Whitepaper

Behavioral Code Analysis in Practice

This whitepaper provides a detailed view of how CodeScene supports an organization with clear feedback loops and actionable insights into the evolution of a software system. The paper explains the purpose and value behind those analyses and how CodeScene complements existing workflows and processes.

Target audience

- Technical leaders
- Software architects
- Development teams
- Managers

About CodeScene

CodeScene was born in 2015 when founder Adam Tornhill published the book "Your Code as a Crime Scene". It introduced a new approach to software analysis which focused on the evolution of a codebase over time.

CodeScene has evolved at a rapid pace to become the next generation of code analysis and is used by global Fortune 100 companies in a wide variety of domains.

CodeScene

Content

1. Introduction

A multi-purpose visualization tool

CodeScene is a reaction and a complement to traditional static code analysis. The main difference between CodeScene's behavioral code analysis and traditional code scanning techniques is that static analysis works on a snapshot of the codebase at a single moment in time. CodeScene considers how the system has changed over time and translates the results to relevant, actionable information directly into business value. CodeScene inspects more than how code changes by considering the organization and the people side of the system in analyzing the system's evolution. The output is a set of valuable information invisible in the source code itself, such as measures of team efficiency, onboarding costs and off-boarding risks.

Serve different stages in software delivery cycle

A behavioral code analysis adds value by introducing feedback loops in all software delivery stages.

Serve multiple roles in your organization

Software delivery is a cross-functional team effort, and CodeScene reveals insights relevant for the various roles involved in your delivery process. This whitepaper introduces the different analyses useful for product managers and product owners, technical leaders, software architects and software delivery managers, software developers, quality assurance and DevOps engineers.

CodeScene main goals and objectives

Management / Product owners

- Bridge the gap between Tech and Business. CodeScene visualizes something as deeply technical as code for non-technical stakeholders. This allows management teams to see the development costs in the context of the system as well as the overall delivery performance.
- Monitor all your products at the inter-project dashboard. The key metrics on the dashboard highlight the progress on the goals, and point you to the products/ codebases/projects in need of attention and actions.
- Perform key personnel analyses to ensure you have an adequate knowledge distribution in the critical parts of the codebase.
- Evaluate organizational efficiency by measuring the operational boundaries of each team and detect modules that become coordination bottlenecks.
- Use CodeScene for project planning with on- and off-boarding simulations that helps you detect and act on potential off-boarding risks before they happen.

Technical Leaders / Software Architects / Management

- Prioritize technical debt based on the expected return on investment (ROI) if the debt is paid-off.
- Supervise planned goals with CI/CD Quality Gates based on CodeScene's Intelligent Notes where you specify your decisions and plans on the identified technical debt.
- Quantify the cost of technical debt and code quality issues through the integration with project management tools.
- Manage the identified technical debt by recording decisions, context, and plans directly in the tool that will then supervise the progress.

Development team

- Prioritize code reviews by integrating CodeScene in your CI/CD pipeline CodeScene classifies the risk of each pull request and/or commit, and delivers early warnings for changes that degrade code quality.
- Support on-boarding and learning through CodeScene's interactive visualizations. Find the weak spots in the codebase, identify the most relevant parts, and see how they fit together and which of your peers to communicate with.
- Guide retrospectives with data from how the team has interacted with the code during a sprint/iteration.
- Use the X-Ray analysis to prioritize refactorings on a function level.
- Evaluate existing designs through CodeScene's unique change coupling analyses that identify hidden and implicit dependencies that are invisible in the code itself.

QA / Testers / DevOps organizations

- Evaluate test automation efficiency by supervising the code quality in your automated tests.
- Predict and detect delivery risks through CodeScene's branch analyses and automated risk predictions
- Guide exploratory testing through CodeScene's interactive hotspot maps that show where most development activity has been over a particular period of time.

