AMBER

2026

THE AMBER QA HANDBOOK

From Metrics to Decisions

INTRODUCTION

Quality Assurance has always been about finding bugs, but in modern game development, that's only part of the story. Today, QA teams are expected to make informed decisions, prioritize risks, and contribute meaningfully to product strategy. And that's where **metrics** come in.

This handbook is not meant to reinvent QA, because you won't find radical new theories here. What you *will* find is a **clear, structured way** to think about **QA metrics** and **KPIs** and, more importantly, how to use them in day-to-day decision-making.

One of the most common challenges we see across teams is not a lack of data, but **a lack of confidence in interpreting and applying that data.** Junior QA team members often collect metrics diligently, yet struggle to understand *what those numbers actually mean*, when to act on them, or how to communicate their insights effectively.

{ THIS HANDBOOK EXISTS TO BRIDGE THAT GAP. }

Inside, you'll find practical explanations of commonly used QA metrics, guidance on how to interpret them, and real-world context for when (and when not) to rely on them. **The goal is to help QA teams move from reporting numbers to making decisions,** supporting both their confidence and the broader development process.

At Amber, we work closely with QA teams across a wide range of projects and production setups. **This material reflects the questions, challenges, and learning moments we encounter most** frequently **when supporting teams in the field.** It's designed to be used as a reference, a training tool, and a conversation starter, especially for teams looking to level up how they use metrics without overcomplicating the process.

1 USEFUL DEFINITIONS

MEASUREMENT = the quantification of attributes of an object or event, used to compare with other objects or events. In other words, measurement is a process of determining how large or small a physical quantity is compared to a basic reference quantity of the same kind or how fast or slow a process advances towards a target.

A PHYSICAL QUANTITY = a measurable property of a material or system. It is expressed as a numerical value multiplied by a unit of measurement. For example, mass m can be represented as m = n kg, where n is the numerical value and kg is the unit.

METRIC = any property, characteristic, or tendency of a system that can be quantified by measurement and expressed by a value (number, percentage, r ratio, etc.). Metrics are quantifiable measurements used to assess performance, track progress, and measure the success of various processes, initiatives, or approaches.

All metrics depend on data that can be reliably collected, tracked, and analyzed over time.

Various metrics can be measured and recorded as:

- Numbers
- Percentages / Ratios
- Counts (headcounts, bug counts, etc.)
- Ratings (confidence, partner satisfaction, Net Promoter Score, etc.)

KPI = Critical / Key Performance Indicators. All KPIs are metrics, but not all metrics are KPIs. KPIs are tied to **specific goals** and have an ideal target value.

Various metrics can be measured and recorded as:

- **Name** (e.g. Average number of regressed bugs per hour)
- **Definition / formula** (e.g. Number of total bugs regressed/Total hours allocated for bug regression averaged daily, weekly, per sprint etc)
- **Unit of measurement** (Bugs/hour, $/h, % completion, open bugs / scrum)

Consequently, any KPI **must** include the same info as any metric: name, definition/formula, unit of measurement, plus a target value, which can be expressed as:

- A single value (e.g.: the QA budget cannot exceed **$10.000**)
- An upper limit, also known as a ceiling (e.g.: if there are >**15 x** 100% repro blocking bugs on the user path, all Programmers will switch focus to bug fixing)
- A lower limit (e.g.: to release this build, the **frame rate** should not drop below **30 fps**, regardless of in-game user actions)
- A range of values
- A percentage of a specific quantity (e.g.: this build cannot be considered final if more than **10%** of all open bugs are of mid and critical severity)
- Accomplished milestones and deliverables (e.g.: we cannot proceed with a soft launch if **alpha** and **beta** haven't been declared for the prototype of our new game)

Following this definition, KPIs are not an essentially different category of metrics.
Metrics are promoted to KPIs only in relation to a specific goal.

Furthermore, KPIs usually fall into one of two categories:

Leading KPIs: Predict future performance outcomes

These enable proactive changes, since they provide an early indicator about how likely you are to meet your goals. If a certain type of testing is leading to very few crashes and critical issues despite substantial QA time investment, this should warrant a revision of the current test strategy by the QA Team Lead to determine whether the product quality has already reached Release C[CR1.1]andidate level. Or simply the test method used starts returning diminishing results. Critical bugs might be found through other test methods.

Lagging KPIs: Measure past outcomes

Leading indicators help guide changes in the day-to-day actions to maximize chances of reaching a goal, whereas lagging indicators help measure past success and provide insights about what worked well and what did not. Lagging KPIs are reactive by nature since they measure what is already finished, and they do not predict future performance.

