EMPIRIX

Empirix's Top Metrics to Achieve Contact Center Assurance

For any organization in an industry that requires frequent communication with its customers, Contact Centers play an integral role in delivering a high quality customer experience. Therefore, it will be critical for these organizations to ensure their contact centers are fully operational and optimized to enhance customer experience. Typically, this is accomplished by a combination of thorough end-to-end testing prior to roll out and comprehensive monitoring during production. It is imperative that organizations look beyond just the technology or network and focus on the customer experience. This requires building an in-depth contact center assurance strategy that ensures quality from idea inception through solution delivery, to ongoing operations, to ensure that the Contact Center, its technology and its people are achieving the primary business objectives established by the parent organization.

Out of every million products manufactured, three of them may deviate from the standard and the process will still fall within the acceptable deviation.

In the past, large organizations—particularly in industrial sectors—were well served by Six Sigma techniques when it came to quality assurance. These techniques, however, are often misused in today's Agile software development model when attempting to create a meaningful feedback loop to quality assurance. Six Sigma tools have been useful in manufacturing sectors and are well suited to that environment, but when it comes to applying them to large enterprises and Contact Centers, implementing Six Sigma tools can be like jamming a square peg into a round hole.

So how should your organization best track the progress it is making and assure a quality experience? What are the metrics that should be compiled and followed to ensure a successful contact center assurance strategy?

Before delving into these questions, it's important to understand what Six Sigma and Agile are, and how they do—or don't—work together.

SIX SIGMA AND ITS METRICS

Six Sigma is a data-driven approach meant for eliminating defects by reducing variation to an acceptable level within a process. Sigma (the Greek letter ‘o’) represents the standard variation of a population of data in statistics. In manufacturing, this variation is usually when the production process yields different results sometimes. This may result in faulty products or products that otherwise do not meet customer expectations. For this reason, variation can be synonymous with deviation in a manufacturing context, as variation from a customer’s expected product or outcome is undesired.

Three deviations per million would mean that, out of every million products manufactured, three of them may deviate from the standard and the process will still fall within the acceptable deviation.

While deviation exists in all processes, different manufacturers will have each defined their own acceptable levels of deviation. For example, three deviations per million would mean that out of every million products manufactured, three of them may deviate from the standard and the process will still fall within the acceptable level of deviation. Given the product’s costs, volume of production, level of customer expectation, etc., the number of acceptable deviations may change. Six Sigma is a tool for helping manufacturers identify and mitigate deviation levels in their output, leading to fewer dissatisfied customers and decreasing the costs involved with compensating for deviation.

The Six Sigma process for measuring how many deviations are reported after a system has gone live can be used in two different ways. One is as a measure of the performance of the system’s developers and quality assurance test teams, where fewer reported defects indicate that the developers and test teams were “good” while more reported defects indicate that they did not meet expectations.

The second way would be to view reported defects as indicating flaws in the gathering of requirements. This interpretation of the data does not assume that reported defects mean the developers and testers didn’t perform well. Rather, it takes into account that the end users intent may have been miscommunicated. If this is the case, then the developers and testers have been given imperfect criteria. In short, they may have done their jobs perfectly, but were developing solutions based on inaccurate or miscommunicated requirements. This is why Six Sigma can be misused in the context of contact centers and Agile software development. It could be used as a tool to improve the process of requirement gathering; however, it is often used as a tool to gauge development and quality assurance teams.

Refactoring code—restructuring existing code without changing its external behavior is a necessary evil. However, Six Sigma processes may deem refactoring code unnecessary since it does not produce a “value add” for the end user as the external behavior of the code has not changed the user experience. However, refactoring code is a long-term investment that can pay off in the long run, increasing readability on the back end and enhancing developers’ abilities to ramp up and join projects, and to inherit code from other developers.