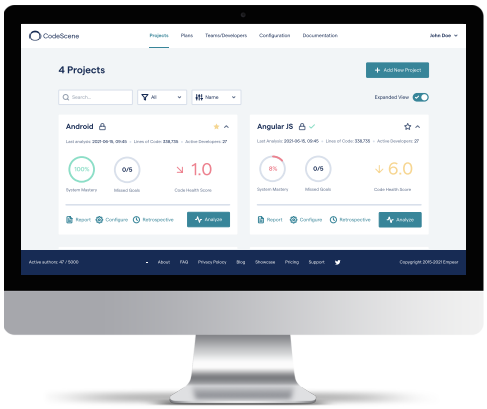
2. Users and Use Cases

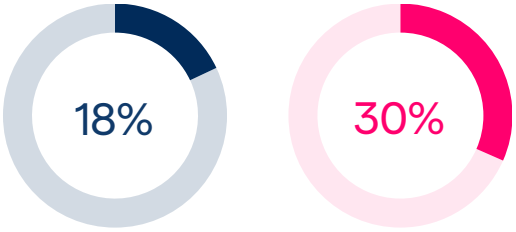
Management and Product owners

Bridge the gap between Tech and Business

Getting situational awareness in a software project is a hard problem. Even with rapid iterations and quick feedback, we're focusing mostly on surface behavior or functional correctness. And when we have code-level metrics, they tend to measure technical details that aren't easily communicable to non-technical stakeholders.

With CodeScene, management teams and stakeholders get the ability to see development costs in the context of the system.Through a customized dashboard you can follow graphs and trends. Understanding your delivery performance and hidden risks today and what direction your business needs to go tomorrow.





18% of code written by 1 developer

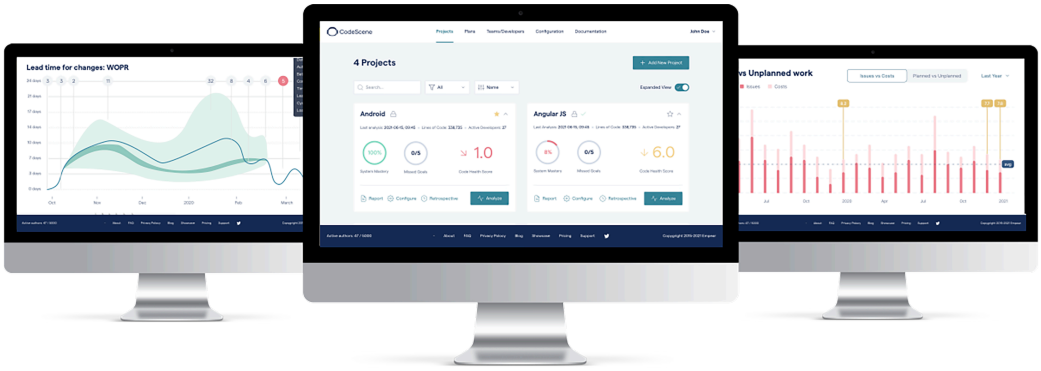
30% total abonded code

Key Personnel Analysis and Team Planning Simulations

CodeScene combines social measures with technical metrics to detect organizational risks in case one or more developers leave the organization. With the simulation module, you can detect any high-risk areas where the off-boarding results in a loss of mastery, which in turn might lead to technical issues like defects and financial risks like delays and missed deadlines.

CodeScene auto-detects key personnel risks. That is, parts of the system that are developed mainly by one or two individuals. CodeScene visualizes exactly which parts of the system that are as risk. CodeScene also includes an off-boarding simulator that shows the potential impact of an off-boarding. Combined with CodeScene's technical analysis, the tool automatically identifies high risk areas that lets you focus on-boarding efforts to where they are needed the most.

Technical leaders, Software Architects and Management



CodeScene Analyses

- Hotspots
- Code Health
- Trends
- Architectural Change Coupling
- Defect Distribution

Recommended Usage Frequency

- Initial in-depth analysis to assess the current state of the code and plan mitigations as needed.
- Weekly follow-ups of critical findings with measurable effects of the mitigations and actions.
- Weekly or monthly monitoring and supervision on the state and progress of the system.