CASE STUDY

Let's consider a plane trip to your favorite destination.
The *cost of plane tickets* is a metric (measure of cost), **as** is the *time to destination* (measure of speed), and the *perceived quality of the seats & onboard service* (measure of comfort).
In the absence of any defined goal, none of these metrics is a KPI.
Depending on the goal, any of them can become one:

- If your goal is to stay within a *given budget*, the ***cost of plane tickets ($)*** is the decisive selection factor, hence it is a KPI.
-
- If your goal is getting to the destination in a *given timeframe*, the ***time to destination (hours / business days, calendar days, etc.)*** will be the decisive factor, hence a KPI.
-
- If you just had a multiple fracture still in the process of healing and having the smoothest ride is essential for you, the ergonomics of the plane seats & the onboard service (comfort rating) will be the driving factor in your decision. Thus, a KPI.
-
- If you had a multiple fracture and **needed** to travel as fast as possible to your destination on a limited budget, **all 3 of the above** become critical factors, hence KPIs.
-
- If you need to travel there to close a business deal, don't have any healing fractures, and the plane tickets are purchased by the **organization** employing you, ***none*** of the above metrics will tell anything about how likely it is that you'll close the deal successfully or not. Thus, none will be a KPI.

1.1 METRICS IN MANAGEMENT AND BUSINESS

The scientific, economic, societal, and cultural changes brought by the industrial revolution prepared the ground for the definition and deployment of metrics in a management and business context. These were formalized for the first time by Frederick W. Taylor in his seminal work The Principles of Scientific Management, published in 1911, establishing the scientific theory of management.

"The principal object of management should be to secure the maximum prosperity for the employer, coupled with the maximum prosperity for each employee," as defined by Taylor as the primary goal.

Taylor's theory focused on four core principles:

- Replacing the traditional, rule-of-thumb methods used to plan and organize work with an analytical approach to task definition, breakdown, allocation, and sequencing, aiming at finding the most efficient, easily measurable, and repeatable way to accomplish a certain type of work. This makes possible standardization and automation.
- Selection and continuous training of workers based on the assessment of skills and the requisite skill levels needed to complete each activity in the production lifecycle.
- Increased cooperation between workers, as well as between workers and managers, by providing a common, objective, and measurable benchmark for performance and efficiency.
- Division of labor, defining different areas of focus between managers and different types of specialists.

In today's competitive landscape, different metrics and KPIs are effectively steering team, business, and organizational-level decisions and behaviors, so developing basic proficiency in interpreting and managing such quantities is of utmost importance for any professional in any field.

As we engage with larger organizations in the Game Development industry, it is not uncommon for the terms and conditions for collaboration and contracts with internal or external QA teams to be entirely defined around such performance metrics. This allows both the evaluation of QA deliverables against set targets and the comparison of different QA teams working on the same project in terms of outcome, efficiency, and reliability.

Bad or poorly thought-out KPIs will usually have a negative impact on the team or organization. Unrealistic KPIs, badly defined or impossible to reach, will typically yield the opposite effect from what they are intended to achieve. For example, a QA Team Lead might hesitate to proceed with the removal of a poor performer if they believe that terminations due to poor performance will count towards total team attrition. Or they might give a tester an unjustified low rating in the annual performance evaluation if they classify as invalid bugs also the NMIs (Need More Informations) and duplicates that were not caused by poor bug writing and dupe searching for the tester.

1.2 KPIS VS SERVICE LEVEL AGREEMENTS

Many established publishers formalize a Service Level Agreement (SLA) before starting the collaboration with different vendors and service providers.

The primary purpose of a Service Level Agreement is to establish a common, measurable basis for objectively measuring service quality and identifying areas for improvement. These agreements establish clear expectations, ensuring both parties understand what is agreed upon and what happens if those promises aren't met.

In the diagram below, we provide a common example of SLAs defined in the master service agreement between a publisher and a QA vendor.

Quality SLAs

Found vs Fixed Rate - Severity, by Active Project. 80% or better goal.

Percentage of issues found vs actually fixed. Broken up by Severity, by Project worked the previous quarter.

Escaped defects (Test Effectiveness) 95% or better goal.

Issues caught post-release / total bugs found during release phase, per project.

Operational SLAs

Updated QA Tasks by end of business, including hours worked.

All in-progress tasks to be updated either in comment or closed before end of business each day.

Regression upon each new applicable build.

