering" program or significantly revamp their current ones. When talking to these engineering leaders at these companies, they were clear that they were looking for an individual to be some amalgamation of an Automation Architect, Testing Coach, and a Quality Agile Leader that could "improve quality." But the catch was none could clearly describe how they expected quality to improve (Better test coverage? New automation framework? Quality Best Practices?), nor did they really understand how these quality initiatives would actually impact quality improvement (eg: Does more automation equate to fewer bugs? Does this mean you will have happier customers? What if the trade off is slower deployment speed, how does this affect customer satisfaction now?).

This struggle can be attributed to what software quality programs used to be like, where companies staffed teams of testing specialists who act as a safety net, waiting for code to be tossed over the proverbial wall by developers to perform validation before a release can happen [1]. These software quality engineers were responsible for testing the application and finding defects.



Shift Left is about doing things earlier in the development cycle. Source: van der Cruijsen 2017.

However, modern agile stresses velocity paired with modern DevOps practices in Shift Left (*an approach to software testing in which testing is performed earlier in the development cycle*) to attain validation. This shift requires developers to bake in testing inherently into the cycle of software releases as they are built, rather than in the end stage pre-deploy. This motion of testing earlier and often and automatically on each code build, paired with maturity in deployment frameworks that allow you to quickly deploy bug fixes to production, made engineering companies re-evaluate if they needed testers at all to perform validation in the latter stages of the development cycle. "*The increasing adoption of agile and DevOps is minimizing the importance of QA for many teams because these ideologies focus on speed, and quality can become secondary.* (Mason, 2019)[2]".

In 2015, Yahoo eliminated QA by moving to CI/CD (Continuous Integration/Continuous Delivery) and having software engineers be responsible for code quality which forced them to develop automated testing tools and execute tests earlier in the pipeline[3]. Many software companies followed this model, cutting a lot of their testing team, or relegating their testing team to be pure automation groups [4]. Quality (requirements, test scenarios, and creation of test cases to be automated) was owned by all developers and no longer a QA team.

However, even with this change, engineering organizations were still plagued with quality issues. What they found was the correlation between having a lot of automation tests didn't always translate to answering the question "Are my customers really enjoying my product and is it of high quality?" [5]

This question, in fact, has become more critical in how successful engineering organizations function. In the World Quality Report in 2011-22, the top quote in the executive summary was "*Most CIOs now value testing more than ever before, and the onward march towards digitization is ensuring that customer experience and quality are of utmost importance.*" [6]

To address how to build a Modern Quality Program that champions the customer experience and enables engineering to deliver with quality, the Modern Quality Program requires a leader that is able to combine both the technical leadership needed to architect the right tooling, and a Quality Program Manager that understands how to mature the Quality processes in that company's Software Development Life Cycle(SDLC). Intertwined with the above, the leader also needs to be able to build the entire operation to ensure that the tooling and process iteratively and pragmatically scales for the engineering organization at hand, and measure to ensure the program is not just solving the right problems, but solving them efficiently. Lastly, this engineering leader needs management savvy to be a solid people leader in order to scale and grow the quality organization to support the company as it expands.

2. The Four Key Roles when Building a Modern Quality Program

The Modern Quality Program is built on the balance of enacting the right quality program and process (Quality Program Manager) versus executing the right automation tooling (Automation Technical Architect). To determine if either programs are fit-for-purpose requires the Quality Operations role. Lastly, none of this is sustainable as a one-person organization if the company scales and grows. The fourth role (Engineering Leader) is essential to properly grow the correct team to carry on forward. Below are the definitions of each key role.

Quality Program Manager: Implements the Quality SDLC Program, driving the engineering organization towards its Quality North Star. By building and enforcing quality guard rails, the Quality Program Manager remediates the friction of failed orchestrations or confusions during the design and build phase, and, instead, allows engineers to focus more on the implementation of their features.

Technical Architect: Visionary of the technical roadmap of tools, frameworks, and QAOPS (the integration of testing and QA with releases during CI/CD) needed for engineering to ship with high velocity. The Technical Architect creates the partnership with engineering to ensure there is a full spectrum of sustainable automation across the testing pyramid that guarantees high feature coverage.

