propelo

WHITEPAPER

Measuring Software Quality

The Path To Engineering Excellence

Introduction

As more and more organizations embrace Digital Transformation and Product Led Growth, the reliance of the business on engineering teams is growing. But so is the demand for accountability and efficiency. To be successful in this new era, Engineering leaders need insight into all aspects of the software delivery lifecycle, and visibility across all their teams and applications. Engineering leaders who have been running their organizations on gut instinct must replace it with a data driven approach.

Measuring Software Quality gives an indication of the output that the team is delivering. But simply measuring baseline metrics like Mean Time to Resolution (MTTR) or Change Failure Rate is not enough. To truly streamline the quality process, and improve quality, it is important to first have a good understanding of the various factors that impact software quality.

Table of Contents

10 Consideration when measuring Software Quality

- Requirements clarity & stability
- Scope creep
- Code/Test coverage
- Defect tracking and correlation
- SLAs for Resolution Times
- Bottlenecks Identification Process
- Security considerations
- Technical debt & Code Hotspots
- Deployment frequency
- Visibility across the organization

Considerations while measuring Software Quality

These metrics allow you to decide how often to deploy, quantify your release risk, and ship secure code. They facilitate pre-deploy decisions and help you tackle your post-deployment issues.

Requirements Completeness

Even before code development starts, there are factors within the planning phase that can eventually cause slowdowns in development.

The Product Management (PM) team defines the requirements that are given to the Engineering team to build. Requirements Specifications should include details of problem statements, business priority and use cases. They should include the user stories, their priorities, timeframes, interactions and other considerations so the Dev team can understand how the different pieces fit together. It can also include customer interviews, product wireframe & mockups, and other related content for full context.

Some considerations when looking at requirements are:

- **Detailed User Stories:** The user stories should be well defined, clear and concise to avoid any confusion. This ensures that the engineers know the details of every item and exactly what to build, and ensures that the code meets the PM specification and the customer needs.
- **Objective Prioritization:** Priorities should be assigned to each of the user stories.Priorities can take multiple factors into account including the business priority, customer requirement, customer escalation, competitive need, interaction between stories, etc. The eventual priority list can include stories from the same epic, or across different epics, based on the Roadmap. This helps the Dev team prioritize their workload.
- **Include Acceptance Criteria & Test Cases:** The Acceptance Criteria (AC) can include metrics such as time required to complete an action. It can also include QA notes so the engineer can test the code. Proper AC can help define what work items are considered “Done”.

5 criteria for the right outcomes:

- Is the problem statement approved?
- Is the Business Priority mentioned?
- Are customer use cases documented?
- Is the UX Design complete?
- Are acceptance criteria and customer environment specified?

