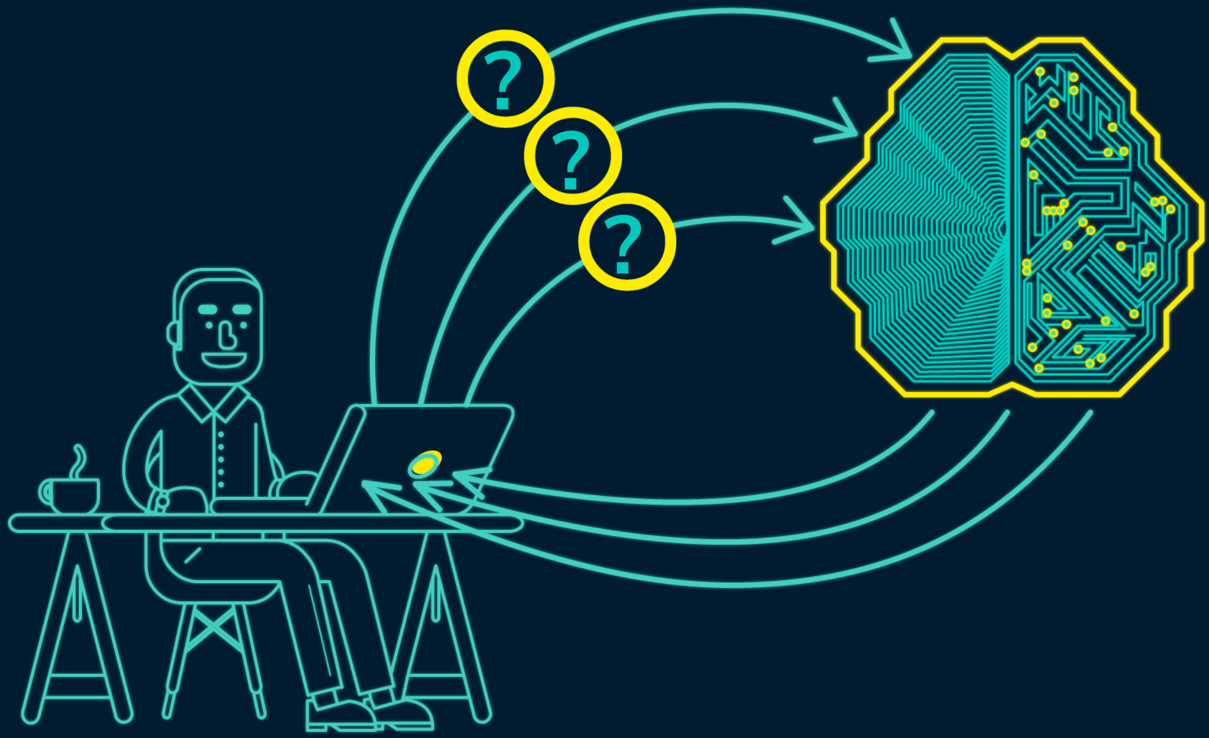
Sealights

QUALITY: THE MISSING METRIC FOR SOFTWARE DEVELOPMENT MANAGERS



Background

Development managers are inundated by metrics - from code coverage to lead time, velocity, MTTR and customer satisfaction. But these metrics are not enough to really understand what is happening “under the hood.” How good is the software being developed, and how likely is it to meet customer requirements when it hits production?

The Achilles Heel of current metrics is quality. Management has very limited visibility into the level of quality of software projects under their responsibility. Required Investment in Testing and Production Readiness are two new metrics which can make a big difference in team leadership and build a more effective development organization.

This eBook Will Help You Understand:

- The common metrics used by development managers and “warnings” you should be aware of for each group of metrics
- The general challenges of current software metrics and evaluation methods
- Which metrics are necessary to provide development managers the visibility and control they need, and the damage caused by the absence of these metrics
- Two new metrics that can provide this missing visibility and help managers control and improve quality and customer satisfaction

Which Software Metrics are Most Used by Development Managers Today?