Quality Operations: Identifies what KPI/Metrics represent Quality, and reviews if the engineering organization achieves its goals. Ensures that the tools and processes implemented are actually efficient and effective.

Engineering Leader: Builds and grows the team to execute all of the above. Ensures that said team works well together, and that each individual member is supported in reaching their career goals. The Engineering Leader balances between validating that the output of your team's current outcome is good while setting up the team for great outcomes in the future. [7]

3. The Quality Program Manager Role

The Quality Program Manager creates and coordinates the various quality programs and processes across the engineering organization. To accomplish this, the leader needs to establish Program Vision, Quality Coaching, and Customer Advocacy.

3.1 Program Vision

Vision can be described as your quality North Star.

Questions I like to pose:

- *What is the charter of the organization that you are trying to build?* I always tie mine to the company's vision statement. At Iterable, my Quality Charter is: To ensure our engineering team is enabled to deliver joyful customer experiences for every organization in the world.
- *What is the stack rank of things that you need to accomplish?* What are the critical areas that I see damaging the ability for us to fulfill our promise of delivery of joyful experiences? This is important because understanding what your customer truly cares about allows you to prioritize. If your customers rely on your product's up-time versus utilizing the application's user interface, then my highest priority for quality will revolve around ensuring our product is available
- *How are you going to go about achieving this vision?* You've already established your roadmap, now you need to define your program milestones. Defining your program milestones and reporting on the velocity of how you are doing to your stakeholders keeps you on track.

Starting a new Quality Program is a lot like being thrown into the middle of the ocean and told to swim to shore. On top of this you will be crashed upon with waves of requests from different stakeholders looking for help. Having a clearly defined vision allows you to steer in one direction by determining what your quality program needs at this time versus what doesn't fit at this juncture.

3.2 Quality Coaching

One of the most critical roles of the Quality Program Manager is Quality Coaching. Coaching is defined by the International Coaching Community [8] as an individual who provides guidance to a client on their goals and helps them reach their full potential. In the same vein, Quality Coaching is helping your engineering organization and leadership team achieve their quality goals. I divide Quality Coaching into two categories: Quality Leadership and Thought Leadership.

3.2.1 Quality Leadership

Quality Leadership revolves around how to establish the right process and programs that will yield better outcomes. This involves understanding the current engineering SDLC and identifying where Quality Artifacts and Programs should fit in (eg: Who should be building a Test Plan if we have no QA engineers at this moment? What does the Test Plan look like? How does it affect the release process?) and working with engineering leadership and developers to officially be part of the SDLC (eg: Training teams to write a Test Plan, Auditing that all releases have a Test Plan).

3.2.2 Thought Leadership

Thought Leadership determines the ROI of each quality process being implemented. Depending on the maturity of the engineering organization, it may not always make sense to implement every standard QA process. On the other hand, as an organization matures and scales, some older practices should be sunsetted. An example of this would be an organization that didn't have automated tests and required all its engineers to manually validate features prior to a deployment. Once a suitable framework that gave good feature coverage was implemented, the act of manually testing probably would not result in a good ROI.

3.3 Customer Advocate

A lot of testing and validation is built around answering the question "Did I build the product right?" But engineering often relies on Product Management to answer that question. This essentially becomes a single point of failure, since engineers are relying purely on Product as a gatekeeper. One of the essential functions of a Quality Program Manager is to act as a Customer Advocate throughout the entire SDLC.

Some examples:

- Partnering with the Product Manager to review the Acceptance Criteria and Workflow Scenarios (which eventually will become Test Cases) through with your GTM (Go to Market) organization. Getting feedback from Solution Engineers and Account Executives over certain features and how customers are most likely to use them gives you insight on what to test against.
- Inviting Technical Solutions Engineers or Customer Service Managers into release Bug Bashes to give you a customer's perspective of how your new features will fare. This often catches usability bugs that would become major issues later on.
- Meeting with the employees that support, sell, and manage your customers to understand their biggest pain points or complaints with the product. This helps you determine where to harden your testing for new features.