Prioritize Technical Debt

Software systems often contain several instances of problematic code that are hard to understand, brittle, and hence potentially expensive to maintain. In a large system, there might be thousands of lines of such problematic code. As an organization we cannot -- and should not – address all of those potential quality problems at once; the task is simply too big, the risk and cost too high. So, we need to balance the trade-off between improving existing code versus adding new features for end users.

CodeScene resolves this by prioritizing the parts of the codebase that will bring the biggest and quickest benefits to your organization. The analysis is called prioritized hotspots. A prioritized hotspot is complicated code that the organization has to work with often. That is, any technical debt in those parts of the code has a high interest rate.

Once CodeScene has detected a hotspot, the tool provides an automated code review that classifies each hotspot according to severity and code health. That classification is available at a glance for both sub-systems and individual hotspots.

Why You don't have to Pay Off all Technical Debt

The hotspot analysis is based on evolutionary patterns and trends, and CodeScene usually identifies about 2-4% of a codebase as prioritized hotspots. The priorities build on a pattern that occurs in any codebase, independent of programming language, domain, or technology. By prioritizing technical debt at the head of the curve, CodeScene ensures that the suggested improvements give you a real return on investment should you decide to pay off any potential technical debt in that part of the code.

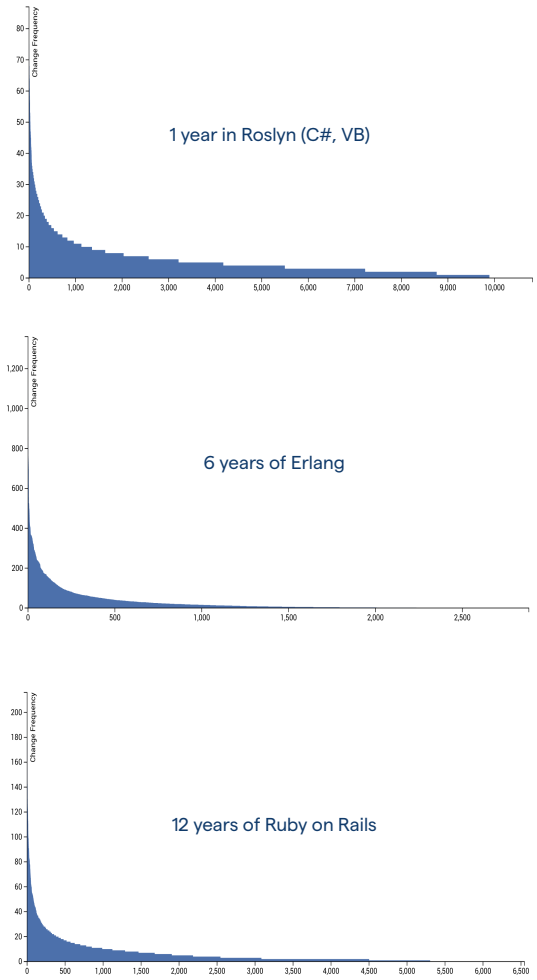


Figure 1. Examples of change frequency of open source codebases

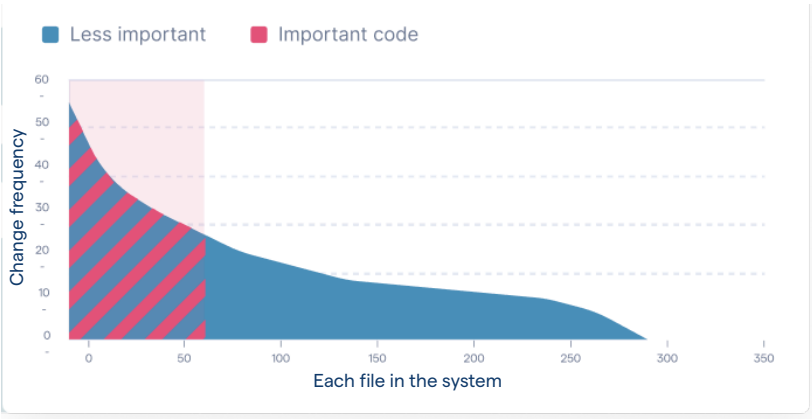


Figure 2. Change frequency of source code files

In the preceding figure 2, the X-axis shows each file in the system sorted on its change frequencies. The Y-axis shows the number of changes done to each file over time. These power law distributions show that:

- Most development activity is located in a small part of the total codebase.
- The majority of all files are in the long tail, which means they represent code that's rarely, if ever, touched.