Formal Code Metrics

These are quantitative metrics that can be computed based on source code. Their objective is to discover the quantity and quality of code, which is the main work product of dev teams:

- Static code analysis—Automatic analysis of source code to evaluate conformance to coding guidelines, and discover coding errors and bugs. Can provide a limited, but easy to obtain, measure of code quality.
- Other code metrics—Lines of Code (LOC), code complexity measures, Instruction Path Length, etc. These metrics were used in the past to evaluate and estimate development work, however, in agile environments, they are considered less meaningful.

Metric warning: While these metrics can provide useful information about the work products of dev teams, they cannot capture the essential aspects of code quality. The modern alternative to quantitative code metrics is manual, peer code reviews. These are extremely effective but do not generate a quantitative software metric you can track.

Developer Productivity Metrics

These are metrics that can help managers understand how much developers are achieving and how effective they are:

- Active days—How much work time a developer spends contributing code or other work directly affecting the software delivery project, excluding planning, administrative tasks or working on other projects. Also known as “capacity” in agile methodologies.
- Assignment scope—The amount of code a developer can maintain in a year. This is used to plan the size of a team required to support large-scale software systems.
- Code churn—When developers revise, rewrite or delete their own code. If on an average day a developer checks in 100 lines of code, but 70 lines replace previous code they checked in on previous days (70% churn), something may be wrong.
- Efficiency—The amount of working code contributed per software developer, taking into account churn. Code that is replaced or modified in subsequent builds or sprints should be discounted or devalued to reflect the developer’s long-term contribution.

- Impact—Measures how “big” code changes are, beyond simple lines of code measurement. The assumption is that the more the developer touches existing code, the more files are affected, and the higher the impact on existing code compared to the project average, the more effort was spent on a specific check-in of code.

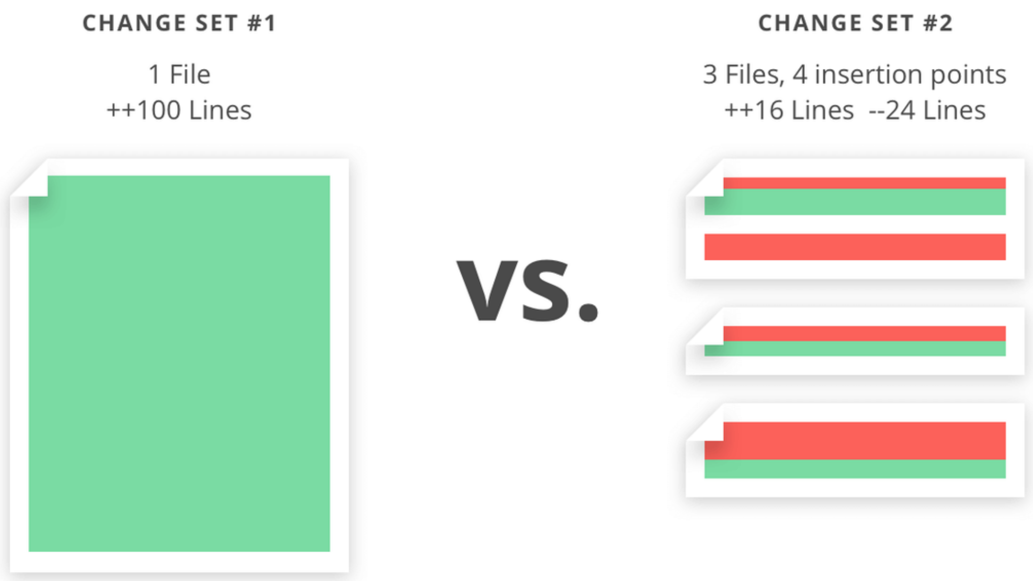


Image Source: GitPrime

Metrics warning: All these metrics (except “active days”) make two assumptions that are frequently untrue:

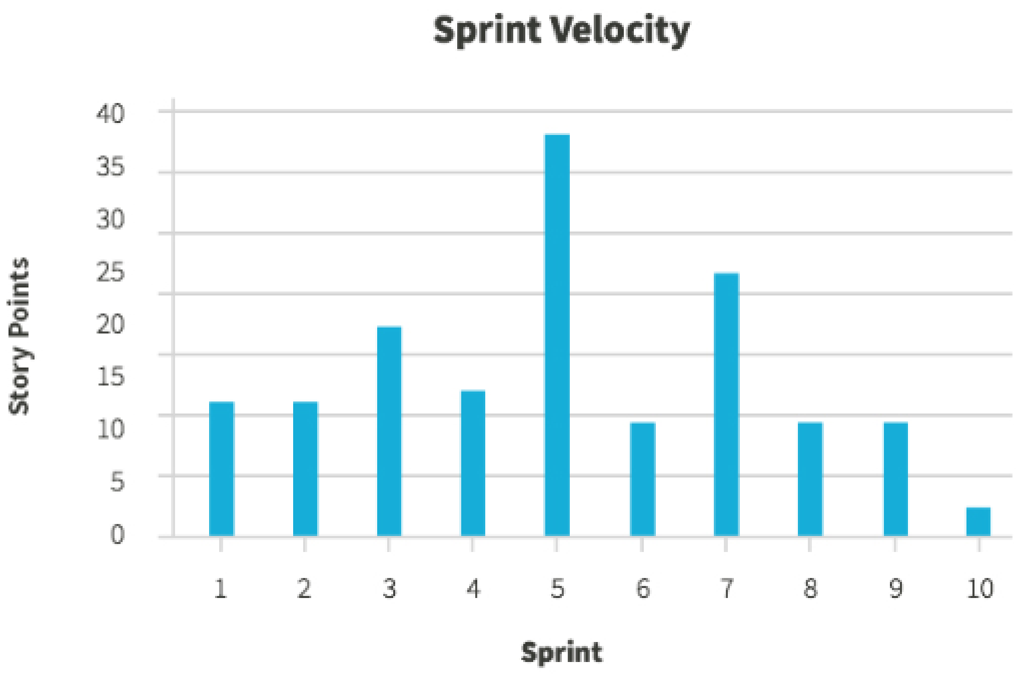
More code = higher development productivity
More changes / refactoring = lower developer productivity

Expert developers write elegant code which can do a lot in only a few lines, but this would be measured as low productivity. Conversely, when a code is revised or refactored it is devalued as churn, whereas effective refactoring of old code can provide high value.

Agile Process Metrics

These are metrics that measure developer progress from an agile perspective:

- Lead time—Average time from idea to delivered software.
- Cycle time—The time it takes for a change in software to be deployed to production.
- Velocity—How much new functionality, usually measured in story points, a team delivers over several iterations.