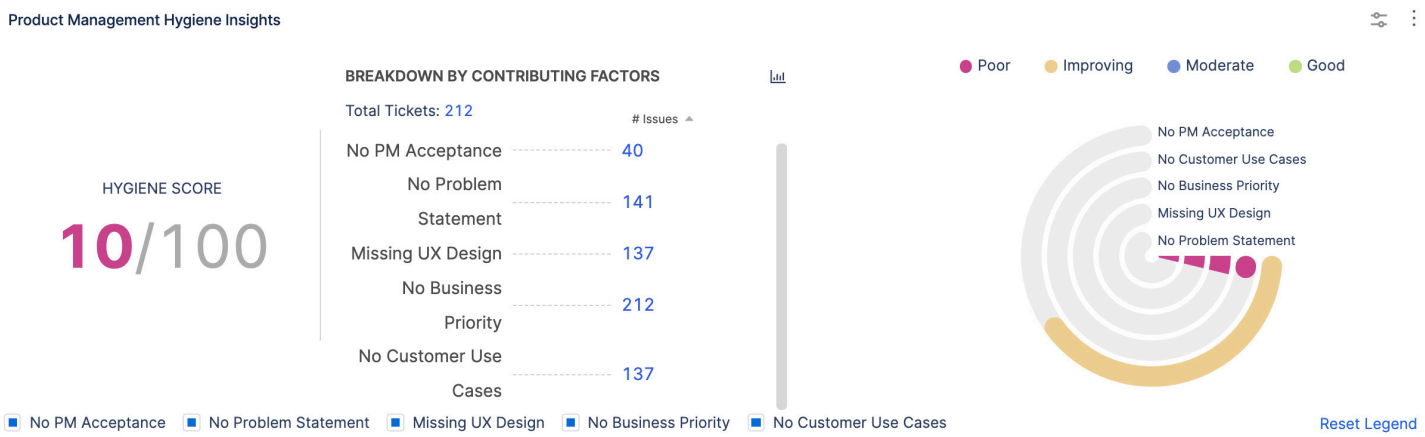


Figure: Product Management Hygiene Insights

If the requirements lack clarity or are incomplete, this can lead to back and forth conversations during the development process, resulting in delays. Worse still, if the requirements were not understood correctly, it can result in the wrong output and result in rework.

Any gap or mismatch between PM output versus Engineering expectations should be measured so it can be reviewed and rectified on an ongoing basis.

Scope Creep

Scope creep can be a result of unplanned work or from shifting priorities. Adding scope in the middle of a sprint can create churn and have a domino effect and derail the entire sprint. Trying to meet deadlines with added scope could result in poor quality work. This is especially true in the final days of the sprint when everyone is scrambling to get the work completed.

Creep could happen for a variety of reasons:

- Urgent bug/defect fixes that impact customers
- Changes due to competitive landscape
- Change in prioritization of features
- New features added from customer feedback
- Unclear requirements
- Unexpected complexity discovered during the Sprint
- And more...

Some of these issues - for example customer related issues or new competitive changes are unavoidable - they impact the business, or customers, or might simply be necessary for the growth of the product. Others, such as those arising due to unclear requirements, are avoidable with better planning.

Recognizing that even the best laid plans are subject to change, organizations must get better at managing scope creep rather than focusing on eliminating scope creep altogether.

Testing Effectiveness

Testing effectiveness is an important metric to monitor. To measure this, one must look at:

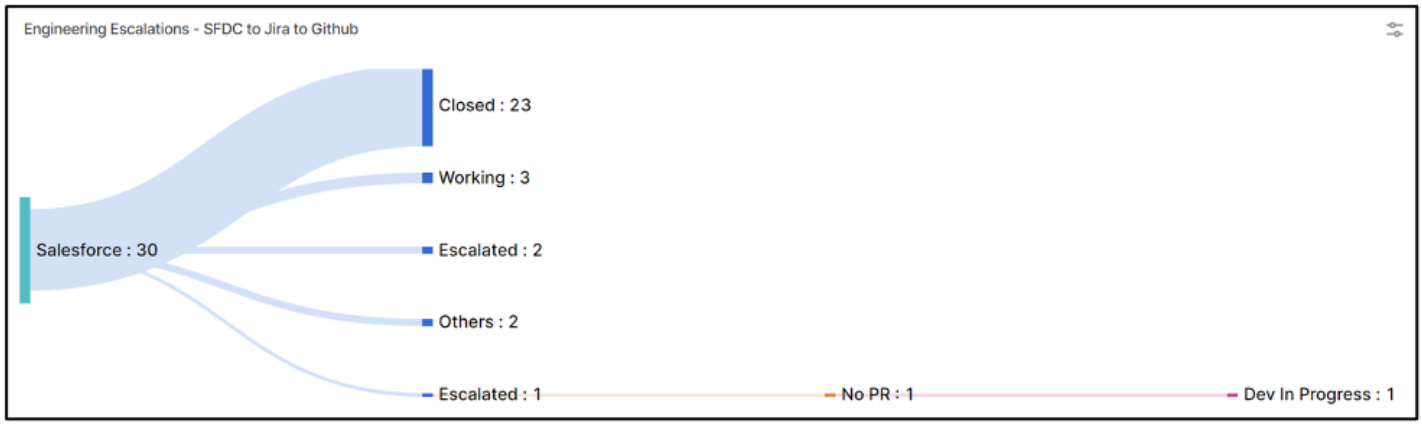
Code/Test coverage

As the team starts building code, right checks and balances need to be in place to ensure that the code quality is adequate. This involves design reviews and regular code/PR reviews.

It is also important to have the right testing strategy defined that covers all aspects of testing - unit, API, component, integration, load/performance, regression and security. Automation should be built wherever possible and manual testing should be well-defined.

