

GO BEYOND VELOCITY WITH ADVANCED PRODUCT METRICS

Configuring a comprehensive
framework of metrics that
drive successful outcomes



Aurimas Adomavicius
President

CREATE
IMPACTFUL
PRODUCTS
**POWERED BY
METRICS.**

- 01 **Measure impact with metrics**
- 02 **Establish the framework**
- 03 **Product value metrics: Drive business outcomes**
- 04 **Quality metrics: Build sustainable software**
- 05 **Velocity metrics: Respond to market needs**
- 06 **Process metrics: Determine product and technical maturity**
- 07 **Build powerful products backed by metrics**

MEASURE IMPACT WITH METRICS

Evaluating ROI and quantifying risk increase in importance as organizations building custom digital products shift toward a product-centric mindset. Powered by data, metrics provide critical insights into the efficacy of products. The ultimate success metric of a product is adoption.

A product transformation, while impactful (e.g., an application transforms a company's operations), could take years—just long enough for the business to question the initiative. The journey requires time, quality execution, significant investment, and a well-defined path to correct course. Organizational structures, funding models, job descriptions, and expectations need to remain nimble to enable long-term shifts at the company.

Many technology leaders rely on classic project management techniques to track the success of individual initiatives and large-scale programs. A project is considered successful as long as it is delivered on budget, ships to production by the established due date, and includes the desired scope (aka functionality). These methods of measurement and accountability, however, severely underdeliver in context of a transforming organization.



DEFECTIVE METRICS

Good metrics are equally relevant to small-scale individual products and \$100 million portfolios. Both require focusing on outcomes versus activity. Bad metrics introduce counterproductive behaviors that:

Promote activity over progress.

Engineering managers track the productivity of an individual based on the number of lines of code written within a given time. This metric incentivizes bad behavior, prioritizing a complex over-engineered approach over simple, elegant design.

Elevate features over outcomes.

When project managers track the completion of original scope, the individual contributor prioritizes functionality over findings from the end user. This leads to products that are overengineered based on biased assumptions, instead of market data collected from real customers.

Lack trending.

Classic project metrics focus on a snapshot in time (e.g., 65 percent budget spent, 45 percent of scope complete) over a holistic view. These indicators provide a limited view of success (or failure), lack historic context, don't consider whether performance gets better or worse, and fail to inform the executive team of what lies ahead.

Are too granular for process change.

Organizations shifting into a product-centric mindset need to quantify and track the benefits of switching away from classic methodologies. Quantifying improvement of time to market as well as business value justifies the necessary investment.

Punish instead of teach.

Deviation of the scope, slipped timelines, or increased project budget do not provide information on the root cause of the team challenges. These same metrics, however, are mostly used to punish and blame parties that fail to deliver over time, creating an environment of silos and defensive behavior.

No matter the role, all team members rely upon metrics to evaluate successful outcomes. Product managers with agile teams need a level of granularity in their metrics to incentivize self-awareness and promote effective behaviors. The development team (e.g., designers and engineers) need to test and evaluate the elements they're building. Product leaders driving the vision on a particular program need effective data to justify the investment to communicate success throughout the product journey, evaluate the effectiveness of the product team, and course correct the product roadmap to hit desired outcomes. Sponsors managing a portfolio of transformative efforts require a dashboard-like experience to aggregate product metrics into actionable cues in order to warrant continued spending, terminate a program, invest in new opportunities, or adjust the execution strategy.

This white paper covers the challenges of traditional project metrics and recommends tactics for self-governing product teams to use in order to build healthy products. I'll go over in-depth Devbridge's proven best practices for gauging product effectiveness, quality, team velocity, process maturity, and portfolio health. Being armed with a comprehensive framework of metrics empowers teams and enterprises to deliver results.

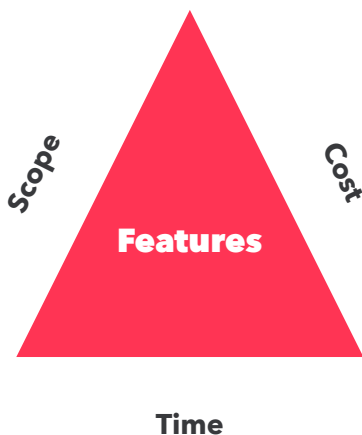
ESTABLISH THE FRAMEWORK

Selecting the right metrics, working toward the desired outcomes, drives adoption. A metrics framework fosters bidirectional communication for setting goals, attacking blockers, and communicating success. When designing the framework, consider the context and objectives of three distinct audiences:

- 1 the working agile team (engineers and designers)
- 2 the product manager (the person handling a single work stream)
- 3 the portfolio owner (the CIO or CPO)

A metrics framework facilitates bidirectional communication. Avoid pointing fingers. Focus on enabling teams to succeed.

Classic project management techniques teach the project management triangle. Its rigid structure implies that scope, cost, and time are interdependent. In this model, increasing scope for a project increases either cost or time, or both. Strategically, this approach fails to consider the effectiveness of the product (Is the right thing being built?), the quality of the product, and the time to market. Blindly following the budget, time, scope paradigm, and so on compromises the results.



As an alternative, try a square construct to monitor the product's quality, customer value, velocity, and organizational effectiveness.

Customer value establishes what's important for users to inform and guide the product build. Customers can be external or internal. Use the build-measure-learn loop to keep the team aligned to outcomes.

Quality measures escaped defects, performance KPIs, and the level of debt accrued. Intentionally sacrifice go-to-market speed (e.g., intentional debt) or industrialize for a mass rollout and mission-critical applications, which in turn lowers overall velocity.

Velocity determines the product go-to-market and feature release speed. This facet helps the team and product leaders project and plan throughput in a given time period.

Organizational effectiveness tracks the health of delivery and people processes (e.g., retention, communication, and engagement). As a result, the enterprise achieves more through better processes, higher engagement from the team, less churn, and the like.



Use the factors outlining the product square as boundaries. A product team needs to decide which of the four variables to invest in, which to sacrifice, and which to appropriate depending on the maturity of the product.

	CATEGORY	METRIC	USE	PRIORITY
CUSTOMER VALUE		Business outcome	Sets specific, measurable, achievable goals (e.g., loan approval reduces from two business weeks to one).	High
		NPS	Considers customer advocacy to understand what's valuable for promoters.	Low
		Customer health score	Tracks renewal, churn, depth of usage, growth of an account, etc.	Medium
		Product debt	Logs, prioritizes, and estimates the backlog product debt.	Medium
		Revenue channels	Evaluates the ability of the product to transact revenue through a channel.	Medium
QUALITY		Escaped defects	Helps the team understand the quality of the development output and testing strategy.	High
		Test coverage automation maturity	Determines the approach to test coverage, tools, methods, and acceptable coverage ratios.	Medium
		Technical debt	Monitors the various types of technical debt accumulated over time.	Medium
		Application performance	Calculates page load times, concurrency limitations, job run times, etc.	Medium
		Downtime and volatility	Indicates the overall stability of the platform and volatility to help the team reflect on improvements made through infrastructure updates and the testing strategy.	High

	CATEGORY	METRIC	USE	PRIORITY
VELOCITY		Sprint velocity	Demonstrates the quantity of software a team ships within a sprint.	Medium
		Velocity trending	Shows process improvement or degradation of team velocity (e.g., automation, DevOps, technical debt).	High
		Feature to production time (backlog aging)	Assesses how quickly a feature can be productized and released to market.	Low
		Estimate accuracy	Rates how confident and realistic the team is when estimating backlog stories. High estimate volatility requires root cause analysis to help the team improve.	Low
OPERATIONAL EFFECTIVENESS		Maturity score	Notes product best practices as well as technical maturity of the delivery framework.	Medium
		DevSecOps metrics	Monitors uptime, security, infrastructure stability, environment build times, and others.	Medium
		Team health	Evaluates the team's burnout, sentiment, and ability to succeed.	Medium

**FOUR
TYPES OF
METRICS
GET
RESULTS.**



PRODUCT VALUE METRICS: DRIVE BUSINESS OUTCOMES

Establish guiding principles for the application in the product vision document (also known as the product charter) and then reference them throughout the software's lifecycle. For example, a car loan application product enables customers to self-serve with features like remote contract signing and comparison of interest rates. The process saves the customer and business ops team time. Leverage an iterative build-measure-learn approach to evaluate success after each release and reincorporate learnings from recent releases back into the product.