Quantify the Cost of Technical Debt and Code Quality Issues

CodeScene combines its technical analyses with data from project management software, like JIRA, Trello, GitHub Issues, or Azure DevOps. CodeScene's cost analyses let you reason about the technical and organizational findings from a financial perspective. For example, how much time do you spend on defects in your top hotspots? What amount of work is unplanned? And what happens over time?

This information is used to ensure that the code evolves in the right direction. In general, the ideal situation is to spend less time on bug fixes and increase the portion of time spent on new features and improvements. Using the cost trends, it is now possible to measure this, which also lets you evaluate the effect of technical and organizational improvements resulting from the other analyses,

Figure 3. CodeScene shows the relationship between code health and issues and costs from your issue tracking system

Manage the identified Technical Debt by Specifying Goals

There's always a trade-off between improving existing code and adding new features. Improvements cost time and money, and when we refactor or even redesign a piece of code, we are placing a bet that our investment will pay off in the future. As such, larger and more significant improvements have to be balanced against the short-term goals of the product. To address those forces, CodeScene lets you plan goals for each hotspot. Your goals are then automatically supervised and act as quality gates in CI/CD.

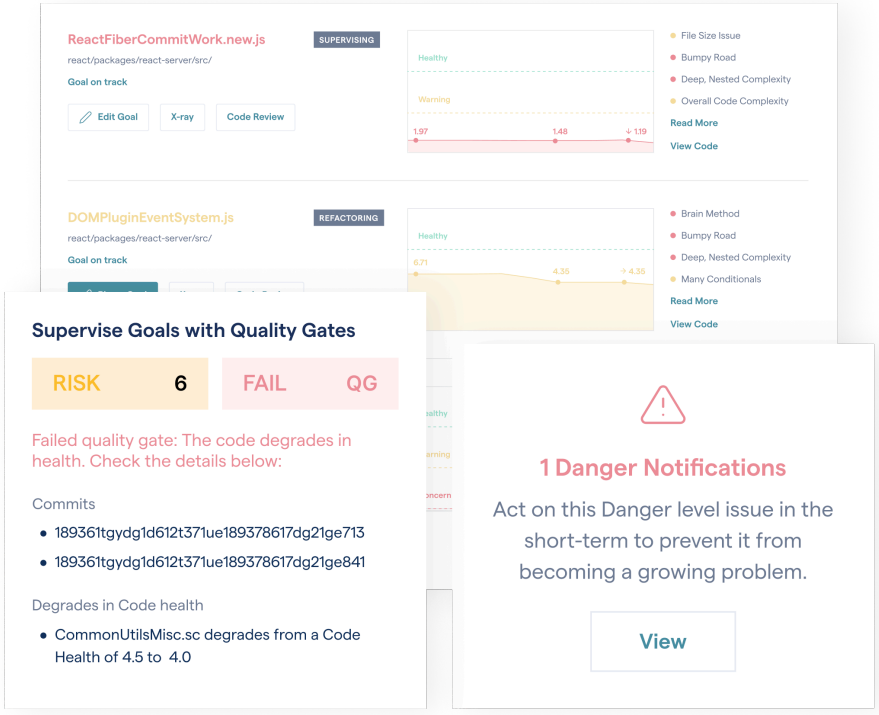


Figure 4. CodeScene provides alerts for any code changes that violate a goal.