Defect tracking and correlation

Defects are a reality of the software world. The testing team tries hard to reduce the number of defects that go out into



production. To further reduce the number of escaped defects and prevent recurring/repeat escaped defects, they should be tracked properly and then correlated with the areas of code they arise from. This helps with 2 things:

- Pinpoint "code hotspots" that defects frequently arise from
- Build more rigorous tests around known problem areas

One popular method to measure the efficacy of a test is to track the bugs uncovered by the test.

Another way is to measure the bugs found in-house vs the customer support issues and see where the gaps are. For example, If you are running 100 additional tests but not increasing either the internal bug count or reducing the customer issues, then clearly the 100 tests do not have the required impact and need to be revisited.

Resolution Process

The effectiveness of the resolution process is a very important metric.

SLAs for Resolution Times

Post-deployment, one of the most common metrics for quality is how many support issues a release generated. But this only tells part of the story. The other question that should be asked is how fast is the team able to resolve these support issues. A higher time for resolution could be indicative of a variety of issues such as poor debuggability, lack of resources or too many incidents.

By setting different SLAs for different projects/priorities and issue types, you are not only setting expectations, but you also can track issues that have missed the SLA and measure the median response time for different groupings—priority, component, assignees, etc.

Missed SLA analysis is a great way to look at failure points in the process and potential bottlenecks. Measuring the median response time for different groupings based on priority, components, assignees, etc can give you even more details.

> PROPELO MAKES IT EASY FOR DEVELOPERS TO SHIFT-LEFT SECURITY BY PROVIDING THE RIGHT INFORMATION A TIMELY MANNER, WITHIN THE TOOLS DEVELOPERS USE

KHUSHBOO KASHYAP
RISK MANAGEMENT LEADER, RUBRIK

Bottlenecks Identification Process

Measuring Resolution SLAs and the issues that miss SLAs indicates that there is a bottleneck somewhere, so the immediate question would be "Where is the bottleneck?". Understanding and easing bottlenecks are super critical to running a quality organization that is interacting with multiple dev and support teams.

Measuring how much time a ticket is spending in different stages helps determine the bottlenecks for that organization or service. For example, tickets spending a bulk of their resolution time in the backlog could be indicative of a resourcing issue. Too much time in Triage could point to the need for better debuggability and logging.

Technical Debt & Code Hotspots

Technical debt is something that is often overlooked by engineering teams. It is the result of issues that were created in the system that organically grow and become complex and hard to maintain. It starts on day 1 of the product, and continues to grow unless it is managed. It often manifests itself as quality, performance or security issues and causes customer escalations.

Sometimes, in order to meet deadlines, developers might take shortcuts or workarounds for certain issues, or leave out small things that might make the code more efficient. Sometimes things get pushed into the backlog, and other times not. These might be minor inefficiencies that they mean to fix in the next iteration, but due to constant time pressure, never get around to it. These inefficiencies can add up to a snowball effect after a few iterations.

They can create code hotspots - areas of code or code modules that consistently contribute to the most defects. Developers have to spend more time working around these hotspots, and this can impact their development velocity. They can result in overall slowness and impact quality or security.

Technical debt should be cleared regularly to reduce quality issues. In addition, there should also be more test automation

7 Reasons For Technical Debt

- Shortcuts used by developers to meet deadlines
- Incomplete understanding of the original design
- Complex code or poorly architected design
- Outdated infrastructure
- Legacy technologies and code
- Repo diversity - too many languages and technologies
- Product backlog

built for the known hotspots, to ensure that they are tested in every release and reduce risk.

By pinpointing code hotspots, teams can prioritize where they should increase their investments - in increased code/ PR reviews, integration & unit testing, design reviews or test automation.

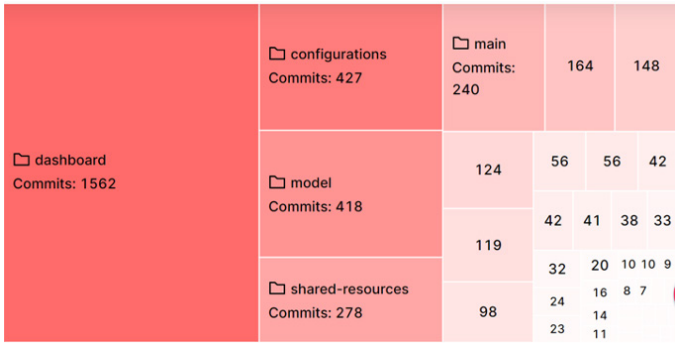


Figure: Code Hotspots

Security Considerations

Security is a critical aspect of every organization these days. But with rapid development and deployment, security testing is often left for the final steps of release planning. If security aspects are overlooked, you might have to start all over again, missing important deadlines. Finding security issues later in the cycle can result in rework and delays.