The build-measure-learn loop is the governing technique to assure that product teams focus on what drives successful outcomes.

Avoid relying on a deep reporting hierarchy that wastes time and money. Instead, use value metrics for the team to self-inform and understand priorities. Measure the impact of a particular feature on a customer, while staying nimble enough to pivot when not achieving optimal results. Localizing decision-making empowers the team to:



- take full ownership of the product's successes or failures;
- make decisions faster to reach of desired outcome; and
- eliminate overhead (e.g., churn between the business sponsors and technology delivery teams).

Identify the metrics necessary to launch the product and align the team after the product hits the market. Upon achieving the initial outcomes, evaluate the following:

- how to quantify marginal improvement;
- how to monitor customer health;
- what financial metrics to include for ongoing investment;
- how much product debt exists; and
- what denotes value for accelerating delivery and reducing debt.

DEFINING REALISTIC OUTCOMES

Establish specific, measurable, and achievable outcomes.



-  Capture 50 percent of total market share. This likely won't happen because the expectation is too high and broad.
-  Automate approvals for 30 percent of all incoming loan applications. A proven best practice is to start by setting and measuring small goals as a point to validate the product.

Set three to five outcomes to start. The team needs to easily recall the target outcomes and recalibrate based on the metrics established in the product charter. When the time comes to release the product, adjust and evolve the desired outcomes.

ENABLING CUSTOMER SUCCESS

Balance the interests of business with customer expectations. Include metrics to evaluate how and if the desired business outcomes affect (positively or negatively) the customer experience. Select metrics in line with the type of product that measure customer detractors and promoters.



-  A lackluster experience and poorly performing internal company product inevitably causes frustration, damaging engagement and productivity.
-  Ease of onboarding increases adoption (e.g., autofilling forms, natural language forms).



Use metrics that evaluate the customer experience.

METRIC	MEASUREMENT
Net promoter score (NPS)	Notes the likelihood of customers recommending the product, predicting customer loyalty, and lifetime customer value.
Customer satisfaction score (CSAT)	Evaluates customer sentiment, which helps the business focus on implementing highly desirable features or pinpointing pressing issues to remedy.
Churn	Tracks the percentage of customers quitting the product in a given period of time, measuring retention or the lack thereof.
Expansion revenue	Calculates the percentage of new revenue coming from existing customers, measuring customer growth within the product.

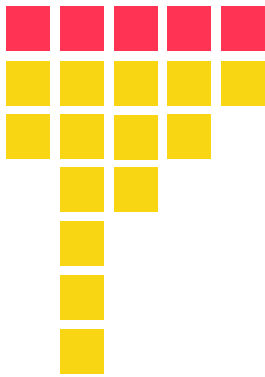
MANAGING PRODUCT DEBT

Product debt is the accumulation of product decisions that have a negative impact on the customer and business. Building successful custom software always leads to:

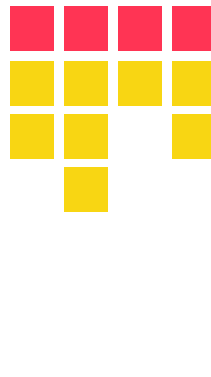
- a large backlog of desired features by business and customers;
- competing priorities, limited time, limited funding; and
- and a struggle between the core product roadmap and customizations for each client.

Keep a prioritized backlog of product debt from the release of the first version. Flag product debt stories initially compromised for one reason or another (e.g., speed, cost) for improvement. Once the product hits the market, review the immediate product roadmap, the technical debt backlog, and the product debt backlog captured during the planning session. Identified, estimated work is easier to manage, plan, and complete successfully.

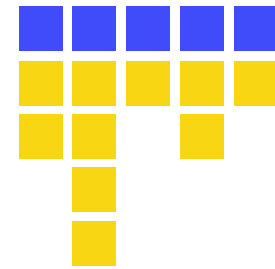
Backlog



Product debt



Technical debt



A bank leaves a mainframe system alone, paying licensing fees to run an old version that lacks modern security functionality or leaves system in a fragile state (i.e., customer data in a vulnerable state).



A bank maintains system software with the latest patches (e.g., security, functionality) keeping the software current (i.e., customer information is kept safe, system is easier to maintain).



QUALITY METRICS:

BUILD SUSTAINABLE SOFTWARE

Quality metrics track the shifting technical health of an application. The team needs to monitor escaped defects, automated test coverage, test run times, accumulation of technical debt, performance, and downtime. All products and custom applications require constant attention.

- New features are released.
- Technologies mature or deprecate.
- Team members come and go.
- The market shifts over time.

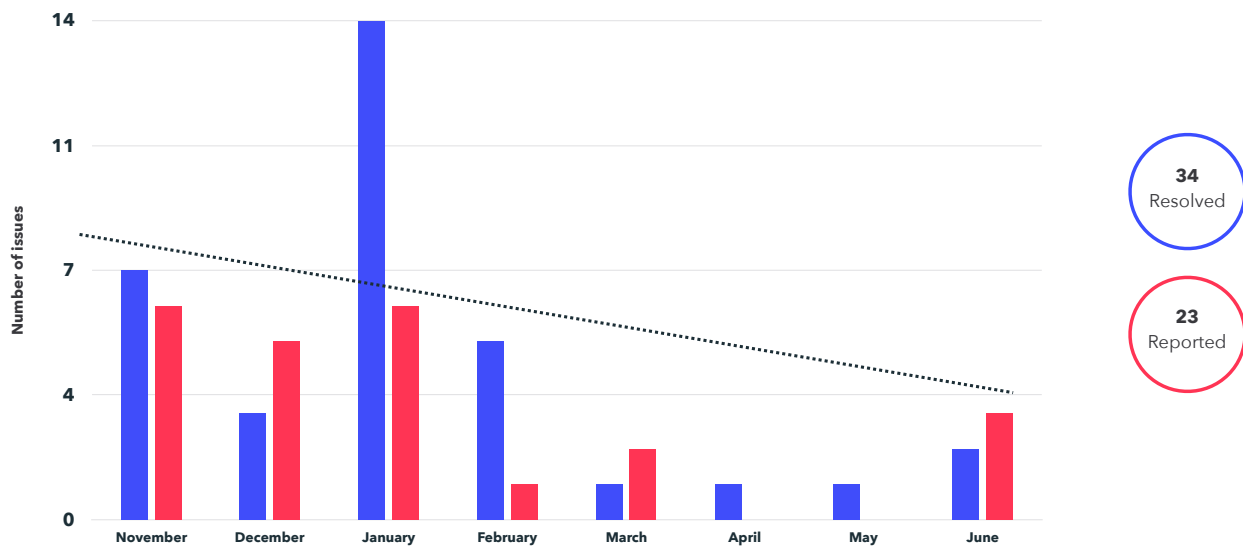
Software needs to adapt to changes within the business, market, and customer base. Scrapping the codebase and starting from scratch burns capital and time. To keep product evergreen, craft clear metrics. Modular design enables teams to maintain, refactor, and improve products incrementally.