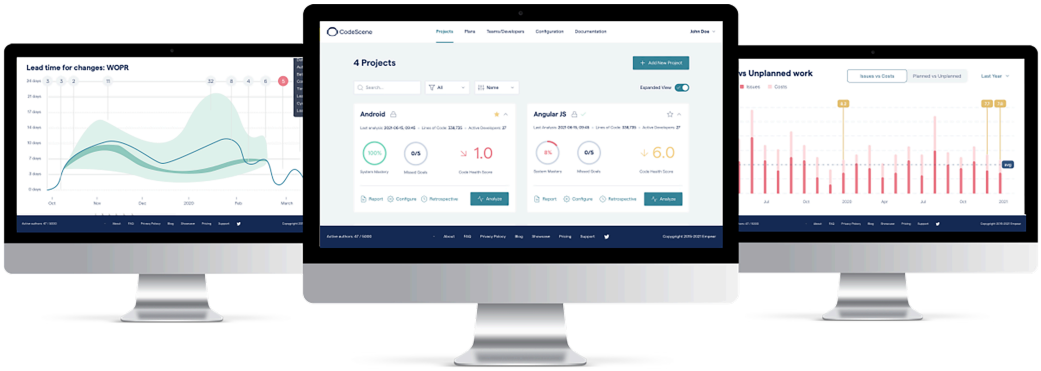
Quantify the Cost of Technical Debt and Code Quality Issues

The earlier you can react to any potential problem or surprise, the better. That's why CodeScene offers integration points that let you incorporate the analysis results into your build pipeline. That way, CodeScene can supervise your hotspots to ensure that all goals are on track and that the code health is maintained. In addition, CodeScene comes with an open API that lets you integrate the analyses into any delivery pipeline.

CodeScene integrates with the following systems:

- GitHub
- GitLab
- BitBucket
- Azure DevOps
- Jenkins

Organizational Analyses



Behavioral Analyses

- Conway's Law
- Inter-Team Coordination Needs
- Operational Team Boundaries
- Knowledge Distribution

Recommended Usage Frequency

- Perform an in-depth analysis as input to planned organizational changes such as new team structures or significant architectural changes.
- Supervise the potential coordination needs between feature teams on a per sprint basis.
- Off-Boarding simulation used as developers leave the organization or are transferred to other product lines or projects.

Evaluate Organizational Efficiency

CodeScene measures the operational boundaries of each team and detect modules that become coordination bottlenecks. This is important because social aspects like coordination, communication, and motivation issues increase in importance with the size of an organization.

Unfortunately, these aspects of software development are invisible in the code itself; if you pick up a piece of code from your system there's no way of telling if it's been written by a single developer or if that code is a coordination bottleneck for several development teams. Hence, CodeScene's behavioral code analysis helps you fill in the blanks.

The organizational and social side of code has historically been left largely to subjective judgments. Using behavioral code analysis, we can start to guide those decisions with objective data instead and measure aspects like Conway's Law.

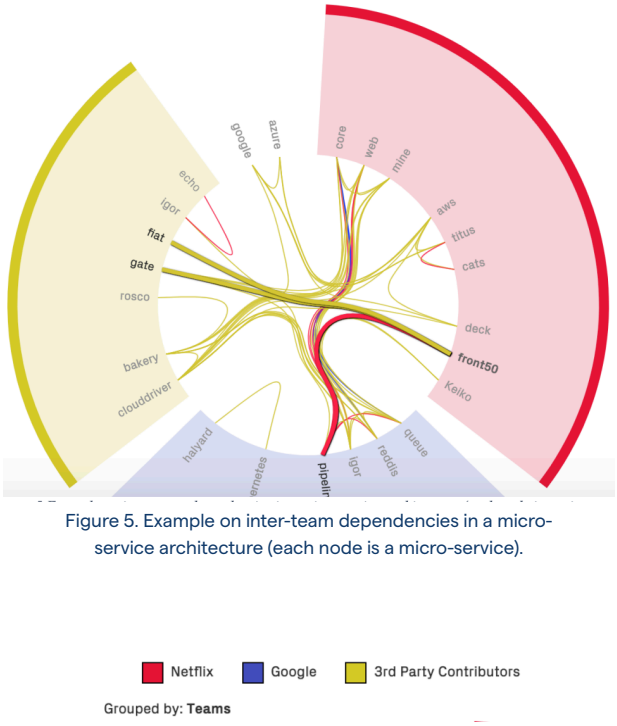
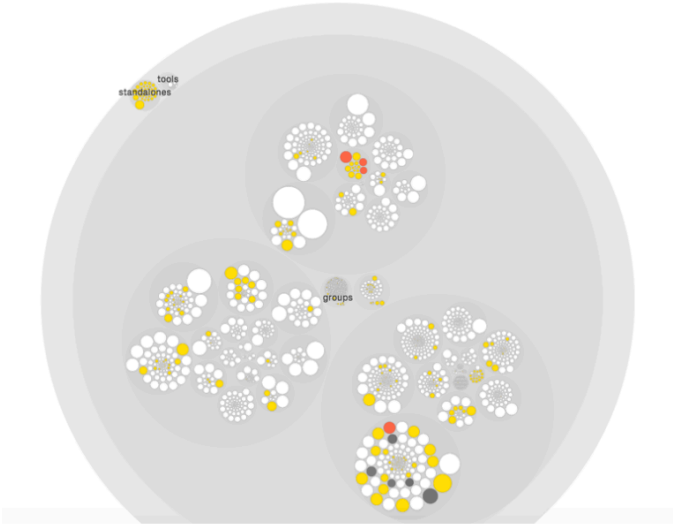