Upon the next indicated available build vendor will regress all applicable bugs, with priority on any Sev1 and Blocker bugs.

Respond to JIRA bugs in QA state requesting information within one business day.

Any bugs requesting QA review or information to be done within 24 hours of request. JIRA statuses can be Open, QA Review, Ready for QA, or In QA.

Whereas KPIs track internal performance against goals (which can be entirely defined regarding SLAs), SLAs define the expectations for service performance, and usually they are legally binding. SLAs can be defined as a set of service performance metrics, but non-compliance may result in penalties, including the immediate termination of the agreement. In contrast, failing to meet a KPI target would normally not have any direct or contractual consequences.

COMMON VARIABLES & QUANTITIES

Game testing projects can be very different in terms of scope, length, and complexity. But despite the low level of standardization in the industry, they will generally follow the same phases and revolve around the common variables presented in Figure 1 below.

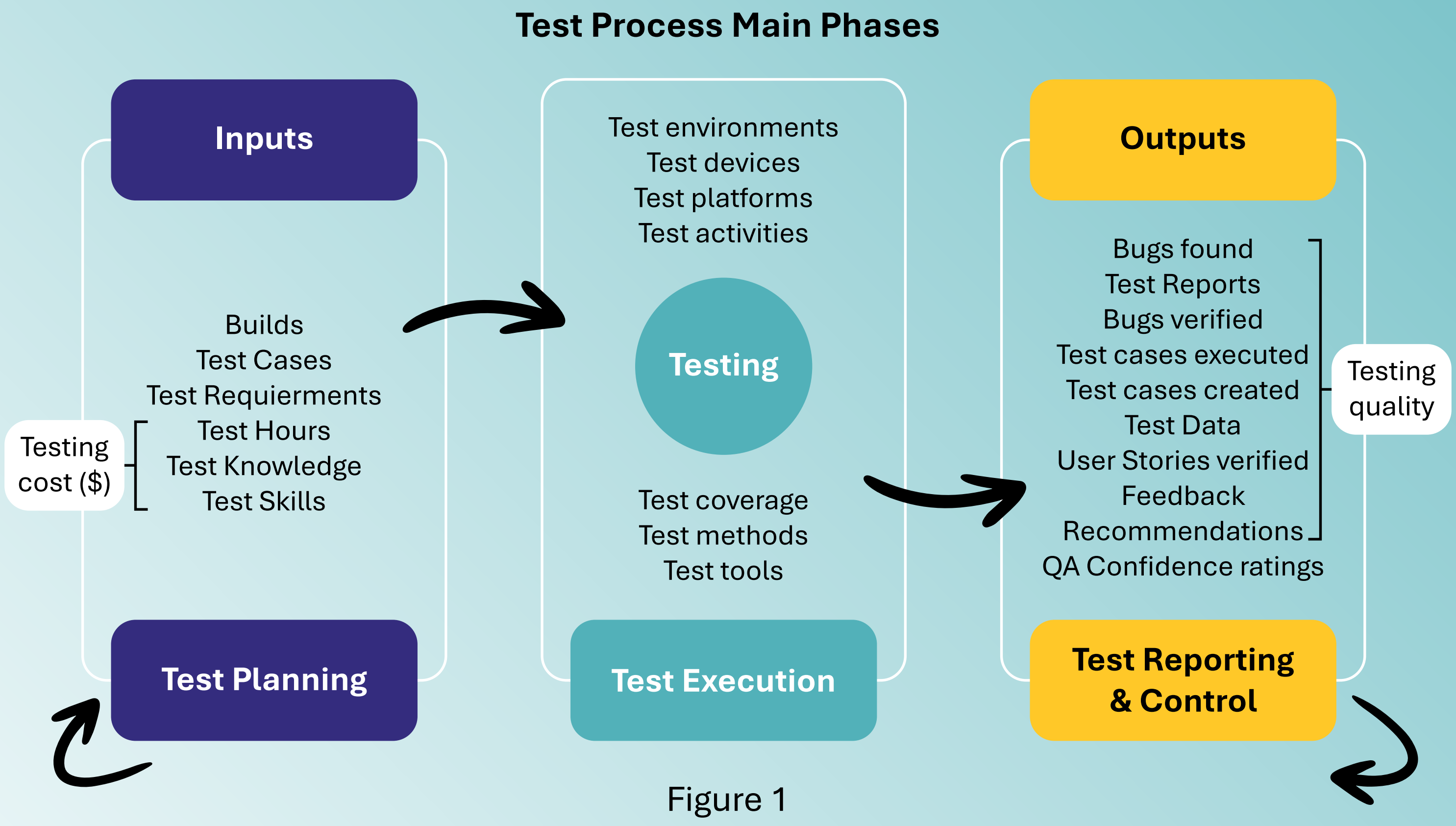

Figure 1

Metrics can be set in relation to one or more variables, depending on the goals of every measurement as established by various project stakeholders.