4. The Technical Architect Role

The Technical Architect Role revolves around partnering with engineering to drive best automation testing practices, and partner/build/lead the charge in creating/maintaining testing frameworks across the application and platform it's built on. To accomplish this, the leader needs to establish Technical Leadership and Automation Mentorship.

4.1 Technical Leadership

Technical Leadership encompasses not just the testing pyramid but the infrastructure the tests are built on as well as the environments in which it runs. A technical leader needs to give direction on the following:

- How good is my unit testing and what should our code coverage be? (Is this important to us or not?)

- Do my developers utilize integration testing? Do they need help in developing this? (Who should maintain this framework?)
- Who is currently writing end to end tests? (Who maintains these? Do developers and QE both write them, or is this the role of your QE team?)
- What environment are we testing in? (Do I have useful test data?)
- Is my current CI/CD pipeline running my tests? (Are they flakey and causing issues? If so, who should be fixing these?)
- How do we test in production thoughtfully? (Who is managing my Feature Flags?)
- Should I build or buy tools for visual testing, AI/ML testing, crowd testing? (What is the integration cost really to stand these up?)

4.2 Automation Mentorship

More likely than not, if you're building your Quality Program you don't have a team, or, if you do, it's very small and unsustainable for building all the automation testing. Even if you had a large Quality team, engineers should be participating in writing automation tests. Automation mentorship revolves around building the programs to help uplevel engineers in writing good automation. Here are some examples of programs that really help mentor engineers in building good automation:

- **Testing Guild or Testing Community of Practice**: Weekly or Bi-Weekly meeting of engineers working on testing best practices or tooling. Engineers that attend bring these lessons with them back to their individual teams.
- **Quality Champions**: Identifying individual engineers who want to engage quality initiatives and find ways to encourage or empower them. At a previous company we had a developer who built a proof-of-concept Flakey Test Detector. His solution would have decreased the time to build in our pipeline since we had less failures. We found time for him to take a few sprints to productionize his tool and roll out to the rest of engineering. The end result was that we saw a 30% improvement in failures and 60% faster deployment speeds.

5. The Quality Operations Role

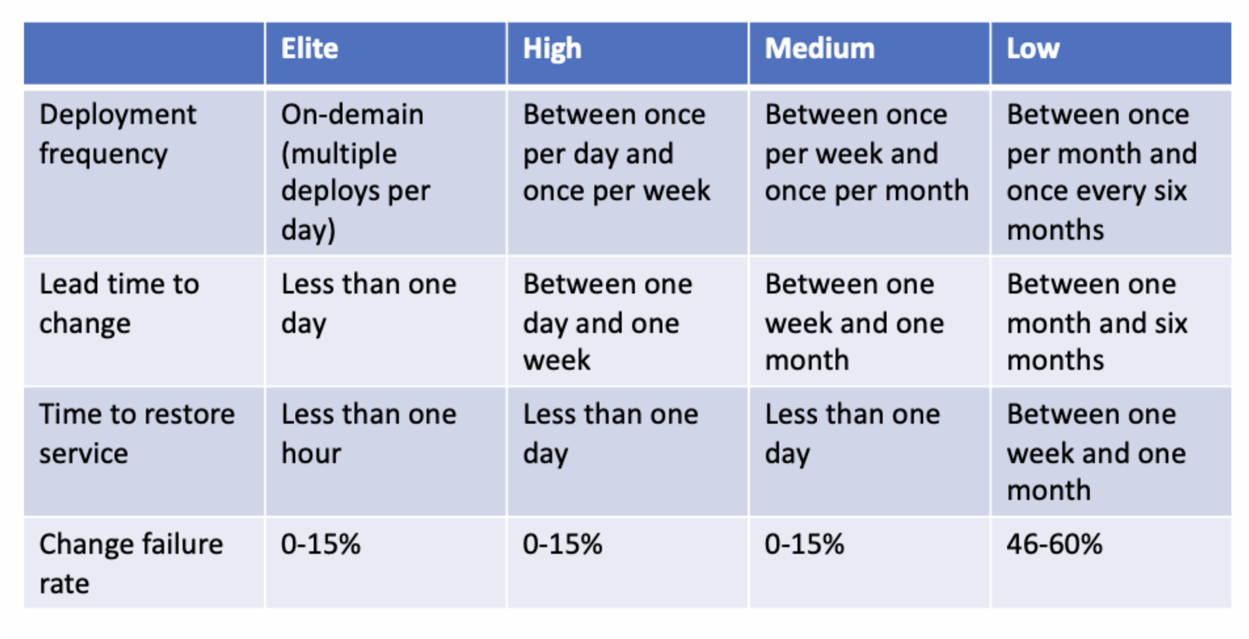
Quality Operations involves identifying the indicators that we want to monitor and measure to determine our system quality and customer satisfaction. This allows us to utilize data driven hypotheses on what adjustments we should make to our quality programs or technical initiatives to achieve better outcomes. Quality Operations runs in a cycle where we constantly review our KPI and plan out our next initiatives.