Many organizations are trying to "shift-left" on security and incorporate it earlier, as part of the overall development process. Incorporating security tools into the CI/CD pipeline and tracking vulnerabilities and best practice violations on the release development dashboard can be very helpful.

Deployment Frequency

As teams grow, it is critical to find a balance between how much and how often to deploy versus how stable the product is. By tracking deployment frequency and deployment sizes against incoming outages and customer issues, you can make data driven decisions on how often to deploy, so you don't have to compromise quality at the expense of velocity. No organization should be achieving higher software velocity at the expense of quality.

Visibility across the organization

One of the biggest challenges around quality is to drive visibility across the organization. People need to be aware of the important issues, and everything should be tracked so nothing falls through the cracks.

Automating the alerting and escalation process can help ensure that SLAs and bottlenecks have the right attention from the right team members, and can be resolved in a timely manner.

What to measure?

Identifying the right set of metrics is the first step towards measuring the development velocity of the Engineering organization.

The Baseline metrics: DORA

To determine the right software metrics to track the effectiveness and efficiency of software teams, the DevOps Research & Assessment (DORA) conducted some surveys, and came up with four key software metrics. These software metrics, known as DORA metrics, can be used to measure Software Delivery Performance.

- Lead Time
- Deployment Frequency
- Mean Time to Restore
- Change Fail Percentage

The following two are the DORA metrics to measure Software Quality:

Mean Time To Restore (MTTR)

MTTR is the time it takes to restore a failure in production. A failure can be an unplanned outage or a service failure. Measuring MTTR can be trickier than it appears because service failures and outages can be of different types or severity making the reporting of a single MTTR score incorrect and not-valuable. For instance, in many organizations minor and low severity issues are never prioritized to be fixed. Therefore these low priority issues should not be counted towards the final MTTR score.

Change Failure Percentage

This is the percentage of deployment causing a failure in production. It is the measure of the number of times "a hotfix, a rollback, a fix-forward, or a patch" is required after a software

PROPELO HELPS US OPERATIONALIZE ENGINEERING METRICS AND KPIS AND AIDS IN MAKING DATA DRIVEN DECISIONS

VISHAL SAXENA
VP OF CLOUD ENGINEERING, AKTANA

deployment or a service change. This is typically measured using data from the project management, version control and CI/CD systems.

Beyond DORA: Advanced Metrics for further insight

In today's complex world of technology, multitude of services interact with other services in complex ecosystems, with third party integrations, accessing and manipulating terabytes of data at near real time speeds. "Do the features work" is no longer a sufficient question to track the quality of a product. The questions about quality have gotten more diverse and complex, spanning multiple tools, teams and initiatives:

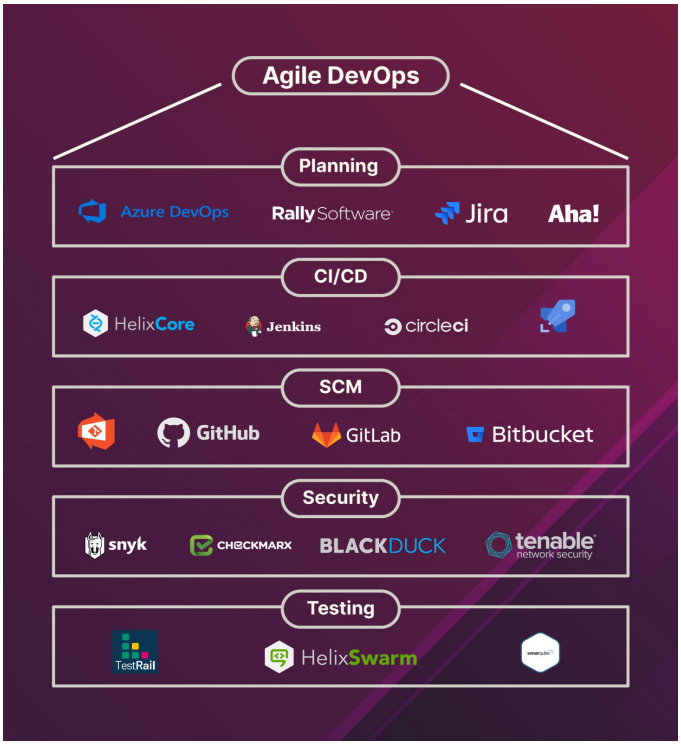
- Do the features work? How many test cases passed / failed?
- How much time is the operations team spending in deploying and maintaining the new service?
- How does the feature impact security posture for the product?
- Are there code hotspots that generate a lot of bugs?
- Is the product maintaining its SLAs and SLOs?
- How quickly can someone debug and fix a P0 issue for this feature?
- What is my deployment cadence and how is it affecting incoming defect rates?