It allows for faster reworking and identification of defects within the code; this reduces the future inherent waste in transitioning code to other developers, streamlining their understanding of the existing code. In real-world terms, it enables the organization to segregate business applications from platform investment and increases the agility in the event that the organization must switch vendors or platforms rapidly based on market conditions or business needs.

If the process of refactoring your code seems prohibitive due to early choices made regarding system architecture, this could point to the quality of the development team. The fact that it did not build the system with potential refactoring in mind—or is otherwise skipping the important refactoring step, perhaps because some Six Sigma processes deemed it wasteful—may raise some questions about applying Six Sigma to software development to begin with, particularly if your system under test is intended for an industry other than manufacturing.

Developing a CTQ tree first involves the identification of critical customer needs.

Critical to Quality

The critical to quality metric is based on the idea that certain sub-processes within a system are critical to the perceived or actual quality of an outcome. Basically, CTQ determines what a customer needs in order to be satisfied with any product, including contact centers. It is visualized in the form of CTQ "trees." These diagram-based tools translate the broad needs of customers into specific and actionable performance requirements that must be met in order to achieve customer satisfaction.

Developing a CTQ tree first involves the identification of critical customer needs. For example, let's say one of the critical customer needs for your contact center is to decrease the time customers spend in a call before their issue is resolved. Your contact center must deliver on this in order to make your customers happy.

Next, you must identify quality drivers in order to achieve this result. Meeting or failing to meet these needs will determine your customers' perception of the quality of your product. Continuing with the aforementioned example, your contact center's ability to route customers' calls to the agents best suited to help them may mitigate the time to resolution for the customer. This may also enhance customer experience, limiting interactions with agents who are not well suited to assist the customer

Lastly, you must establish quantifiable performance requirements against which to measure the efficacy of the sub-process in question. This is a critical step as it enables your quality assurance team to measure whether or not the sub-process has functioned properly and contributed to overall customer satisfaction. This could mean running a test group of users through the contact center's self-help system with a variety of queries, and evaluating its ability to route these calls to the agents who can best handle them.

Critical to Customer

The critical-to-customer metric is, more simply, used to determine what the customer wants and measures whether or not your contact center is enhancing the quality of the experience that a customer has when interacting with your organization. An example might be the number of questions a customer must answer in the contact center system before their issue is addressed. The CTC metric may stipulate that the customer need not answer more than three questions before being provided with a potential solution to their query. The IVR menus must then be redesigned to meet this metric and the agents must leverage the information at hand in order not to have customers repeat themselves.

AGILE METHODOLOGY AND ITS METRICS

Agile is a software development methodology that is more in line with how contact centers and enterprises deploy services today. This means it corresponds to contact centers and to software- and cloud-based services, and is less geared toward hardware and paper-based manufacturing plants (though some might argue that producing happy customers in any industry is similar in principle to manufacturing a physical product). The Agile methodology generally used in software development provides an alternative to traditional project management and arrives at solutions through collaborative, cross-functional teams. Agile methodology utilizes a range of metrics, including burn down rate, velocity, defect classification and count, and defects deferred to a future release.

Agile methodology utilizes a range of metrics, including burn down rate, velocity, defect classification and count, and defects deferred to a future release.

Burn Down Rate

The burn down rate is the measure of a team's progress. It compares to the amount of work completed to the remaining work to be completed over a specified timeline, or a "sprint." The determined burn rate is useful for calculating the required rate of work necessary to meet project deadlines by a given team. Burn down charts help to visualize the amount of work completed versus work yet to be completed.

Velocity

Velocity is the work-rate average of a "scrum team," a holistic development team within the Agile methodology. This metric helps predict how long it may take for the scrum team to complete a work backlog by tracking predicted workloads and work completed over multiple iterations, with each additional iteration rendering a more accurate velocity prediction. Monitoring velocity is an ongoing process: as velocity is measured over time, the metric indicates optimization of team work processes—or a lack thereof. This helps teams understand which new processes and team changes are resulting in improved work rate, and which are not. If monitored velocity decreases, this may be a sign that the development process has become inefficient and requires optimization.