5.1 KPI and Metrics

I like to use a mix of KPI and metrics to measure quality. As with all metrics, it's important to view them collectively to understand what is happening with the system as opposed to using them individually. I also want to introduce a caveat, the KPI and Metrics I highlight on the bottom are different than your traditional QA Metrics (eg: escaped bugs, test coverage, test reliability, time to test, time to fix, etc) because depending on your program you are building, these might be prioritized or targeted differently. Some companies will absolutely care and set high SLA, while others may have other more pressing challenges and not prioritizing these at all. Therefore I like to pick metrics that tell a more visual story of overall engineering health and effectiveness which rises above simple quality metrics. I break these up into two categories: KPIs that are benchmarkable across multiple engineering companies, and KPIs that link quality with business success.

5.1.1 KPI and Metrics that are Benchmarkable

The following metrics like Change Failure Rate and Mean Time To Restore allow me to benchmark against other companies to see how far we are faring [9].



	Elite	High	Medium	Low
Deployment frequency	On-demain (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time to change	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service	Less than one hour	Less than one day	Less than one day	Between one week and one month
Change failure rate	0-15%	0-15%	0-15%	46-60%

Key Metrics of the Dora Benchmark

- **Change Failure Rate**: For the primary application, what percentage of changes to production resulted in degraded services that subsequently required remediation like a hotfix or rollback? I use this benchmark to determine how effective our system is in prevention of major issues per deployment.
- **Mean Time to Restore (MTTR):** Time to restore service that causes failures. The longer the outage the more unhappy our customers are which is an important factor. MTTR gives insights into code quality, reliability, and stability. Proper quality processes can help improve on these all and can reduce MTTR.
- **Lead Time for Changes:** How long does it take to go from code-committed to code running in production. As our testing framework is often part of this calculation, it's important to see if we are negatively impacting this time or not. A low lead time can affect incident response time and will lead to a drop in developer productivity.

5.1.2 KPI and Metrics that link Quality with Business Success

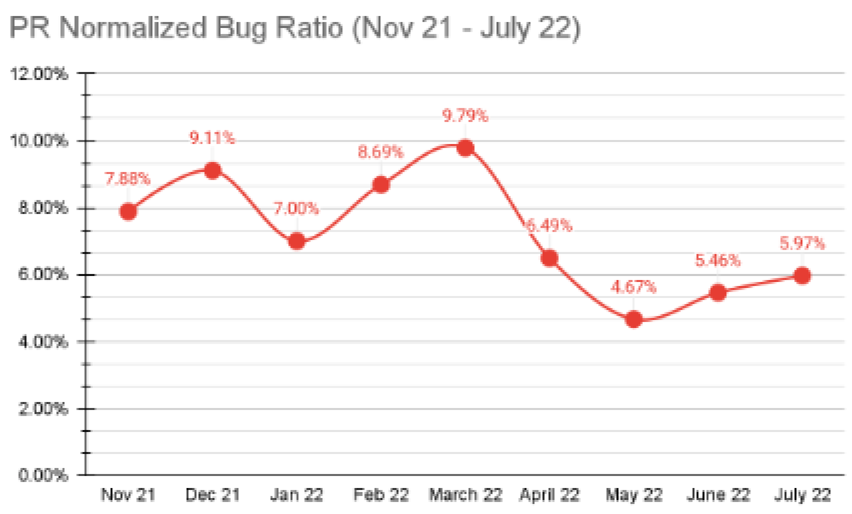
The following metrics are some that I have found measure quality in the view of business success. Every company I have implemented these at have had different formats and values, so they aren't benchmarkable like the last section.