Answering some of these questions can be quite complex since there are many things which can contribute to the quality of software - right from the requirements all the way to running things in production. DORA metrics give you a good foundation to measure the overall performance of your engineering organization. However, they don't necessarily give you further insight you need to determine the reason behind any quality issues.

The challenge getting the right metrics

Recognizing that there is a lot that can be measured, and identifying those advanced metrics is part of the equation. Being able to collect data to measure them is a totally different challenge.

- Multiple data sources: Data for the metrics comes from multiple systems across the DevOps toolkit - project management, SCM, defect management, service desk, CI/CD, testing, monitoring, security, etc. It is hard to get it all together into a single view.
- Data Hygiene: Even if it is possible to get data from different sources, correlating it can be difficult due to missing data. Certain fields might not be populated correctly. For example tickets might not be "groomed" properly.



Some companies have data organizations that are set up to do this type of analysis. However, those tend to be centralized teams that provide services to multiple other teams, and do not provide much flexibility in what can be done with the data.

How Propelo helps

Propelo correlates data from all the tools across your DevOps toolchain, and provides 150+ Metrics & Insights including DORA metrics that help engineering organizations improve their Agile efficiency. It is purpose-built to address the data challenge in engineering, and has highly sophisticated mechanisms to automatically manage the twin challenges of (a) a plethora and ever growing list of data sources and (b) the data hygiene challenge.

Propelo helps engineering leaders to prioritize their investments by examining areas which contribute to software quality. The data can be rolled up to the organizational level to get a big picture overview, but can also be broken down into the individual teams, applications or systems. This allows engineering leaders to drill down into problem areas and determine the improvement strategy.

Propelo's RPA allows automating repetitive, mundane tasks to reduce cognitive overload. Automated routing of issues, nudges and alerts for SLA escalations help reduce manual handoff times and also improve the process, delivering better quality. Workflow automations help maintain data and process hygiene.

"PROPELO PROVIDES US THE DATA-DRIVEN INSIGHTS ON HOW TO REDUCE DEVOPS FRICTION AT A VERY GRANULAR PER SCRUM TEAM LEVEL

JOE CHEN
VP OF ENGINEERING, BROADCOM

Conclusion

The quality of software produced by the engineering organization impacts the entire business. To get insight into the software quality and issues that impact it, it is important to measure various factors across the SDLC.

The data to get the right metrics can be pulled from different sources across your DevOps toolchain such as GitHub, Jira, Jenkins, PagerDuty, etc, and then must be correlated, which can be difficult and time consuming.

Propelo provides out-of-the-box software metrics and insights into the performance of engineering organizations. It aggregates & correlates data from over 40 tools across the DevOps toolchain, and centralizes it for easy consumption and reporting.

With Propelo, you get in-depth understanding of the factors that affect your teams, so you can prioritize and resolve the issues that have the most impact on your business.

Propelo is a venture-backed silicon valley company that serves customers from Fortune 100 companies to Series B startups. Propelo provides out-of-the-box software metrics and insights into the performance of engineering organizations.
Learn more about Propelo at propelo.ai or contact us at sales@propelo.ai

Propelo Inc.
700 S Bernardo Ave, Ste 103
Sunnyvale, CA 94087
sales@propelo.ai