Systemic observation and correction of quality metrics prevent software from becoming a legacy application. Treat all products as evergreen.

On the surface, business value metrics and quality metrics appear unrelated. However, failing to maintain a proper testing strategy or tolerating a volatile production environment in the long-term impedes business outcomes and customer success. It is thus important to establish a system of metrics instead of looking at each in isolation when prioritizing investment.

ESCAPE DEFECT REPORTS

Each user story goes through several development and testing stages before meeting the Definition of Done and being released into production. While techniques vary between Scrum, extreme programming, and test-driven development, our recommended best practice is for each engineer to own both implementation and rudimentary testing of a story. In other words, the feature needs to work before a story is handed over to testing engineers. Testing engineers provide an additional layer of strategy with multiple tools and methodologies at their disposal (e.g., end-to-end tests, unit tests, integration tests, interface tests, and manual tests).



To flag potential issues needing repair, track two types of defects:

- those caught by testing engineers within the sprint.
- those reported in the production environment that escape initial detection.

Testing engineers document both types of defects with a project management tool like Jira or Azure DevOps—allowing the team to report aggregate numbers over time. Detect and resolve defects in context of continuous improvement.

MEASURE	VALUE
Monitor escaped defects across various teams and look for outliers for teams with an extremely low or high number of defects.	Evaluating the testing strategy and domain complexity in both scenarios empowers the team to make educated adjustments to the overall testing strategy.
Review the root cause of each escaped defect in terms of the overarching narrative unearthed in the defect reports.	Understanding the root cause helps the team revise the testing strategy and prevent future defects.
Investigate internal defects within the team stemming from a lack of domain knowledge, onboarding, or skill set—causing significant loss of efficiency. <i>The recommended ratio of engineers to testers is around 3:1 or 2:1.</i>	Looking at the ratio of engineers to testers ensures the team operates efficiently. <i>A team operating with an equal number of engineers and testers may present issues with the testing strategy or the team's competence.</i>
Keep a record of defect metrics before and after a test automation strategy.	These numbers communicate the ROI of a sound strategy that requires investment.

AUTOMATION MATURITY & TEST COVERAGE REPORTS

A testing strategy encompasses functional and nonfunctional validation of the product (e.g., manual and automated performance, accessibility, user interface tests). Test coverage monitors the percentage of functionality covered by unit tests or end-to-end tests. Successful testing strategies lower the lifetime QA costs. The number of defects goes drastically down. Technical debt decreases. Delivery maintains a higher velocity.

A counterproductive industry expectation is for every product to have a large percentage of testing automation (e.g., 80 percent of functionality covered by tests). These standards become inefficient when the feature set being evaluated is low value or non-mission-critical. The spend becomes a financial burden, difficult for the business to justify.

The following are best practices to review when establishing metrics for automation maturity and test coverage:

Agree on automation depth. While automation is preferred, it is often an underinvested area of a testing strategy, especially when time to market is the focus. When it is critical to minimize the number of defects, however, full automation takes precedence over budgetary or schedule restrictions.

Embed elements in the toolkit. Include static and dynamic code analysis, test case management with TestRail™ or other tools, and integration to security monitoring tools.

Establish healthy documentation standards. Choose living documentation over static artifacts for historical testing evidence. Opt for a self-explanatory behavior-driven development (BDD) test to drive the acceptance of stories.

Collaborate with business analysis teams. Ensure test engineers work together with product managers and business stakeholders to expand the story acceptance criteria, as well as define developer checklists to verify when writing code for a story.

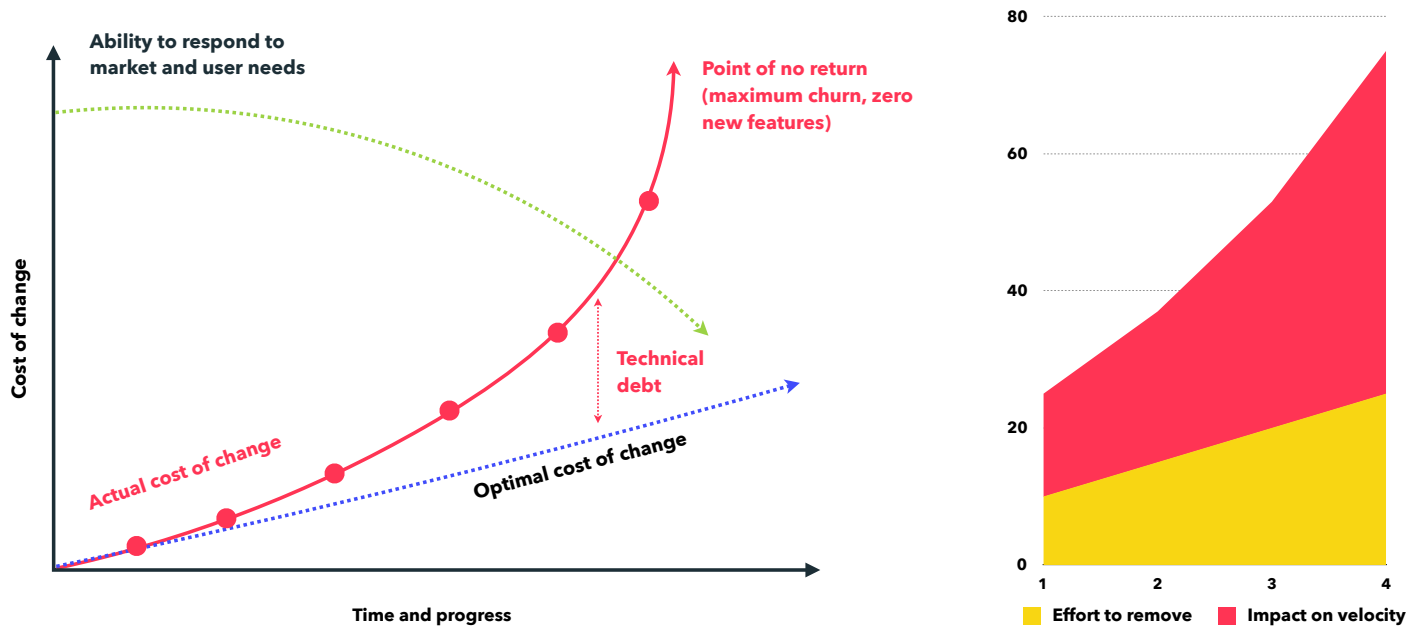
Integrate testing into the build pipeline. Include automated tests that run each time a code check occurs into the source repository and reject deployment when detecting defects.