- **Normalized Quality Ratio**: The number of incoming customer reported bugs versus the number of outgoing pull requests. This allows me to get context in what our bug counts mean. I could understand if we delivered a lot of features and had a lot of incoming bugs, but I would be concerned if we had a lot of incoming bugs and shipped no code. Generally if we had the latter, this is symptomatic of severe technical debt and is a great indicator that the costs to just Keep the Lights On (KTLO) is a risk factor that needs to be addressed.

- **Time to Close High Severity/Mid Severity Bugs**: Depending on if we had an SLA, this data is important to collect. The business contract, either internal to customer success or external to our customers, needs to be upheld. This data allows us to help drive prioritization over future work.
- **Customer Net Promoter Score (NPS)**: Depending on how your Customer Support team collects this information, I like to see how happy my customers are using our product. No matter how many bugs you close, if your customers are miserable, then you aren't winning. The caveat here is NPS can be influenced by a lot of factors and not feature quality, so the signal here isn't always as important.

5.2 The Quality Operation Cycle

1. **Data Collection**: Identify what Key Performance Indicators (see section 5.1) you want to monitor to determine the quality of your engineering output.. Identify how to collect this data and begin collecting.
2. **Visibility**: How do you want to share your findings, and how do you tell the story of what the data is showing you? I used a combination of monthly Quality Reports and a Quarterly Service Delivery Review to go highlight our findings.
3. **Implementation**: Drive discussion amongst stakeholders around findings. Partner with stakeholders to create engineering initiatives to implement action items in addressing issues discovered.
4. **Governance**: Build into place a system to hold the stakeholders accountable. These could be dashboards to show progress, or error budgets that trigger to force a code freeze.
5. **Reflection and Analytics**: Understand if the initiatives actually made a difference. If not, how should we iterate to perfection? Were we looking at the wrong data? What learnings should we take as we make changes? All of these should be applied as we start back at step one. One example seen at Iterable was we benchmarked our PR Normalized Bug Ratio over a course of a quarter and noticed over 30% of incoming bugs were related to one area of our code base. We saw that the new code was breaking existing functionality, creating regression issues. The Quality Engineering team created an initiative to increase test coverage in that area of code by 275% (8 tests specs to 22). The Quality Engineering team also required a launch checklist and combined bug bashes before teams could release. After a quarter we saw a dramatic decrease in defects from new code in that one area and an improvement in regression. Looking at our PR Normalized Bug Ratio, we see that we are still shipping features at the same rate, but the incoming bugs have improved, which ties together our initiative with real time results.



Tracking to see if our new testing initiatives improved our metrics. PR Normalized Bug Ratio is defined in 5.1.2 and the lower the value the better quality exhibited.