Defects Classification and Count

When faced with defects, Agile teams can work with a triage approach which allows them to manage the flow of defects and direct them toward their corresponding resolution processes. Defects are thus divvied into one of three categories: defects found during the development phase of the contact center system, defects found after a system rollout or release by operations teams, and defects found by customers.

Defects Deferred to a Future Release

Not all defects can be addressed in the near term. Deferred defects cannot be fixed or prioritized within a particular cycle and are therefore deferred for the foreseeable future. They are thus shelved in the interim, with the intention of addressing them in future releases.

TOP METRICS FOR ALIGNING AGILE AND SIX SIGMA FOR A CONTACT CENTER ASSURANCE STRATEGY.

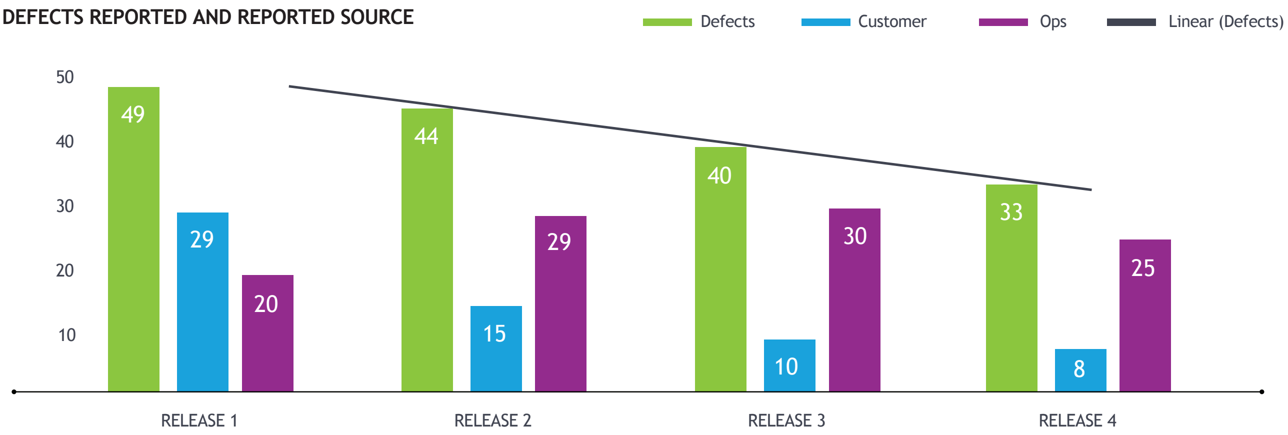
Your enterprise may be shifting to a more Agile development methodology but may have Six Sigma programs in place to monitor software quality and defects. Empirix recommends focusing on these 10 metrics as part of a holistic contact center assurance program that keeps major business objectives and the customer experience in mind.

10. Defects Reported During the First Eight Weeks after a Major Application Rollout

Some defects will invariably make it through the testing and rollout process, and will only be discovered after rollout. One effective metric for measuring these defects is to analyze the number of defects reported by operations teams and customers in the first eight weeks following a major application rollout.

This number can tell you how good your customers and operations teams are at finding the defects in your system. This metric can also teach you how defects are found and how they are experienced by system users. This metric can be achieved through your operations ticket system and through call center agent feedback reports.