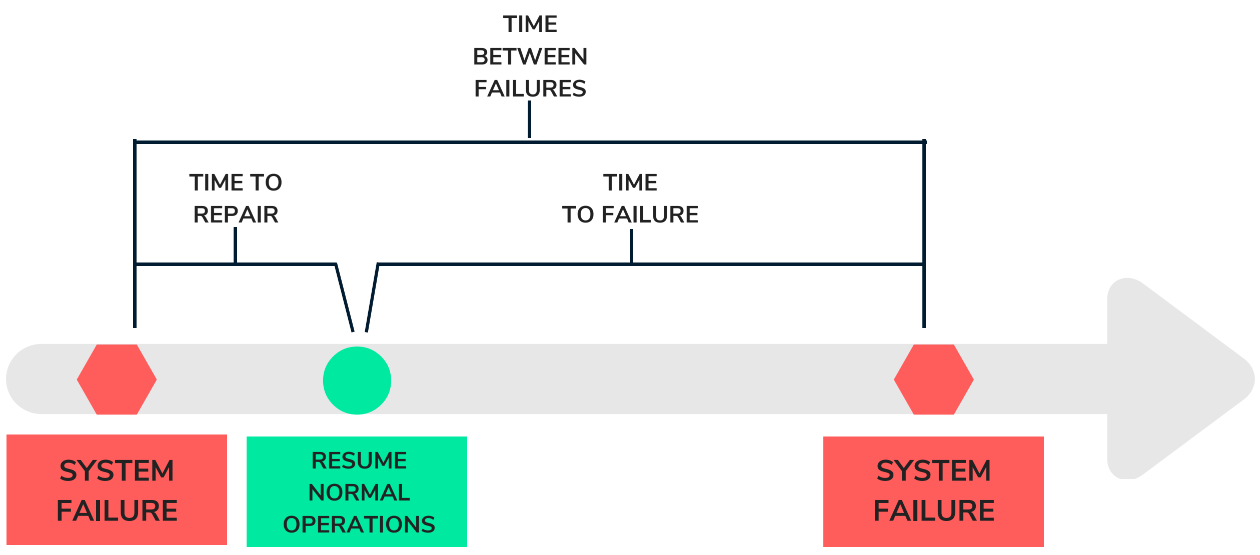
- Open/close rates—What percentage of production issues reported in a given time period, was closed within that same time period. For example, 15 production bugs reported in a week and 10 resolved in the same week is a close rate of 66%.

Metrics warning: Agile metrics do not provide a complete picture of quality. Teams moving very fast and delivering software rapidly would look great with these metrics, even though they might be trading high velocity for comprehensive testing or bug fixes.

Operational Metrics

These metrics check how software is doing in production and how well ops teams are taking care of problems:

- Mean Time Between Failures (MTBF)—A period of time the software typically functions without major disruption. This is a measure of stability.
- Mean Time To Recover (MTTR)—The typical time it takes to restore the software to normal operations after a major disruption.



- Application Crash Rate (ACR)—Mainly used for mobile apps, the number of app launches divided by the number of crashes.

Metrics warning: Operational metrics are of prime importance, but they are influenced by parameters which are outside the control of development teams. For example, an unreliable cloud service causing faults would cause MTBF to shoot up, while the underlying software could be very stable. Conversely, mechanisms like high availability, automatic failover, and cloud provisioning could improve MTBF and MTTR, “dampening” real issues in the software product.

Test Metrics

Test metrics can help shed light on how comprehensively a system is tested, which presumably is in high correlation with software quality:

- Code coverage—A rudimentary but useful metric showing how many lines of code or code statements are covered by unit tests. Does not take into account other types of tests like UI automation, integration tests or acceptance tests.
- Percent of automated test coverage—How many of the application's tests have been automated. This is used to measure the maturity of the testing infrastructure.
- Escaped defects—How many serious faults affected production users.

Metrics warning: The expectation of management is that test metrics would provide a measure of software quality. However, measuring software quality is difficult, and involves intangible aspects like maintainability, code quality and level of risk of undetected bugs in the codebase.

At best, test metrics can tell you what teams are doing to achieve quality, not whether they have actually achieved it.

Customer Satisfaction

There are several standard customer satisfaction metrics which can be used in software projects:

- Net Promoter Score (NPS)—How likely customers are to recommend your software and why.
- Customer Effort Score (CES)—How easy it was for a customer to complete an interaction with the company or product.
- Customer Satisfaction (CSAT)—How happy the customer is with the software they used. Typically uses Likert scale questions between 1-5 with 5 being "highly satisfied" and 1 being "highly unsatisfied."

Metrics warning: Customer satisfaction is considered, rightly so, to be the "be-all-and-end-all" metric. However, customers are not aware of the inner workings of the software product they are using and may not be able to identify specific software issues. Customer satisfaction can identify a problem, but in most cases it won't point you directly to a solution.