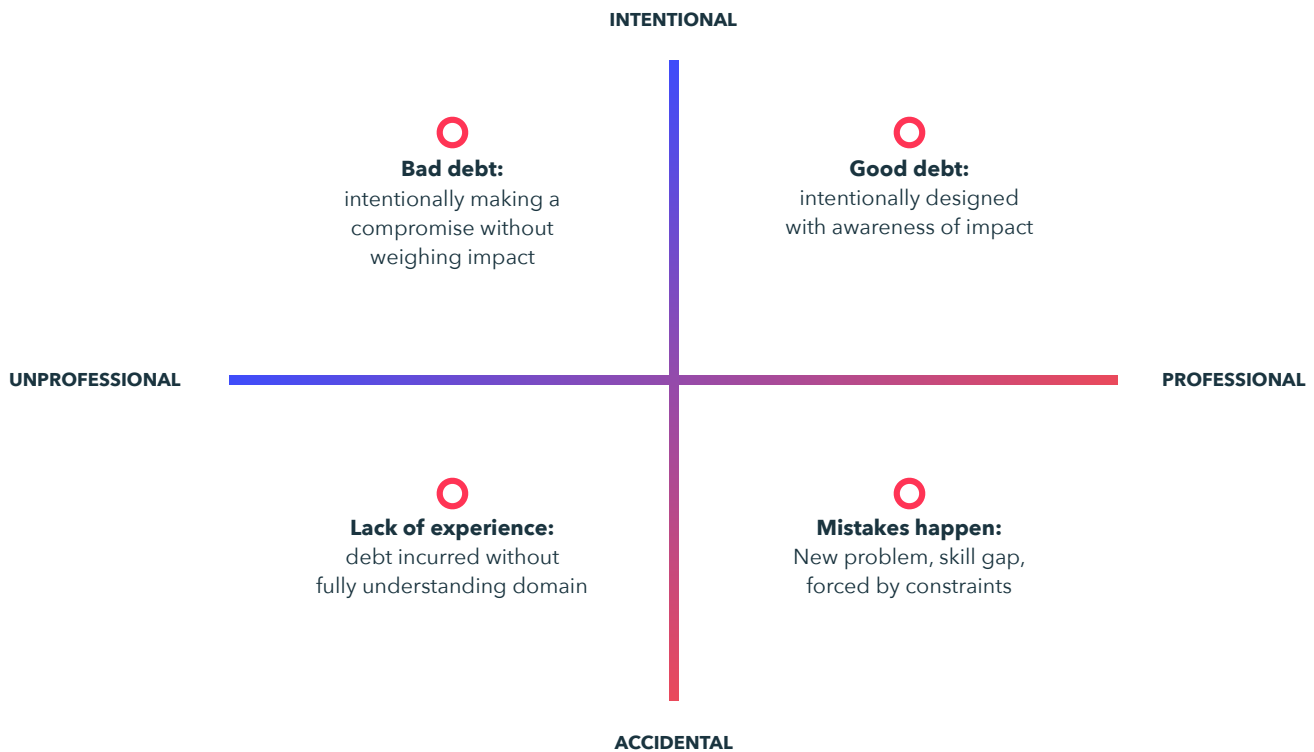
Automate all tracked metrics in Jira or a project management tool. Consider tracking the defect resolution time from discovery through release and defect resolution cost. Compare these metrics over time to determine if the testing strategy continuously improves the team's output and efficiency.

TECHNICAL DEBT REPORT

Technical debt is a software engineering term that describes the accumulation of undesirable decisions in the codebase. When writing code, the team settles for a quick fix over a better approach that takes longer to execute. While the compromise appears sensible, undesirable decisions in the codebase add up and require rework (aka technical debt). These decisions, intentional or unintentional, result in code that's hard to maintain, inhibits the product longevity, and lowers team velocity.

Teams always work within constraints of time, quality, and cost. Consequently, they incur debt to ship product to market faster. Accumulating some debt, especially when prioritizing market launch speed, is acceptable. However, too much debt has the power to cripple a team and product with performance issues or poorly designed architecture that deteriorates maintainability of software. To account for this reality, track the debt so that it's easily addressed in a future sprints of hardening.





1 **BAD DEBT** UNPROFESSIONAL & INTENTIONAL

Arises when writing bad code is created intentionally due to laziness, ignorance, or other unethical reasons.

THE FIX: Tackle technical debt and hold the team accountable for following best practices.

2 **GOOD DEBT** INTENTIONAL & PROFESSIONAL

Occurs when the team selects an easier, faster solution intentionally, fully aware of the long-term impediments.

THE FIX: Have a senior team member evaluate the pros and cons to determine whether the benefits of delivering outweigh the compromises.

3 **LACK OF EXPERIENCE** UNPROFESSIONAL & ACCIDENTAL

Stems from decisions that require rework after completing code review.

THE FIX: Use this type of debt as a vehicle for inexperienced team members to improve and learn.

4 **MISTAKE DEBT** ACCIDENTAL & PROFESSIONAL

Happens when a mature team with technical skills makes a bad decision with too little context or time constraints.

THE FIX: Allocate time to pinpoint and resolve mistake incurred by design decisions or changing requirements.

Similar to product debt, the main objective for a technical debt metric is to capture all known issues, estimate remediation, and continuously prioritize a certain amount of time for resolution. Create a technical debt burndown chart to track the remaining work against time. Beware if the backlog growth outpaces resolution velocity.

APPLICATION PERFORMANCE REPORT

A performance strategy correctly sets initial targets while also providing the testing framework to monitor and ensure ongoing compliance. Poor performance like a long load time detracts from a positive customer experience. Worse yet, performance issues frustrate users to the point where they leave and never come back.

To track efficacy, run tests and reports on the server-side and client-side to see how the application functions. Once in flight, analyze the performance requirements of new features proactively (e.g., data design). Then respond to the results from testing activities.

FOR USER-CENTRIC METRICS

Google's RAIL performance model is a solid starting point:

- **Response:** User input response occurs in 100 milliseconds or faster
- **Animation:** Display transitions and animations smoothly—60 frames per second
- **Idle:** Loading as little data as possible first and then using idle time to load the rest
- **Load:** Content appears in 5 seconds or faster

Try PerformanceObserver for client-side performance testing to track the first contentful paint (FCP), first meaningful paint (FMP), time to interactive (TTI), and more. For custom metrics such as single component initialization, render, or patch times, consider using a User Timing API. To monitor client-side performance, implement client app integration with a monitoring service (e.g., Kibana, Splunk, New Relic) and push client-side performance metrics to the monitoring service for continuous tracking.

FOR SERVER-SIDE METRICS

Track resource/endpoint response time characteristics such as median, average, error rate, and percentiles. Ensure that high percentages (95 percent or 99 percent) fall below five seconds under a high, but typical load.

JMeter is the gold standard when performance testing. Use monitoring services such as Kibana, Splunk, New Relic, or InfluxDB. The same instance of server-side monitoring services track client-side metrics as well.

FORECASTING PERFORMANCE NEEDS

Establish base target numbers to forecast future processes for a targeted implementation including the following:

- **The load hitting a feature.** Use existing data to determine patterns and necessary volume. For current products, refer to the latest monitoring to forecast the typical traffic on a specific feature. For new products, make educated guesses about concurrent users and feature usage frequency. Then make corrections once the MVP reaches the market.
- **The data variations/distribution.** Recognizing that many performance issues are data-driven, examine the application in light of datasets that may be used at high volume or concurrency. Include data research as part of exploratory testing. In the majority of cases, data exists and is accessible before implementation because most performance issues are data-driven. Simply query the required distributions from the respective databases or APIs.

BECAUSE OF HIGH COST, PERFORMANCE TESTING ENVIRONMENTS DO NOT HAVE EQUALLY BALANCED BACK-END RESOURCES THAT MIMIC PRODUCTION ENVIRONMENTS. TAKE THIS LIMITATION INTO ACCOUNT WHEN ESTABLISHING METRICS AND TESTING STRATEGY.