DEFECTS REPORTED AND REPORTED SOURCE



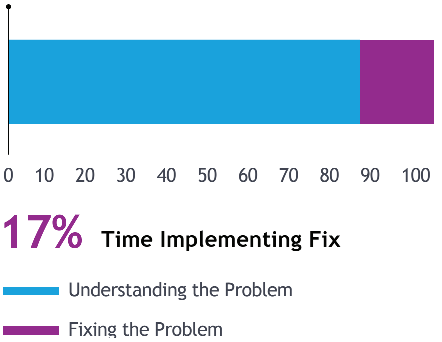
9. Mean Time to Understand

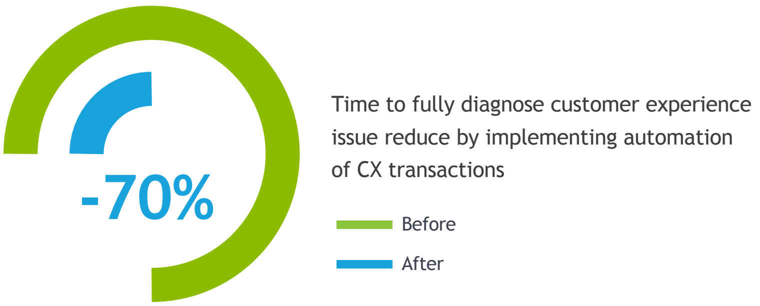
Your quality assurance team needs to be able to find and fix system problems that may negatively affect user experience. This is often referred to as the “mean time to repair.” Unfortunately, this metric can lead to distraction as a number of quality assurance players attribute errors to different causes or other teams, undermining the general effort to repair. Teams should approach this with a different metric: **the mean time to understand** the problem. Fixing an issue is usually not that hard once the problem is understood, so understanding the problem is the key to resolution.

The mean time to understand metric is a good indicator as to whether or not you are monitoring systems in the right way. Following this metric can help to determine whether you are meeting some of the following standards when monitoring your systems:

Do you have multidimensional awareness of your system?

- Passive network indicators: useful for network monitoring and forensics;
- Active user experience indicators: visibility beyond your system firewalls;
- Sub-component dependencies: linking to business transactional value.





Time to fully diagnose customer experience issue reduce by implementing automation of CX transactions

8. Cost Impact to Revenue Based on These Defects

We know that customers are more profitable over time[1], and reducing customer churn by as little as 5% may have a profitability impact of 25-125%. The negative impact that a service-quality issue may have on the bottom line is a critical metric which must be monitored and reported on. Focusing on the contact center, most contacts occur within the first 90 days of customer attainment. Their experience within this time window will be heavily weighted and solidify the lifetime value of that customer, increasing the likelihood of retention, repeat business, and referrals. This is the key to driving profitability and growth based on customer experience improvements.

The cost efficiency to service an increasing number of customers in a self-contained, self-service system is too powerful to ignore, and “containment rate” often becomes a metric measuring its success. What has been the movement of customer satisfaction for those who engaged the platform versus those who bypassed it? Could they have engaged the platform if it were designed or operated better? How is that aligned with recent changes to the platform?

Outage scenarios are easy to measure; if customers were unable to purchase or pay the business during this time, how many customers were impacted? How long was the outage? What types of customers were impacted (high value or low margin)? These types of analytics are vital for making proactive outbound decisions around attempting to regain or resolve these problems.

Self-service platforms that are confusing and poorly designed may frustrate, and thereby impact the overall customer experience to the point where the customer will either learn to never engage the platform or simply move to a competing service. Therefore, knowledge of customers who have abandoned and exited the self-service platform is a critical piece of data most organizations have, yet they are unsure of the “intent.” Were calls routed out of the platform by default, based on a technology failure? Was it a user’s failure to engage the platform? Was the menu design too confusing, or were users unable to gain value from the platform due to their perception that their inquiries could not be handled? Is there a regional or demographic bias against the self-service platform? What kind of standard usability and stability tests were done based on culture/demographics?

Does your platform offer any value-add to users that self-serve rather than those who choose another channel of contact?

For example, if an IVR menu redesign reduces time spent in the IVR by 5% but the containment rate drops by 3%, is that a positive outcome? If agent talk for non-revenue generating service calls increases by 10% after a menu rewrite, has the organization benefited?