6. The Engineering Leader Role

While to be successful in the Engineering Leader role is not much different than being a successful engineering manager in general, there are a few caveats that I have found to be essential to consider:

- **Thoughtful Team Building**: Modern Quality Programs tend to be very lean agile teams. This means that while you may need an engineer that has Mobile testing experience, hiring a specialized engineer may not give you the most bang for your buck. Instead, figure out what can be trained and what skill set is the minimally viable for success.
- **Security Champion**: Partner closely with the security leadership to embed a lot of the security improvements into your roadmap. As the world improves connectivity, modern "bad actors" have become more savvy in penetrating applications. This creates a huge business requirement for greater security and resilience. This requires earlier testing and QA which requires closer partnerships. [6]
- **Soft Power**: Modern Quality Programs rely heavily on other organizations to accomplish compliance and engineering resources. You will need to be comfortable utilizing soft power to accomplish your goals
- **Swiss Army Knife Job Roles**: While building a Modern Quality Program, there will be times when the required solution is outside your job description, but still heavily impacts quality. To be successful, you may have to build the solutions in order to enable your own success (be comfortable unblocking yourself). It isn't uncommon that while in this role you may have to become a Jira Admin, Technical Program Manager, Product Manager for certain internal tools, Release Manager, or a Developer Experience Champion for initiatives that help enable your engineers.
- **People Manage**r: A lot of the above roles I mentioned end up leaning very heavily into the Individual Contributor work sphere. It's essential that to grow a successful Quality Program, your program has great staff to execute your quality vision. Things to remember:
    - **Empower them to execute in their role:** Remember from above, you aren't building a program of testers. You are building a program for quality. Ensure that your engineers are

spending the right ratio of time driving the quality program, coding automation, and measuring the success of their own actions.
  - **Delegate:** As your program grows, don't be afraid to delegate responsibilities to your engineers as you spend more time trying to scale your program. You won't be successful if you keep trying to do everything, eventually you will hit your breaking point.
  - **Mature and Grow your engineers:** Most likely going to be hiring and leading a team. Your responsibility as a people manager means that you need to grow your individual directs while setting them up for success. This might include ensuring they are getting the right technical training, mentorship on career growth, understanding their impact, or just ensuring they are motivated and satisfied in their role. Ultimately the success of your program depends more so on the people who will execute on your behalf and less on your pure execution alone.

7. Conclusion

As companies scale and grow, their engineering organizations need to be able to justify whether or not their outputs are achieving effective business outcomes. Building a Modern Quality Program is meant to help engineering define the correct outcome, put in place the right governance and guide rails to help developers efficiently and effectively deliver, and successfully scale the whole infrastructure as the engineering team matures and grows.

The greatest challenge isn't necessarily implementing a certain process or building the automation framework, it's determining if that process or testing will actually improve the business outcomes and growth against the tradeoffs of implementation. Using the correct data to justify the right decisions is the crux of deciding on which tasks the program should execute.

References

1. Heusser, M. (Retrieved on 2022, June 9) The future of software testing: How to adapt and remain relevant. Retrieved from: https://techbeacon.com/app-dev-testing/future-software-testing-how-adapt-remain-relevant
2. Mason, R (2019, May 18) Is Agile Killing QA. Retrieved from: https://www.forbes.com/sites/forbestechcouncil/2019/03/18/is-agile-killing-qa/?sh=40f846d734d6
3. Perry, T.S. (2015, Dec 11) Yahoo's Engineers Move to Coding Without a Net > What happens when you eliminate tests and QA? Fewer errors and faster development, says Yahoo's tech leaders. Retrieved from: https://spectrum.ieee.org/yahoos-engineers-move-to-coding-without-a-net
4. HelpSystems Armenia (2019, Oct 10) Is QA Dying? [Blog Post] Retried from: https://medium.com/helpsystems-armenia-writes/is-qa-dying-730b0a166c4d
5. Ghahrai, A (2020, March 11) Problems with Test Automation and Modern QA [Blog Post] Retrieved from: https://devqa.io/problems-test-automation-modern-qa/
6. World Quality Report, 2021-22, 13th Edition. Capgemini. Retrieved from: https://www.sogeti.com/globalassets/reports/wqr-21-22/world-quality-report-2021-22.pdf
7. Zhuo, J. "What is Management?" The Making of a Manager, 2019.

8. What is Coaching (Retrieved on 2022, June 9) Retrieved from: https://internationalcoachingcommunity.com/what-is-coaching/
9. Accelerate State of Devops 2021. Google Cloud. Retrieved from: https://services.google.com/fh/files/misc/state-of-devops-2021.pdf