For example, a metric can be set in relation to the test requirements of a new feature and the number of associated test cases created. The purpose is to evaluate the current requirements coverage, determining the percentage of requirements that still do not have any associated test cases (TCs). Such a metric would then allow the QA Team Lead or QA Specialist to estimate the remaining effort needed to write down test cases until reaching 100% requirements coverage.

Another example would be a metric set in relation to test hours, and the number of bugs regressed to determine how many testers would be needed to regress 100 bugs during one overtime day. If, on average, a tester would regress 3 bugs per hour, then for 8 hours, the length of a usual business day, one tester would be expected to regress on average 24 bugs. Dividing 100 bugs by 24 bugs per tester results in 4.16 person-days, meaning a QA Team Lead could confidently commit to regressing all 100 bugs during one overtime day with five testers.

An unlimited number of metrics can be generated this way. Their validity depends strictly upon the extent to which they obey the mathematical rules applying to the types of quantities involved, as well as the physical constraints of the system where they reside. Relevance depends entirely on the individual goals driving this effort.

2.1 TYPES OF QUANTITIES

This list is not exhaustive and introduces only the most common measurable quantities involved in game testing processes.

COUNTS = The total number of individual things in a given sample obtained by counting, usually expressed as positive integers.

Examples: Bugs (in general or of different severities, types, statuses, reproduction rates, or numbers of bugs verified - total or by regression status), Test Cases (created, executed, passed, failed, reviewed, blocked updated, etc.), Builds, Testers, Failing Smoke Tests, etc.

PERCENTAGES = An amount or share of something, in relation to a whole that is often expressed as a number out of 100 (%).

Examples: Share of blocking bugs out of total open bugs, Fix Rate (the percentage of bugs fixed out of the total amount of bugs claimed fixed), share of failing Test Cases out of the total number of Test Cases in a Test Pass, etc.

AVERAGES = A number expressing the central or typical value in a set of data, most commonly the mean, which is calculated by dividing the sum of the values in the set by their number. Many Gaming QA metrics are mean values related to the test team's combined efforts and output.

TIME INVESTMENT = Total time allocation spent on test activities, particular test types, test areas, etc. Usually expressed in hours or person-days.

1 person-day = 8 hours; this particular measurement unit simplifies the conversion of any test effort estimate expressed as a number of hours in the corresponding number of Testers needed for completing the work (e.g. 40 hours = 5 person-days = 5 Testers working one business day each, or 1 Tester working 5 business days).

RATES OF CHANGE = How quickly something changes over time. They can be thought of as a kind of speed, since speed is the rate at which an object's position in physical space changes as time goes by. By knowing the speed and the time elapsed, one can find the distance covered within this amount of time. Rates of change are a valuable progress indicator and a great aid in forecasting time or resources needed to complete a given amount of work in a given amount of time. It's typically calculated by dividing a changing quantity (x) by time, often expressed in hours; hence, its measurement unit will be x per hour.

Example: Test cases executed per hour provides the speed at which TCs are burned down by the test team. Other common rates of change are Bugs verified per hour, Bugs found per hour of exploratory testing, etc.

CONCENTRATIONS = The amount of something that exists in a defined space, volume, or static component. Whereas the rate of change can be thought of as a speed, concentration can be thought of as a density.

Examples: Bugs found in a specific game area or component, open critical Multiplayer, In-App Purchase, or Audio bugs, etc.

RATIOS = Similar to percentages, since a percentage is a ratio that compares a number to 100, ratios help illustrate the relationship between a slice and the whole.

Examples: A 10:1 ratio between Testers and QA Team Leads, or a 4:1 ratio between Programmers and Embedded Testers.

COSTS = An amount, usually a sum of money, that must be paid or spent to obtain something. Typically expressed as a positive integer and a currency. In most cases, time investment can be converted directly into cost by multiplication with an hourly rate, and vice versa.

2.2 SAMPLE SIZE

Sample sizes depend on the types of quantities involved in each metric as well as the type of analyses they are employed for.