EXAMPLE AFTER ACTION COST REPORT

INCIDENT #2016987		
SOS API Limit — Concurrent Connections to Backend Limit Reached	**Data**	**Cost**
Length of event [A]	00:00:45:00	$(355,500.00)
Customer satisfaction index degradation [B]	-3%	($72,000)
Total number of customers impacted during event [C]	2000	$(6,000.00)
Incident summary cost		**$ (433,500.00)**

A. Based on $7900 Per minute of downtime: Dr. Ponemon - Ph.D industry survey - Ponemon Institute
B. Based on 1 Year Lifetime value of $1200 Per Customer impacted
C. Based on 5000 calls per hour, self-service containment outage to live agents @ $3 per call

7. Interactive Voice Response (IVR)/Self-Service Application Regression Test Coverage as a Percent

This metric provides you with an idea of your release velocity capabilities and determines whether your test automation tool is indeed built for test automation. This is determined through specific questions and measurements:

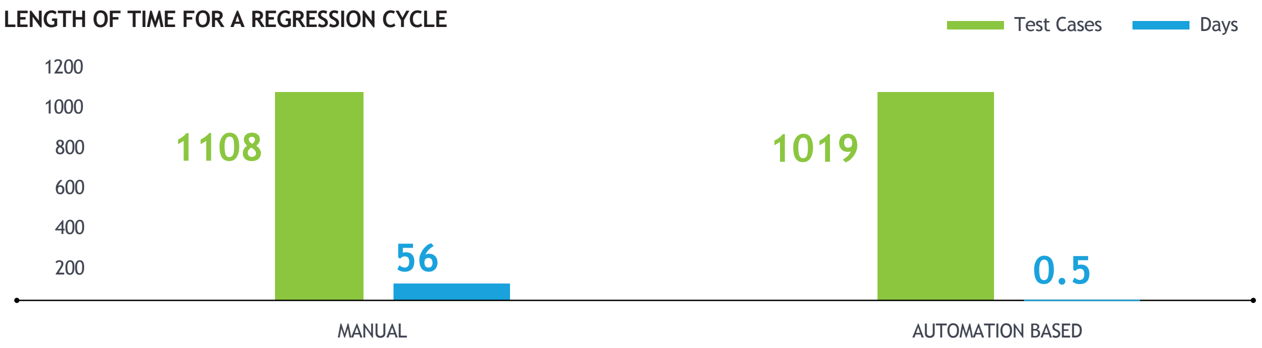
- Does the tool build the use cases for you based on the application design?
- Does it integrate with your back end for runtime capabilities to increase the lifespan of your automation scripts?
- Is it tied to the application's design code?
- Is it aware of its own environment to test grey paths—the valid paths that abruptly take invalid, or unexpected conformation, paths at some point?

Use of this metric results in greater test coverage, which in turn results in fewer defects and faster release schedules—especially considering the time spent performing regression tests after each Agile sprint.

6. Length of Time for a Full Regression Cycle

The length of time for a full regression cycle is the total time, in days, it will take to execute all of the test cases in the regression library, including report review. Regression testing is necessary to ensure that previously developed and tested software has not regressed, that errors and defects that may have emerged as the software has changed or interfaced with other software are found and repaired.

The length of time to complete a full test for software regression is one of the leading indicators of whether or not you are using the right tools and methodology for test automation. For example, one organization used manual test case and execution, with a regression cycle time of eight weeks and they believed they were touching only 5% of their entire application while another business implemented a model-based automation framework that performed automated test case creation, scripting and execution—their regression cycle time was only six hours, and covered over 1,019 test cases. Reducing this metric to streamline your regression cycle can be a huge time and resource saver.