Challenges with Software Metrics

Good metrics are critical for managing large-scale business activities effectively. Dev teams need to understand how they are being measured and will optimize their activity appropriately. Managers need a feedback channel to quickly understand how teams are doing, which projects are on track, and which require improvement.

In software development, there are several inherent problems with metrics used by teams and their managers:

- Software is intangible—Software is inherently difficult to measure. Treating the code as a "work product" that must be analyzed leads to a very narrow assessment of software, which usually does not reflect value delivered to customers.
- Software metrics are difficult to standardize—It is common to see different metrics, or the same metrics with different meanings or formulae, used across dev teams. Agile methodologies encourage this by allowing each team to pick its own definition of story points which define the team's velocity, and their own Definition of Done. Non-standard metrics make it much more difficult for management to compare the performance of teams, departments, and projects, and see how they are trending over time.
- Software metrics can drive non-productive behavior—It is natural for anyone to modify their behavior in accordance with their performance metrics. If we encourage and reward an increase in LOC, developers will try to write a lot of code, even if it isn't tight or efficient. If we encourage and reward code coverage, developers will put a lot of effort into unit tests at the expense of other types of tests which may be more important.
- The most important things are difficult to measure—In an MIT evaluation of software metrics, Prof. Nancy Levinson says that "programmer ability swamps all other factors in factor analyses." The metrics we have today cannot measure "how good" a programmer is or "how good" is the code they have written.

What is Missing? An Effective Metric of Software Quality

As a result of the limitations of today's software metrics, development managers don't have the metrics they need to effectively evaluate and lead software development projects.

It is well known what is being produced and at what pace. What is less clear is what is the quality of the software, how likely it is to satisfy customer requirements, and what are the risks of quality issues that will hurt customer satisfaction.

The lack of real quality metrics results in:

- Limited visibility over large-scale development efforts
- Limited ability to define goals and ensure they are met
- Limited control over the return on large R&D investments

Conversely, adding real quality metrics can improve the ability to evaluate teams, guide them in improving their performance and drive more ROI from software development initiatives.

The Missing Metrics—Investment in Testing and Readiness to Ship

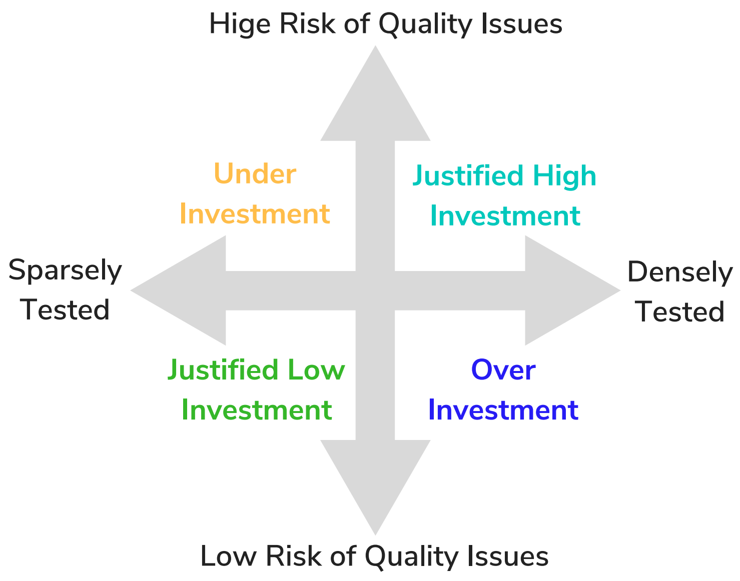
We suggest two new metrics that can change the game and provide real visibility into "what's inside" the code that the software dev teams are producing.

Investment in Testing Metric

This metric shows how much the organization is investing in testing, compared to the actual level of quality risk in an application. The metric looks at two sets of data:

- Which parts of an application are at higher risk of quality issues because they have new functionality, code changes, or because they are used frequently in production.
- Holistic test coverage—Which parts of an application are tested across all testing levels; not just unit tests but also UI tests, integration tests, end-to-end tests, etc.