Delving into any advanced topics related to Statistics and Theory of Probability will be avoided, as that would unnecessarily stretch the present introduction to metrics and KPIs beyond its scope, sacrificing utility. Therefore, empirically and generally, for any kind of sample where averages are relevant and surface a trendline in indicative of the team's average performance pace or the tendency of a system evolving. It should include a minimum of 30 data points.

Example: For a relevant estimation of the average rate at which a team is clearing out bug regressions, measuring the average RBpH at the end of each day should start settling towards a meaningful average value after 30 workdays. This is provided that the volume of daily regressed bugs remains relatively constant and the turnover within the dedicated team is not exceeding 30-40% during the same timeframe.

Similarly for determining the average number of TCs executed per hour.

It's important not to focus excessively on any absolute numbers for determining the sample sizes but instead to understand exactly what you're looking for and how many samples are necessary for statistical confidence, depending on each goal.

MAIN CATEGORIES OF METRICS

As mentioned in the introduction, metrics can provide only hints or warnings regarding the present state of a system, as well as tendencies of the likely future evolution in the absence of any strong perturbation. A process of evaluation and interpretation of results in the specific context of work where they are measured is indispensable. To avoid the risk of using the wrong metrics for the wrong areas of decision-making and to further guide the reader, below are the proposed categories of metrics and KPIs. These emerge naturally in the context of Gaming QA.

Service Delivery / Service Quality

Applied to external QA teams, regardless of working within such a team or together with an external QA partner, these usually indicate how well the delivered service conforms to the client's expectations. These measures are being carried out to evaluate the health of the vendor - client relationship (e.g. Net Promoter Score, Partner Satisfaction Ratings or various Service Level Agreements), service delivery, organizational maturity level, and adherence to agreed requirements.

QA Process (general & effectiveness related)

Measure the effectiveness or yield of existing test types and test processes (e.g. Bugs found per hour by test activity - Exploratory Testing, Scripted Testing etc., Bug detection efficiency, Defect capture rate, TC status /test run over multiple test-runs etc.).

Product Quality, Performance, and Reliability

Evaluate the degree to which the game under test or the product meets design, customer, 3rd party requirements and works efficiently (e.g. Mean time to failure, Defect densities in various application components, Framerate, etc.).

QA Team Performance

Facilitates the evaluation of how well a QA team carries out testing. These metrics may revolve around the degree to which deadlines or testing objectives are met, compliance with internal and external standards for testing, team capacity (understaffed, overstaffed), team's level of seniority, team health, turnover, etc.

Test Execution Velocity

Progress rates for recurrent QA activities (e.g. Bugs verified per hour, TCs executed per hour etc.).

Dev Team Performance

Similar to QA Team Performance. Evaluates how well a Development team carries out Development. It may revolve around the degree to which deadlines or development objectives are met (like Bug Fix Rate or Smoke Test Pass Rate), compliance with internal and external standards for software development, quality, velocity, and effectiveness of development.

APPLICATION

All metrics are indifferent measurements in the absence of specific goals. For example, the driving distance between CDMX - GDL is 532 km: one can live without this information most of the time, but it becomes critical if one's success depends on reaching either city with an almost empty fuel tank.

The recommendation is to carry out this exercise on your project as a collaborative effort between the QA Project Manager & QA Team Lead:

1. Identify all metrics currently tracked on a given project (you can start with the info tracked in the end-of-day reports).

2. Formulate hypotheses regarding underlying goals and stakes behind every metric (why is it useful?).

3. Establish which team member each goal belongs to (QA Team Lead, QA Project Manager, QA Manager, the organization employing you as a whole and its mission and strategy, the Audit or Finance department, the client's organization and its strategic interests, Development QA Team Lead, Producer, etc.).

4. Connect current metrics tracked on the project with the parties discussed in the previous point. Which one could be KPIs for only one party, for all, for some but not all, or for none? Which ones could be SLA metrics?

5. Map the existing metrics to the following categories (check multiple categories if the metric is relevant for all):

	Service Quality / Delivery	Amber Team Performance	Test Execution Velocity	QA Process Effectiveness	Product Quality / Performance	Dev Performance	SLA
Metric 1	🟨				🟨		
Metric 2		🟨					
Metric 3				🟨		🟨	
Metric 4			🟨				🟨

6. Define at least 3 project goals, supporting the decision-making process of the QA Project Manager, QA Team Lead, or the strategic direction of the department or organization. At least one supporting metric could be used as a KPI for each.