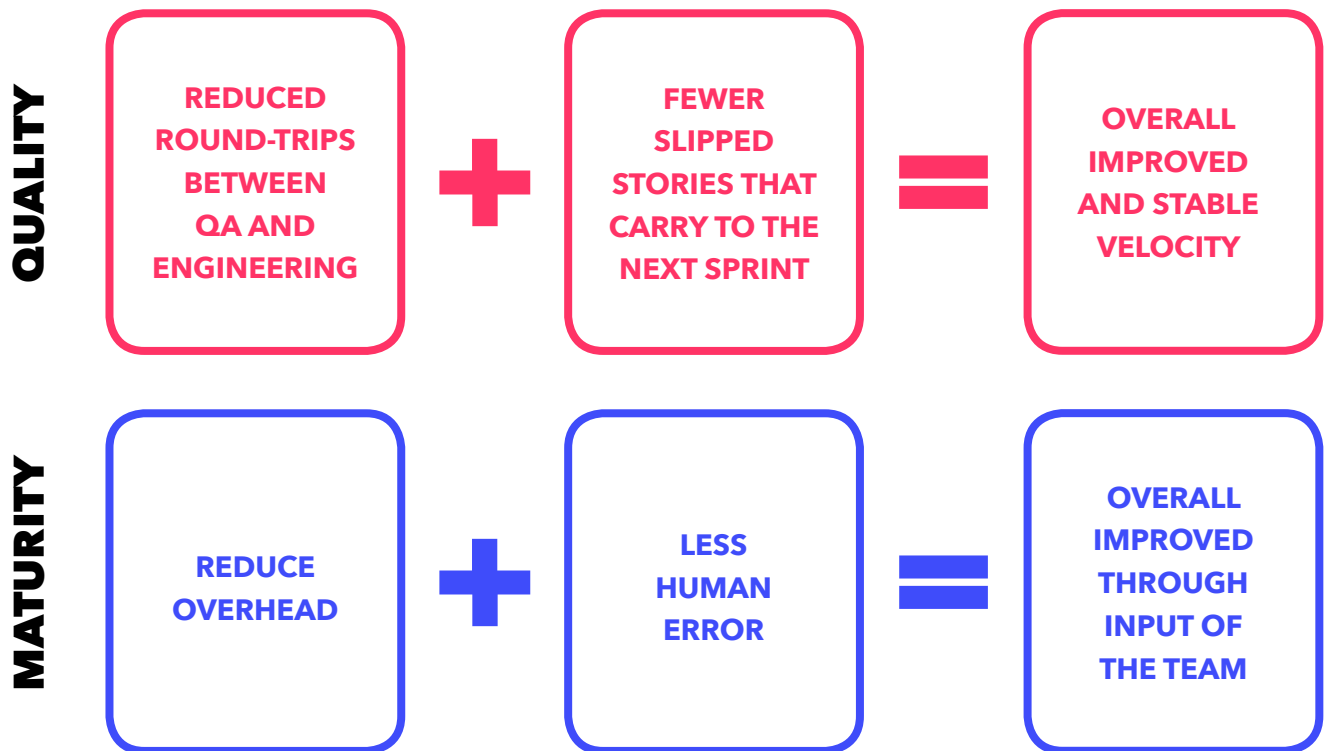


VELOCITY METRICS: RESPOND TO MARKET NEEDS

Predictability and speed to market are desirable for businesses. Velocity measures story points and represents the amount of work for a team to accomplish in a given sprint—noting the key milestones of the build. With an estimated backlog measured in story points, divide the total points by sprint velocity and project the number of sprints necessary to release desired scope to market.

Velocity helps the team project key milestones of the build. Using the estimated backlog size in story points divided by the average sprint velocity determines the number of sprints to release.

The quality and maturity of the organizational processes (e.g., mature DevOps featuring automated builds, tests, and code standards) influence velocity and prevent developers from shipping defective software.

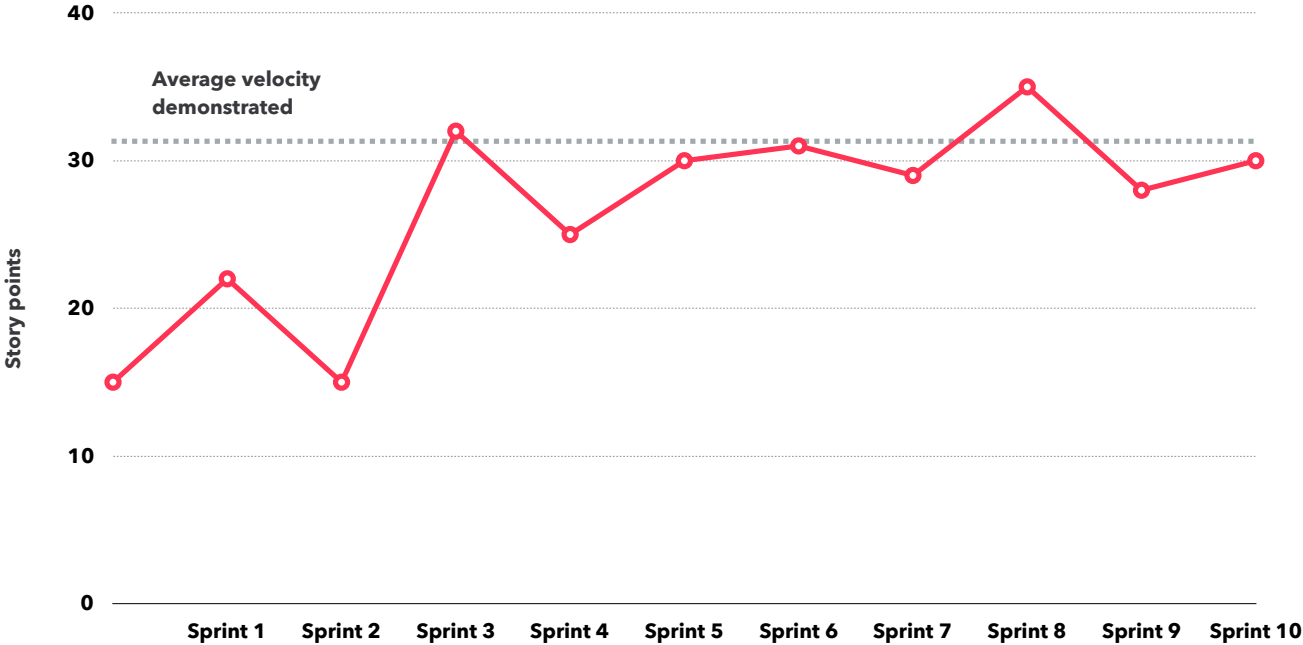


Velocity is not an absolute metric and should not be used as a means of comparison across multiple product teams. Story point estimates are based on arbitrary numbers from the Fibonacci sequence to indicate the anticipated complexity of a given feature. Estimation standards vary by team. For example, one team chooses to estimate a complex feature in nine points, whereas another uses fifteen.

No two teams are identical. Use velocity in historical context for a single team. The sprint velocity depends on the number of team members. There are a variety of tactics to manage teams at scale, including a velocity trending report, a cost per story trending report, a backlog aging report, and an estimate velocity report.

VELOCITY TRENDING REPORT

Each team should monitor its velocity over time. Sprint 0 normally starts slow. Building momentum takes time. After a few first sprints, a long-term stable velocity should emerge. Realistically, the report will have outliers. For example, a complex highly integrated feature throws off the data.