5. Length of Time to Reproduce Defects Found in Operations

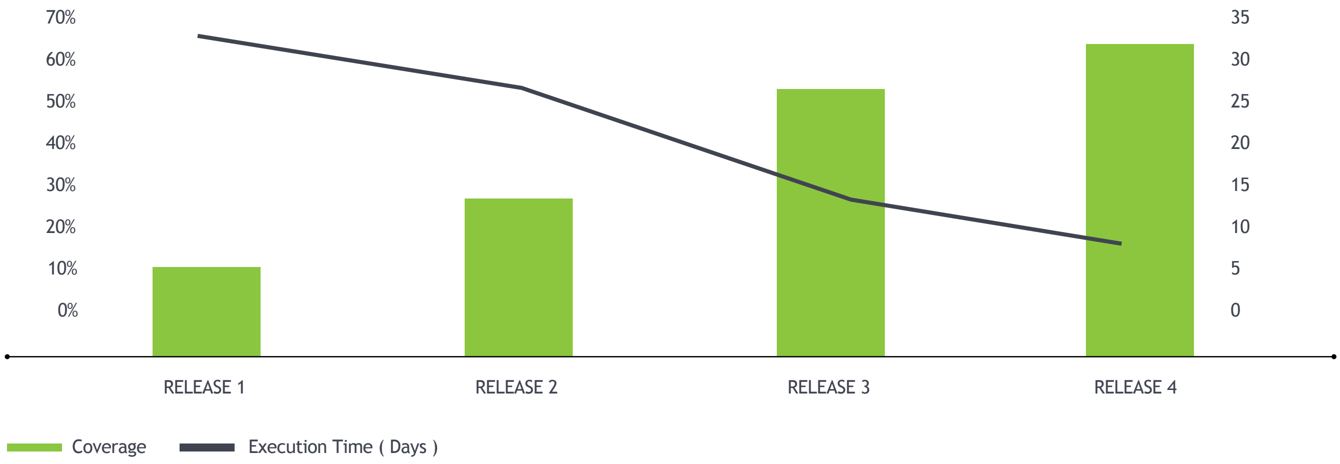
Defects are not always easy to reproduce. reproduce. Sometimes they are only revealed under certain conditions. Sometimes, your vendor may not acknowledge one exists until it's reproducible. However, it is critical that these defects are reproduced so that development teams can produce and prove a solution before rolling it out. This metric is leveraged in order to evaluate your test- staging environment for quality assurance. If you aren't able to quickly automate a use case within such an environment or to apply it under the proper load in a meaningful way, the related defect is likely to re-occur after a fix has been rolled out because the root cause was never fully realized and reproduced. The developers may have made a "best effort," but this leaves room for defects to persist.

Investing in this metric is important, and generating business buy-in is necessary to achieve it. To make the case, you will need to map the length of time to reproduce an error against the mean time to eliminate the defect—in other words, the timeline from when an error is reported to when the engineering team has reproduced the conditions that caused it and developed a fix.

4. Average Percent That Application Test Coverage Increased vs. Percent That Time Regression Cycle is Shortened Through Automation

It's important to ensure that application test coverage is increasing and that regression cycle time is decreasing. If the trend is moving in the other direction, it's an indication that a process is incorrect—be it test case design, tool capabilities and the understanding of said tool, or the business side's ability to communicate its requirements or support your team with integration processes to make automation successful.