QA Team Lead

Usually, the first level of QA Management, responsible for coordinating and directly managing a team of testers of various seniorities, usually confined to one project

QA Project Manager

Usually manages a group of QA Team Leads and a portfolio of projects. Owners of the project Test Plan and main drivers of QA strategy on their projects

QA Manager

Usually manages multiple QA Project Managers and are responsible for maintaining the overall quality and operations of their teams while keeping a close connection with the Production stakeholders. More senior managers are responsible for long-term, multi-year strategies of teams, continuous improvements and development of capabilities and adaptability

SELECTING THE RIGHT METRICS

It's unlikely any single metric will provide a complete description of reality. Metrics & KPIs inform the human decision-making process but cannot replace it. *Accountability can't be transferred to any metrics; it always stays with the human agent making the decision informed by them.*

The effectiveness of any metric & KPI deployment strategy is ultimately reflected only by the empowerment and improvements in the range, accuracy, and quality of the decision-making process that it supports. A high volume of data leading to no actions or decisions should be a clear red flag and call for a swift revision of the current process and expectations.

A strategy for data tracking should be established early on, tailored to the needs of all key stakeholders and parties involved in each project / partnership. Furthermore, controlled, or, ideally, standardized data tracking is a critical prerequisite for high accuracy / relevance of data collected.

Statistical relevance of collected data is also a key factor in determining its reliability. Average progress rates, in the conditions of relatively stable QA & Dev team size and no significant changes of team members, tend to become highly stable over time (from sprint to sprint or between different iterations of the same franchise). They get highly accurate short-term effort estimation for most recurrent test activities).

Before starting to build a metrics & KPI deployment strategy for your project, it’s recommended to consult the guidelines below:

1. Have clarity on the goals this strategy is expected to support.
2. Have clarity on expected outcomes, what is possible and what is not to achieve by it, and general success criteria.
3. Evaluate the estimated ROI vs. the compound effort (cost) of all future tracking.
4. Determine areas this strategy is expected to support. Gauging the current service quality, QA team performance, effectiveness of test methods used, or product quality are extremely diverse areas of focus. Hence, most of the time, they will require different metrics, and some metrics will require constant tracking for an extended time to become statistically relevant.
5. Define relevant metrics & KPIs, tracking method, cadence, owners, and tools for each.
6. Define at least one quality control measure to ensure minimum tracking consistency & accuracy.
7. Define at least one milestone to review actual progress vs. expectations & general effectiveness of current strategy.

METRICS CATALOGUE

A list of the most common Gaming QA metrics is provided below. The list is not exhaustive, and we encourage the reader to use the information from the 2nd chapter to set up custom metrics depending on their project’s current goals and challenges.

COMMON GAMING QA METRICS

METRIC NAME	UNIT	DEFINITION
Daily Attendance Percentage	% of Testers / QA Team Leads etc. present at work	$\frac{\text{(Actual resources)}}{\text{(Allocated resources)}} \times 100$ (actual = present)
Attendance Ratio	Ratio Actual / Allocated	Total number of Actual Resources vs Total number of Allocated Resources (allocated = hired employees)
Actual vs Planned Ratio	Ratio Actual / Planned	Total number of Actual Resources vs Total number of Planned Resources (planned = total headcount needed, not necessarily all hired / allocated)
Allocated vs Planned Ratio	Ratio Allocated / Planned	Total number of Allocated Resources vs Total number of Planned Resources
Staff deficit	Headcount	Total planned headcount - Total allocated; > 0
Staff attrition (turnover)	% of staff leaving the company or project in a given period	TBA
Manager - Individual contributor ratio	Ratio Managers / Individual Contributors	Total number of Actual or Planned Managers (TL, PC, PM) / Total number of Actual or Planned Resources
Billable utilization	% of billed resources	$\frac{\text{Billed resources or hours}}{\text{Allocated billable resources or hours}} \times 100$
Buffer percentage	% of buffer (billable) resources	$\frac{\text{Buffer resources}}{\text{Allocated billable resources}} \times 100$
Overhead percentage	% of overhead (non-billable) resources	$\frac{\text{Overhead resources}}{\text{Total resources}} \times 100$
Buffer billable utilization	% of buffer not billed	$\frac{\text{Actual buffer resources not billed}}{\text{Allocated buffer resources}} \times 100$