To monitor velocity over time, consider adopting the following tactics:

- **Analysis:** Analyze the root cause for spikes and dips of velocity. While normal, it's important to understand and anticipate shifts.
- **Trends:** Game velocity by assigning larger story point estimates to smaller features. Try to review estimates in parallel to the velocity trends.
- **Change:** When evaluating product maturity, there tends to be a slight drop in velocity once a product reaches a certain point of maturity. This often happens around the year or year and a half mark for a greenfield initiative. Normal and expected saturation of functionality, interdependencies, size, and number of teams all have an impact. There is a price to pay at scale.

Reporting velocity adds value by demonstrating ROI when an organization is going through a transformation effort. It's important to note activities that boost team productivity or increase adoption. Take time to document implementation of best practices (e.g., automated testing, CI/CD, code standards) that manifest higher velocity.

COST PER STORY POINT TRENDING REPORT

Track the cost per story point to evaluate the efficiency of work being completed. To measure, divide the total cost (i.e., burn) of running a team for a sprint by the number of story points delivered in the sprint. This report monitors velocity trends offset by adding or removing team members (e.g., difficulty identifying velocity gaming or loss of productivity because a team adds two engineers after eight sprints).



Accounting for both the cost of running the team and the scope shipped increases the average cost of a story point through time. The cost per story point typically experiences a slight uptick as the product matures or reaches a saturated state. Take time to analyze the root cause of these shifts. Left unchecked, the changes have the potential to cause damage or lead to inefficiencies that accrue over time (e.g., a testing strategy that no longer services the product adequately, outdated code standards, or slow build pipeline).

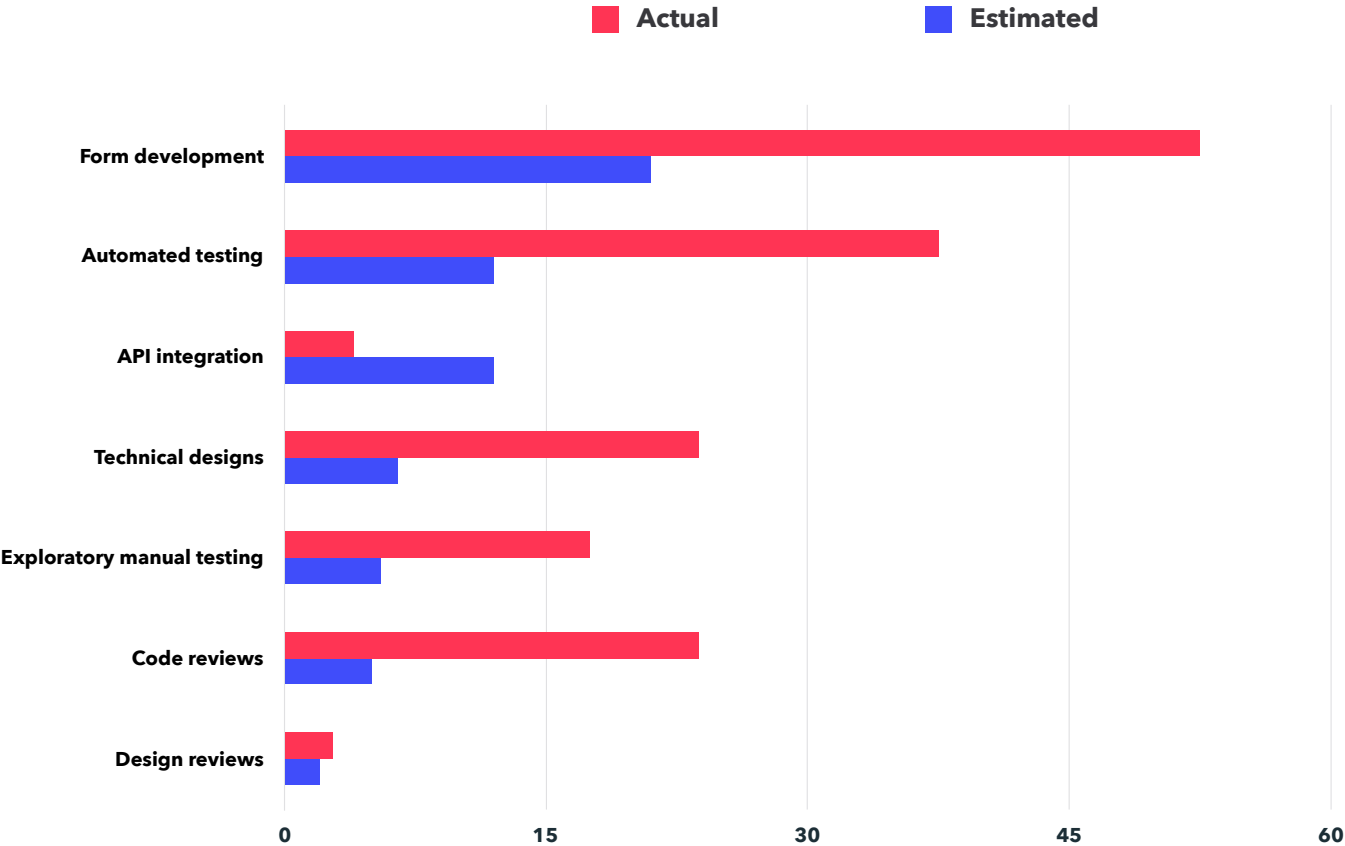
BACKLOG AGING REPORT

Backlog aging becomes useful when releasing a new version of a product to market, and new features start queueing up in the backlog. The report tracks how long a feature sits in the backlog before being shipped into production. Use this report to capture a snapshot in time.

- Was the business more or less responsive to market needs?
- Is the backlog growing too fast for the delivery teams to manage?
- Is the delivery team able to keep pace with demands and ship in a timely manner?

ESTIMATE VOLATILITY REPORT

Especially for custom product builds, developing sound estimates relies on forming educated guesses based on a variety of unique factors (e.g., prior experience, projected team size, project goals). Be sure to reflect on estimate accuracy as a healthy best practice. The data helps craft better projections going forward, especially for teams unable to reach Definition of Done within a designated sprint.



Track estimate volatility at the feature level. Individual stories tend to be too granular to showcase trends. Compare the original estimate to the actual effort required to ship the feature, MVP, or product. Here are a couple of insights from one of our teams:

- Teams owning a feature from start to finish outperform teams with partial ownership of features both in velocity and estimate accuracy.
- Highly volatile teams require additional domain education to grasp context. Once implemented, volatility decreases significantly.