TEST COVERAGE VS REGRESSION CYCLE TIME



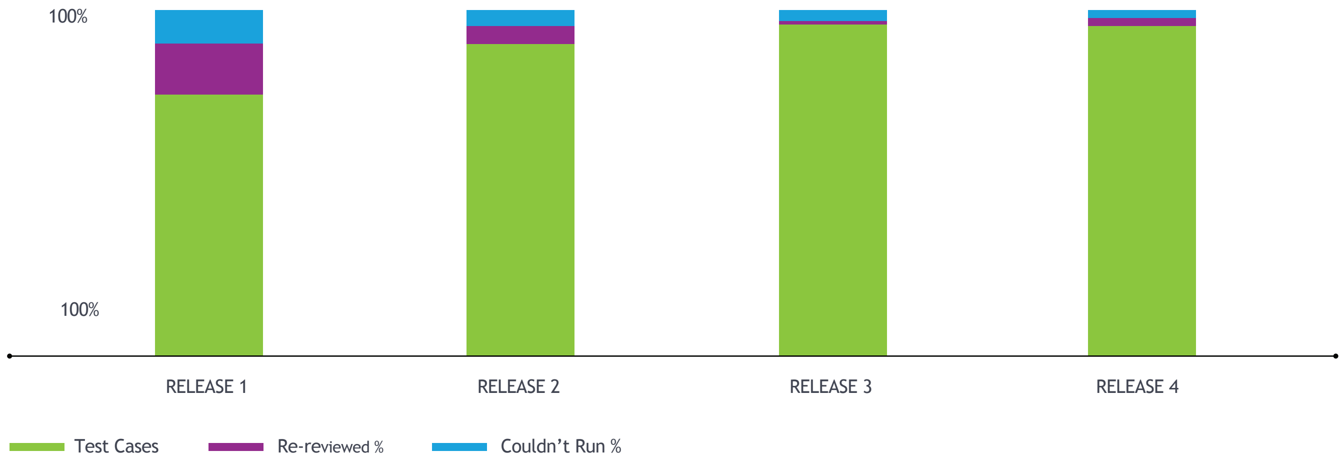
This metric is a leading indicator of your knowledge of test automation methodology. Your testing vendor should be able to communicate to you best practices beyond their tools and software, providing general knowledge regarding the test hooks that need to be written in a test automation framework in order to increase coverage and decrease execution time.

3. Percent of Test Cycle Impacted by Poor Data/Lack of Integration or Lack of a TRUE Automation Framework

This metric measures the number of test cases and overall instances that could not be run, had to be run again, or had to be re-reviewed a number of times as a result of bad test design, a lack of integration into the back-end data systems, or organizational politics that have prevented a proper automated method to achieve true automation.

It provides a business case for adopting a better methodology and better interoperability, and for investing in a testing framework that is truly automated. It also reveals the impact of organizational politics and red tape on testing methodology and generates the case for adopting true automation frameworks and the real-time intelligence required in order to provide real-time business value.

TEST CASE IMPACTED BY POOR DESIGN OR LACK OF INTEGRATION



2. Percent of Structural Exits Out of the IVR vs. Non-Structural Exits for the Past Six Releases

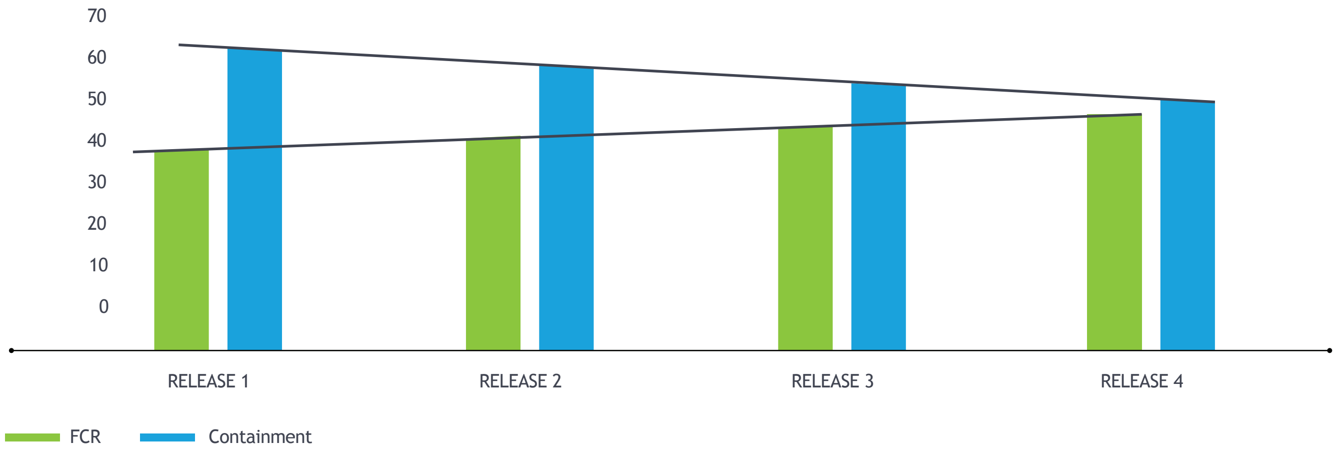
This is a percentage of calls that leave an IVR system at the "right time." For example, the right time might be a prompt from the IVR indicating that the customer is being transferred ("Please wait while I get someone to help you"). The "wrong time" would be when a frustrated customer hits 0 for the operator or when a technology fault forces the customer out of the IVR, typically accompanied by a message similar to, "Sorry, I couldn't find that. Please hold while I transfer you."