Buffer utilization	% of projects that can use buffer resources	$\frac{\text{TBA}}{\text{TBA}} \times 100$
Number of unstaffed hours (can be defined also as %)	hours	Total number of planned hours $-$ Total number of actual hours
Test hours per bug	hours/bugs	$\frac{\text{Total test hours delivered by the QA team}}{\text{Total no of valid bugs reported by the QA team}}$
Bugs found per test method	bugcount	Bug yield for common test methods. Total bugs found by smoke tests, general scripted testing, unscripted testing, ad-hoc requests, etc.
Bugs found by test method (scripted, unscripted, ad-hoc requests)	% of bugs found per test method out of total bugs found	$\frac{\text{Total bugs found via Test Method "Y" in timeframe A}}{\text{Total no of bugs logged in timeframe A}} \times 100$
Bugs found per hour of scripted testing execution	bugs per hour	$\frac{\text{Total bugs reported during scripted testing execution}}{\text{Total hours of scripted testing execution}}$
Bugs found per hour of unscripted testing (exploratory, destructive etc)	bugs per hour	$\frac{\text{Total bugs reported during execution of unscripted testing}}{\text{Total hours of unscripted testing execution}}$
Bugs found per hour of ad-hoc request execution	bugs per hour	$\frac{\text{Total bugs reported during execution of ad-hoc requests}}{\text{Total hours of ad-hoc request execution}}$
Bugs found by severity/type	% of bugs of a certain severity/type out of total bugs found or open	$\frac{\text{Total bugs of severity / type "X" logged in timeframe A}}{\text{Total no of bugs logged in timeframe A or Total open}} \times 100$
Bugs found by reproduction rate	% of bugs of a certain repro-rate out of total bugs found or open	$\frac{\text{Total bugs of repro rate "X" logged in timeframe A}}{\text{Total no of bugs logged in timeframe A or Total open}} \times 100$
Device specific bugs	bugcount	Total number of device specific bugs
Platform specific bugs percentage	% of platform specific bugs out of total bugs reported	$\frac{\text{Total platform specific bugs logged in timeframe A}}{\text{Total no of bugs logged in timeframe A}} \times 100$
Duplicate bug submission rate	% of duplicate bugs out of total bugs reported	$\frac{\text{Total duplicate bugs logged in timeframe A}}{\text{Total no of bugs logged in timeframe A}} \times 100$
Invalid bug submission rate	% of invalid bugs (duplicate, not a bug, invalid) out of total bugs reported	$\frac{\text{Total invalid bugs logged in timeframe A}}{\text{Total no of bugs logged in timeframe A}} \times 100$
Design change rate	% of bugs closed due to design changes out of total bugs reported or open	$\frac{\text{Total bugs closed due to design changes in timeframe A}}{\text{Total no of bugs logged in timeframe A or Total open}} \times 100$

Will not fix bug rate	% of bugs closed as WNF out of total bugs reported or open	$\frac{\text{Total bugs closed as "Will not fix" in timeframe A}}{\text{Total no of bugs logged in timeframe A or Total open}} \times 100$
Need more info rate	% of bugs returned to QA as NMIs out of total bugs reported or open	$\frac{\text{Total bugs returned to QA as NMI in timeframe A}}{\text{Total no of bugs logged in timeframe A or Total open}} \times 100$
Average time spent in QA owned status	hours	Average time elapsed between a bug being flipped to a QA owned state (NMI, Ready for QA etc) until the bug is actioned by QA
Bug fix rate	% of bugs fixed out of total bugs regressed	$\frac{\text{Total bugs marked by QA as Fixed in timeframe A}}{\text{Total no of bugs regressed in timeframe A}} \times 100$
Regressed bugs per hour (RBpH)	regressed bugs per hour	$\frac{\text{Total bugs regressed in timeframe A}}{\text{Total hours invested in bug regression in timeframe A}} \times 100$
Regression blocked	bugcount	Total bugs ready for QA for which reg is blocked
Total open bugs	bugcount	Total number of bugs existing at a given moment in the database in any open state
Bugs by game component / area	bugcount	Total number of bugs existing in a specific component or area of the game (UX, Audio, Game Levels, Server Connectivity etc)
Workload completion rate	% of QA tasks/activities completed out of total tasks assigned to QA	$\frac{\text{Total QA tasks completed by QA in timeframe A}}{\text{Total no of tasks asgnd or available for QA in timeframe A}} \times 100$
Test Pass completion status	% of TC executed by status	$\frac{\text{Total executed TCs closed with status "X" in Test Pass "Y"}}{\text{Total no of TCs in Test Pass "Y"}} \times 100$
Total TCs executed by status	% of TC executed by status in relation to the total no of TCs executed	$\frac{\text{Total executed TCs closed with status "X" in timeframe A}}{\text{Total no of TCs executed in timeframe A}} \times 100$
TC efficiency	bugs per number of TCs executed	$\frac{\text{Total no of bugs logged against a TC in Test Pass "Y"}}{\text{Total no of TCs in Test Pass "Y"}}$
Test detection efficiency (defect capture rate)	% of bugs not found (escaped in Prod or found after the completion of QA efforts / during audit) out of total valid bugs reported by QA	$\frac{\text{Total no of bugs escaped}}{\text{Total no of valid bugs reported during testing RC build}} \times 100$
TC burndown rate	Test Cases executed per hour	$\frac{\text{Total TCs executed}}{\text{Total hours of scripted testing execution}} \times 100$
Requirements coverage	% of requirements / areas / components covered by TCs	$\frac{\text{Total no of requirements/areas etc covered by TCs}}{\text{Total no of requirements / areas / components}} \times 100$