Figure 5. Example on inter-team dependencies in a micro-service architecture (each node is a micro-service).

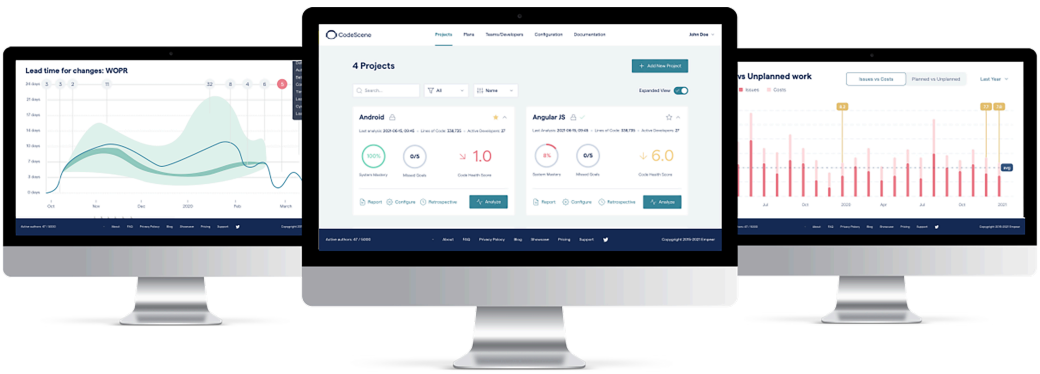


Perform key personnel analyses

Maintaining a high system mastery means ensuring you have an adequate knowledge distribution in the critical parts of the codebase. CodeScene measures and identifies potential risks in this area, and visualizes the knowledge distribution in an interactive map:

- Knowledge island in Complex Hotspot
- Knowledge island
- Risk: Complex Code by Former Contributors
- Multiple Active Developers

Development Team



Behavioral Analyses

- Hotspots
- Code Health
- Virtual Code Review
- X-Ray
- Change Coupling
- Risk Prediction through CI/CD integration.

Recommended Usage Frequency

- Indirect daily use through integration of CodeScene in the continuous integration pipeline.
- Sprint-based usage to support retrospectives and sprint planning
- Spontaneous as part of learning a new part of the codebase or as on-boarding support.
- Occasional usage (approximately weekly or monthly) to Investigate hotspots in depth when they are detected.
- Occasional usage (weekly or monthly) to follow-up refactoring effects with virtual code reviews.
- Weekly supervision of implicit dependencies at the architectural level.