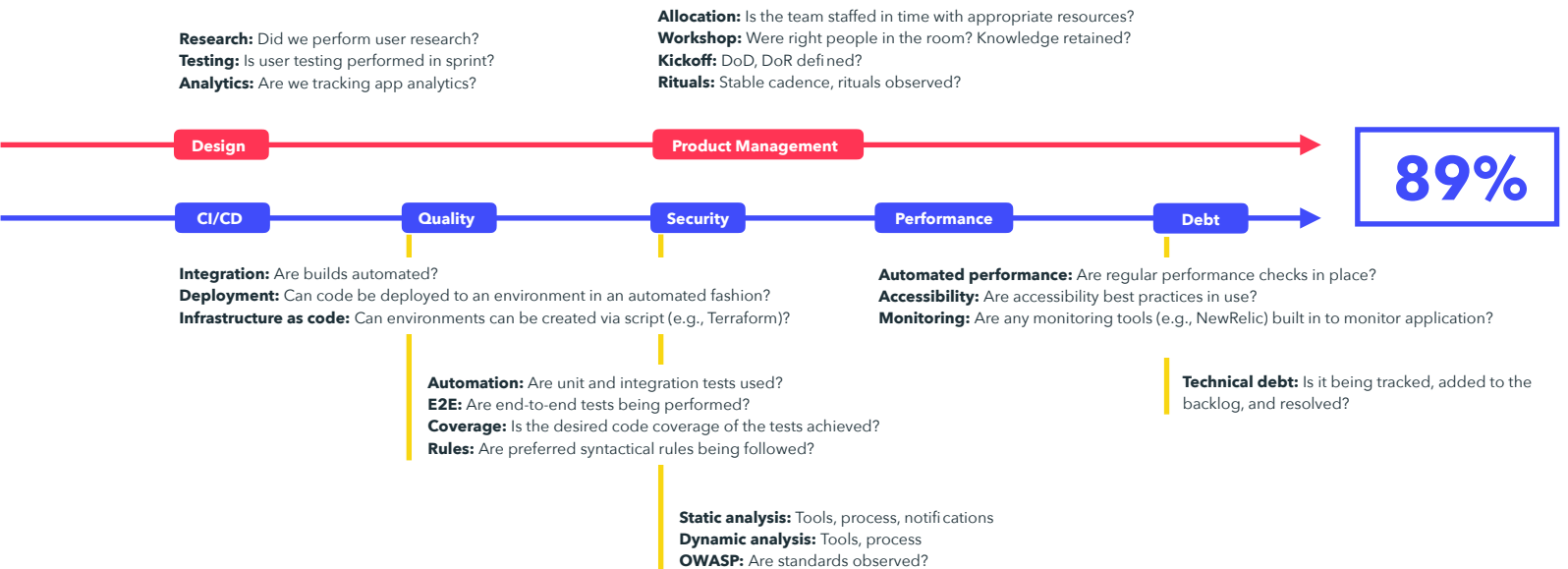
PROCESS METRICS: DETERMINE PRODUCT & TECHNICAL MATURITY

The last set of metrics measures the maturity of practices in a product organization. Fortune enterprises don't transform into product-centric organizations overnight. Even new greenfield product ventures make compromises to ship to market faster. As an organization converts the portfolio, leverage product and engineering best practices, which include tracking quantitative data on a per-product basis and documenting progress over time. Understanding where an application and corresponding teams sit on the maturity scale helps product leaders identify elements needing attention and products carrying the highest risk.

Large enterprise programs and small greenfield builds benefit from a maturity score. Technical and product debt accumulates even when taking precautions.

The maturity score measures percentages across two domains:

- 1 **Product management and design**
- 2 **Engineering**



THE SPECIFIC WEIGHTING ASSIGNED TO A CONDITION IS OUTSIDE THE SCOPE OF THIS PAPER. THESE ATTRIBUTES TEND TO BE UNIQUE TO A GIVEN ORGANIZATION AND ITS PRODUCTS.

PRODUCT MANAGEMENT & DESIGN MATURITY CONDITIONS

Product managers and designers are two key roles in product teams. Subsequently, evaluate the efficacy of various product management and design activities. Measure adoption and execution best practices (e.g., product canvas, user research, workshop participation, stakeholder commitment to rituals, and healthy team allocation). Track team allocation, activities during the workshop, kickoff of the project, and delivery rituals.

	CATEGORY	CONDITION	WHY IT MATTERS
ALLOCATION OF RESOURCES		Allocate a full-time product manager (or several)	The team manages scope and feature decisions without external dependencies.
		Allocate a full-time designer (or several)	Make sure design activities are part of the sprint; design needs to be consulted in the requirement definition and the final UI QA.
		Allocate enough engineering and testing members	Understaffed teams are bound to fail; ensure ample engineering resources to accomplish predefined goals.
WORKSHOP EFFECTIVENESS		Administer at least one product workshop within the last six months	Workshops guarantee alignment and clear outcomes for the next six months of releases. Going longer than six months will lead to churn.
		Ensure the cross-functional team members attend the workshop	Knowledge transfer is highly effective for cross-functional team members. Product managers, engineers, and designers all need to share information with another when developing products.
KICKOFF EFFECTIVENESS		Share the Definition of Ready and Definition of Done with team	The DoR and DoD are critical for effective cooperation inside a cross-functional team. Lock in expectations between a product owner, the engineers, and designers.
		Make sure the backlog is fully estimated	Story point estimates help the team establish initial milestones and assign risk to stories with high complexity.
		Get access to end user design research	Software should never be built in a vacuum. User research, user testing, and demos need to have a real audience.

Groom two sprints of the backlog in advance

Grooming a couple of sprints ahead reduces noise for the team, provides the ability to quickly swap in groomed stories when another piece of work is blocked or finished early. Avoid idling of engineering team members at all cost.

Follow sprint rituals (e.g., stand ups and retros)

Observing rituals ensures ongoing team communication and supports continuous improvement through collaboration.

Require stakeholders involved in decision-making to attend sprint demos

Demos give stakeholders an opportunity to accept work as well as make the software real. The build-measure-learn loop doesn't work without them.

Provide normal sprint reports (e.g., spend and burndown)

Use sprint reports, many of which are covered in this paper, actively as a tool.

Document change requests in Jira (or the internal tool of your choice) to communicate the impact of adjustments to the business

While change requests are not part of the agile process, tracking pivots in the product strategy helps product leaders explain decisions made to sponsors, some of which work outside of the product organization.

ENGINEERING MATURITY CONDITIONS

Like product managers and designers, engineers play a key role in building products. There are standard engineering competencies and best practices that each software project should observe to some degree to inform behaviors—good or bad—for the future of the product and organization. Measure the adoption of technical best practices like CI/CD, code quality, security, performance, monitoring, and technical debt). Track DevOps, testing, technical maturity, coding standards, security, accessibility, and monitoring.



Data strategy

Determine if needs are transactional or analytical. The use cases for machine learning and migrating away from ETLs.



Testing strategy

Proper testing strategy guarantees adequate coverage with sustainable investment, lower lifetime support, and QA costs.



Security

Design products to be secure instead of retrofitting them. Retrofitting is taxing and prone to issues. Ensure OWASP best practices are observed by team.



Code quality

Code quality standards, static code analysis, source control strategy, and code reviews guarantee a maintainable code base.



Scalability

Scalable-by-design applications scale easily and quickly without developer involvement. Performance targets are tested and monitored.



Performance

Establish requirements for application load speed, response to users, as well as automation of ongoing testing.



Monitoring

Active monitoring for established performance targets, application logs, and errors allows the team to resolve proactively.



DevOp

Using infrastructure as code enables rapid delivery and efficiency through self-provisioning.