Test case fail rate	% of runs where a given TC was failed out of the total number of Test Passes containing the TC	$\frac{\text{Total no of Test Passes where TC "Z" failed}}{\text{Total no of Test Passes where TC "Z" was included}} \times 100$
No. of new builds brought in testing	Buildcount	$\frac{\text{Total new builds brought in QA Monday + same for Tue + ...Fri}}{5}$
Smoke Fail Rate	% of failed smoke tests out of total smoke tests executed	$\frac{\text{Total no of failed smoke tests}}{\text{Total no of smoke tests executed}} \times 100$
TC Automation %	% of automated TCs	$\frac{\text{Total TCs automated}}{\text{Total active TCs}} \times 100$
Mitigation rate	% of risks mitigated out of total risks identified	$\frac{\text{Total number of risks mitigated}}{\text{Total valid risks identified}} \times 100$
Mean Time to Failure	hours	$\frac{\text{Total playing time}}{\text{Total no of crashes,disconnects \& critical failures}}$ average continuous playing time elapsed until a critical failure forces the user to leave a game session
Partner Satisfaction Rating	% PSAT score	$\frac{\text{No of positive scores (ratings 4 and 5)}}{\text{Total scores offered}} \times 100$
Framerate Average	Average framerate value	$\frac{\text{Sum of framerate value measured across all checkpoints}}{\text{Total number of checkpoints}}$
Framerate Minimum	Minimum measured framerate	The smallest value for frame rate across all measurements
Battery Consumption Rate	% of battery drained during 30 minutes of continuous gameplay	The % decrease of battery level between the level displayed at start and the one displayed after 30 mins of continuous gameplay
CPU Average	Average CPU usage value	$\frac{\text{Sum of CPU usage value measured across all checkpoints}}{\text{Total number of checkpoints}}$
App used RAM	Average RAM usage value for AUT	$\frac{\text{Sum of RAM usage value measured across all checkpoints}}{\text{Total number of checkpoints}}$
Packet Loss	% of packet loss	$\frac{\text{Data packets lost}}{\text{Data packets sent in total}} \times 100$
App Size	Size of the app (MB, GB)	Usually this is measured once right after the installation is successful (fresh install size) and a second time after the first game session is completed (app size)

KEY TAKEAWAYS: TURNING METRICS INTO MEANINGFUL QA DECISIONS

- **Metrics are tools, not goals.**
Numbers only create value when they support better decisions, clearer priorities, and healthier teams.
- **Context matters more than volume.**
The same metric can tell very different stories depending on scope, project phase, and constraints.
- **Bad KPIs can do real damage.**
Poorly defined or misaligned metrics often drive the wrong behaviors, reduce trust, and hide real issues instead of solving them.
- **Junior teams need structure, not just data.**
Metrics become powerful when they are clearly defined, consistently applied, and explained in ways that build confidence, not confusion.
- **Good QA metrics support planning, forecasting, and accountability.**
Used correctly, they help teams estimate effort, manage risk, and align expectations across stakeholders.

HOW AMBER HELPS TEAMS MAKE METRICS WORK

We build QA metrics frameworks for real projects, defining practical, project-specific measurements that reflect real workflows, not abstract ideals.

We design and validate KPIs, from SLAs to internal performance indicators that are measurable, fair, and decision driven.

We turn data into QA leadership, with experienced QA Leads translating raw numbers into insights for staffing, planning, and quality strategy.

We provide scalable QA support, adapting metrics and reporting across Full Game Dev, Co-Dev, Live Ops, and long-term partnerships.

We coach growing QA teams, helping junior and mid-level testers become confident, metric-literate contributors without overwhelming them.