This metric, when applied to a software portfolio, can divide projects into four quadrants:



Development managers can take different actions in each quadrant:

Quadrant	The Situation	Management Action
Under Investment	Software that is at high risk of production issues and doesn't have enough tests	Invest more in quality to reduce the risk
Justified Low Investment	Software at low risk of production issues (e.g. legacy software) with low test coverage	Do not invest more in quality, focus on maintenance
Justified High Investment	Software at high risk of production which has good test coverage across all test levels	Provide enough resources to maintain the current high level of quality
Over Investment	Software at low risk of production issues but with an unnecessarily high level of test coverage	Retire some test infrastructure, divert resources to other applications

Readiness to Ship Metric

This metric measures the likelihood of different software versions to break in production. It looks at the same data as the Investment in Testing metric - which parts of an application are at higher risk, and the holistic test coverage.

However, here the focus is on evaluating software versions released by a development team. The metric can assign two numeric values to each release:

- Number of test gaps—Important and untested features
- Holistic test coverage percentage—How comprehensively the release is tested as a whole

The lower the number of test gaps and the higher the test coverage percentage, the more a release is ready to ship.

Here are a few ways you could use the Readiness to Ship Metric:

- Readiness to Ship trend across sprints—As a team produces more user stories and gets closer to completing a release, is Readiness to Ship increasing, decreasing or remaining stable? If decreasing, the team is emphasizing velocity over quality.
- Readiness to Ship across teams—Are one team's builds more stable and ready to ship than another? Might be worth sharing their best practices with the other teams.

- Readiness to ship across products and product lines—Are specific products or product areas at higher risk? Slow down and ask teams to focus on quality.

Using these two metrics, development managers can get a good idea of "how good" is the code developed and shipped by development teams, and what action is needed to reduce risk and improve customer satisfaction.

Quality Intelligence: Delivering Next-Generation Quality Metrics

In this white paper, we communicated the power and value of having metrics like Investment in Testing and Production Readiness at your disposal, however we didn't address where the data comes from or how exactly to compute the metrics.

In reality, these metrics are not available to the vast majority of development organizations, for two reasons:

1. Missing data—There is no central repository of data showing quality risks in applications and tests across all testing levels.
2. High complexity—Complex software projects have thousands of tests, millions of lines of code in constant flux, and a large number of build artifacts. Calculating risk or holistic test coverage, even if the data was readily available, is a formidable task.

This is where Quality Intelligence technology comes in. It provides visibility for software development teams by:

- Monitoring tests and test frameworks, including unit, functional, UI, integration, end-to-end, and manual tests.
- Collecting data about which tests were run for each software version and their results.
- Tracking code changes, to identify major changes and whether they are tested.
- Tracking code usage in production, to identify which features are in active use and which are "dead code."
- Correlating tests with code changes and production usage, to identify high priority features for testing.
- Visualizing the data to provide visibility into which tests are not really needed and represent wasted effort, and which parts of the software are at risk of quality issues.

SeaLights is a quality intelligence platform which provides the data necessary and the intelligence to compute meaningful metrics like Investment in Testing and Production Readiness.

SeaLights combines data about code changes, production uses and test execution, to provide:

- Test gap analytics—Identifying areas where the code was recently changed or executed in production but has not been tested. Test gaps are the best place to invest R&D and QA resources to improve quality.
- Quality trend intelligence—Showing quality trends over time, per system component. This shows which parts of the system are improving in quality coverage, and which are getting worse. meaning more testing time should be invested.
- Release quality analytics - SeaLights performs real-time analytics on hundreds of thousands of test executions, code changes, builds and production events to assess the readiness of a release. Which build is best and provides the highest quality for users?

A Quality Intelligence Platform completes the missing metrics sorely needed by development managers and provides actionable insights that can steer teams and software portfolios in the right direction.

Request a live demo of the SeaLights platform to see how quality intelligence can help you make better decisions and build a more effective development organization.