This metric can indicate whether or not you are creating value for the company by enabling self-service: if you are structuring your application in a way that people can engage with it, if they are willing to engage with it, if you are functioning well from a technical standpoint, and if the right data is available at the right time. The value behind this metric is critical to your contact center assurance program's overall direction.

1. First Contact Resolution Over Time vs. Self-Service Containment Ratio

First contact resolution over time is the percentage of calls in which the customer believes that their issue or issues were resolved during their first interaction with the organization; it is important that interactive voice response is included in this metric. FCR is measured by conducting customer satisfaction surveys and after-call analyses, as well as via milestone acceptance and repeat contact calculations with agent call codes. The self-service containment ratio metric measures the percentage of customer calls that begin and end in the IVR system without moving the call to a real person. It indicates the effectiveness of the IVR in resolving the customer's queries and issues—typically frequently asked questions—but it could include complex transactions like change of address or re-booking a flight—without having to pass the caller to an agent. If contact center assurance is your goal, then these two metrics should never be separated.

FCR VS CONTAINMENT



There is some disagreement around how FCR is defined and what it indicates. Many argue that it is really a measure of agent empowerment and ability to solve customer issues, but it is also a measure of customers' willingness to engage with the platform and of the platform's capabilities regarding segmenting, classifying and leveraging skills-based call routing (assigning incoming calls to the most suitable agents, rather than choosing whoever is available to take the call).

Beyond being a good way to measure customer satisfaction, this is an important part of a solid contact center assurance strategy. The relationship between the FCR and self-service containment ratio metrics will change over time, indicating how well your self-service applications are designed, whether or not these applications offer a worthwhile user experience, if they provide customer segmentation correctly and if they route calls to agents who are able to serve customers.

CONCLUSION

In the end, the concepts of Six Sigma, a continuous quality improvement program, and Agile are not mutually exclusive, however they do speak very different languages at times. Agile's iterative approach is a perfect opportunity to apply Six Sigma principles to reduce defects and improve quality over all the numerous iterations that Agile offers. Often the velocity of agile methods are too fast for the data requirements and analysis of Six Sigma programs, however the metrics mentioned above give the best guidance in a contact center setting to gauge 'right direction/wrong direction' for executive management.

Agile's iterative approach is a perfect opportunity to apply Six Sigma principles to reduce defects and improve quality over all the numerous iterations that Agile offer.

To get started towards a more agile Six Sigma contact center assurance strategy, you would require: a uniform method of collecting the following is required:

- Development data
- QA data
- Operational data regarding customer experience
- Customer engagement with the platforms
- The system performance those applications exist on
- The visibility into the network performance those systems communicate through and
- An analytical engine that could offer you feedback in real-time

This would not only help direct your operations teams, but your marketing and application development teams as well. Contact Center Assurance programs are in fact Customer Experience Assurance programs.

[1] Institute of Chartered Accountants in England & Wales – Customer Profitability Analysis
[2] Ponemon Institute. (2016). 2016 Cost of Data Center Outages. Columbus OH: Emerson Power.