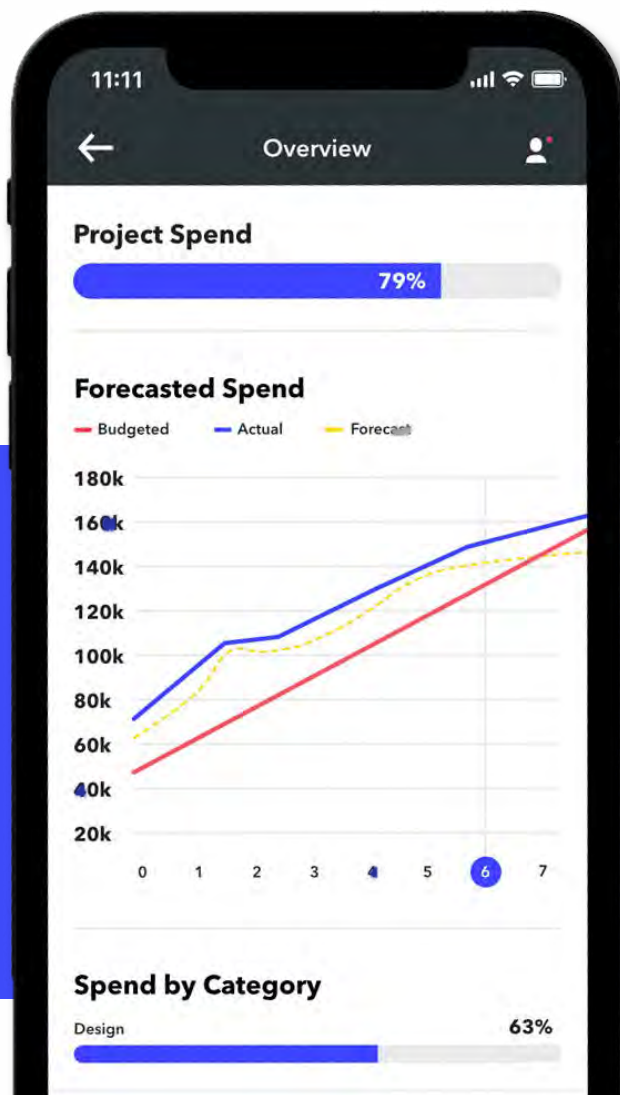
	CATEGORY	CONDITION	WHY IT MATTERS
DEVOPS		Feature continuous integration (CI)	CI allows automated integration of multiple workstreams into a single main branch, multiple times a day. It reduces overhead and risk, increases confidence, and improves communication.
		Automate deployments	Building and deploying code in the dev, staging, UAT, and production environments happens with a push of a button, eliminating room for user error.
		Use containerization	Containers prime the application for scale with each having lateral scalability without unnecessary overhead.
		Include continuous deployment (CD)	While challenging to attain, CD pushes code from development into production multiple times a day (assuming all tests run successfully) and represents the target state for mature product companies.
TESTING		Run unit/integration tests and coverage metrics	An advanced testing strategy automates the mundane and drastically reduces the long-term costs of quality assurance. Furthermore, less technical debt is accrued and software is easier to maintain.
		Administer performance testing	Performance goals and tests help teams avoid production challenges by anticipating future needs.
		Add end-to-end testing	End-to-end tests guarantee that the software and integrations work correctly.
TECH MATURITY		Track technical debt	Tracking technical debt mitigates the risk of blindsiding a product team with productivity dropping over time as the debt mounts.
		Evaluate and review the architecture	As an application matures, the underlying architecture needs to be evaluated and areas for refactoring need to be identified to avoid reaching a legacy state.

CATEGORY	CONDITION	WHY IT MATTERS
CODING STANDARDS	Provide code style check automation	When the code style is checked automatically and conforms to the rules defined by a project team, maintainability improves in the long run.
	Use static code analysis	Tools automatically scan code for vulnerabilities and security best practices, helping automate and avoid security issues.
	Offer feature branches	Use pull requests and feature branching as part of the engineering strategy. The code from the master branch should always be ready to be released to production.
	Make sure an artifact repository is used	Add an artifact repository (e.g., Sonatype Nexus) to store intermediate and final results of the software engineering process (e.g., npm packages, OSS project forks, docker images, release packages), which simplifies management and propagation of reusable components.
SECURITY	Complete a vulnerability and security assessment	Establish and observe security standards (e.g., OWASP 10) for the application.
	Provide a secret management option	Provide a central secret management platform such as HashiCorp Vault to avoid secrets being stored openly in code.
ACCESSIBILITY	Follow compliance standards	Establish and observe a level of application accessibility.
MONITORING	Monitor logging and reporting	Monitoring solutions provide insights and alerts for irregular patterns in production environment.

BUILD POWERFUL PRODUCTS BACKED BY METRICS

Enterprises need to ditch traditional project metrics and embrace modern metrics in order to build healthy products and deliver results. Evaluate product value effectiveness, quality, team velocity, and process maturity to determine portfolio health. These four categories of metrics form a system of indicators and guardrails that enable product teams to self-govern, product owners to deliver results, and product leaders to drive systemic organizational change to the desired state.

When assembling the final set of metrics, consider the objectives set out for the team, the product, and the organization. The maturity of the product and the organization inform which metrics are beneficial at which time. Use metrics as tools to identify challenges in delivery, evolve delivery process, and plan future releases. Under no circumstances should these metrics be used to punish poorly performing teams as that will incentivize other teams to game the system and post fake data. Approach challenges with the mindset of using them to remove blockers, educate, and enable professionals to succeed instead of looking for blame to be assigned to an underperforming party.



Checkout Devbridge's [PowerUp](#) which tracks delivery metrics for an individual team as well as a portfolio of projects. The application aggregates data from Jira and time-tracking software to accurately and transparently represent the current project state, team activity, burnup, burndown, velocity, and a variety of other helpful metrics.



AUTHOR

Aurimas Adomavicius

President and co-founder of Devbridge

Founded in 2008, Devbridge revitalizes the largest of enterprises with custom software. When not in the trenches working with clients, Aurimas is an active speaker and writer on product design and engineering best practices.

GET STARTED WITH A LEAN REQUIREMENTS WORKSHOP

Put custom software development on the fast track. First, get key stakeholders together. Then, let our product development team orchestrate the ultimate workshop. Without fail, we get the stakeholders aligned in a couple of days—which is more than most companies can do in months.

One rule: Don't bring documentation.

SIX KEY OUTCOMES

- 1 Defined business goal**
A clearly defined business goal with success metrics
- 2 MVP requirements**
The minimum amount of requirements necessary to kickoff the design process
- 3 Hidden requirements**
Bringing people from different functions together to uncover what impacts goals, scope, and priorities
- 4 Shared understanding**
A shared understanding of the business process, end users, and their pain points
- 5 Scope & priorities**
An agreement on scope and priorities to meet the business goals
- 6 Product release strategy**
A phased approach to releasing your product to market



Our experienced teams deliver software **4x faster** than the industry average.

We are a full stack research, design, and software delivery company. Our teams facilitate a creative, iterative process, powered by some of the most passionate individuals in the industry. We build extraordinary custom applications that solve complex problems and deliver measurable results.

Our scalable cross-functional product teams and tailored services transform businesses in agriculture, aviation, financial services, fraud & forensics, healthcare, hospitality, logistics, and manufacturing.



Mobile applications for B2B and B2C

We create elegant, intuitive mobile applications to enhance the customer experience.



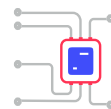
Product research and design

We build custom applications that solve real business problems to drive measurable value.



Scaling product delivery

We go beyond team aug and dedicate full cross-functional teams to scale feature delivery.



Data strategy and microservices

From dashboards to master data strategy, we can help you leverage your data to grow your business.



Legacy software modernization

Replatform aging systems and legacy architecture to meet the needs of your business now.



Workflow optimization

Streamline your business processes to reduce time, costs, and improve productivity.

500

Full-time employees

5

Offices

12

Years in business



Chicago

London

Toronto

Kaunas

Vilnius