Prioritize code reviews

Like all manual processes code reviews are hard to scale. As an organization grows, code reviewer fatigue becomes a real thing: there's just so many lines of code a developer can review each day. Beyond that point we're likely to slip. The result is increased lead times, bugs that pass undetected to production, and – in extreme cases – the risk for burnout. At CodeScene we have developed a system for automated risk classifications to prioritize the code in need of a review. The risk classification is exposed through an open REST API that lets you integrate the classification into your continuous integration pipeline. The following figure shows an example from a Jenkins build where a high-risk change is detected:

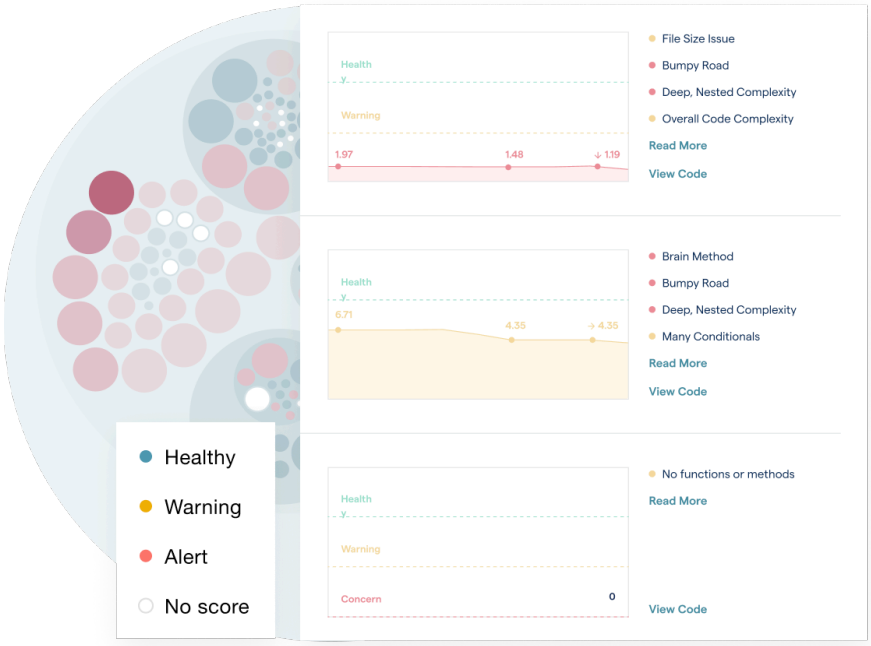


Figure 6. CodeScene automated code reviews

Guide Sprint Retrospectives

The traditional format of sprint retrospectives – like most group activities – makes us sensitive to a number of biases such as availability heuristics, pluralistic ignorance, social desirability, and confirmation bias. By informing decisions with data about how our system evolves, we make an important shift: we move away from speculations and reduce a number of social biases in the process.

A behavioral code analysis provides data that reduces such biases, and CodeScene comes with a special retrospective feature that runs an analysis on a team's development activity over the last sprint. That means you get data on how your development efforts, features and stories actually impacted your codebase.

Support On-boarding and learning

CodeScene's interactive hotspot maps show a holistic overview of the codebase in its socio-technical context. The map is a great starting point when trying to get insights into a new codebase. From the hotspot map, developers can focus on the core parts of the codebase, and the virtual code review provides detailed insights into the technical and social factors of a module or file.

Hence, when working in a new part of the codebase, use the patterns of the previous developers to uncover technical challenges and identify the primary developers behind each piece of code.

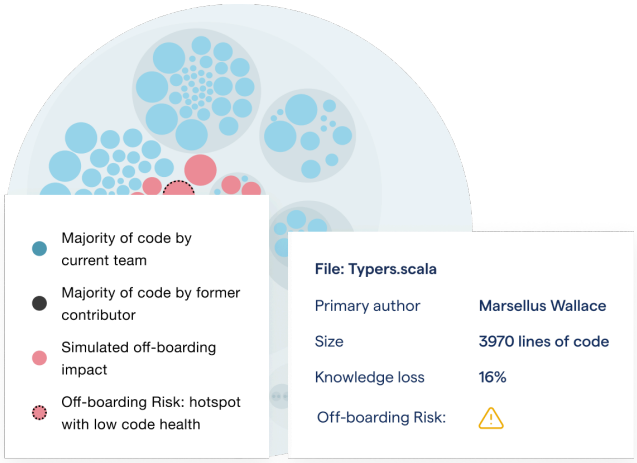


Figure 7. Identify primary developers behind each piece of code.

Use the X-Ray analysis to prioritize actionable refactorings

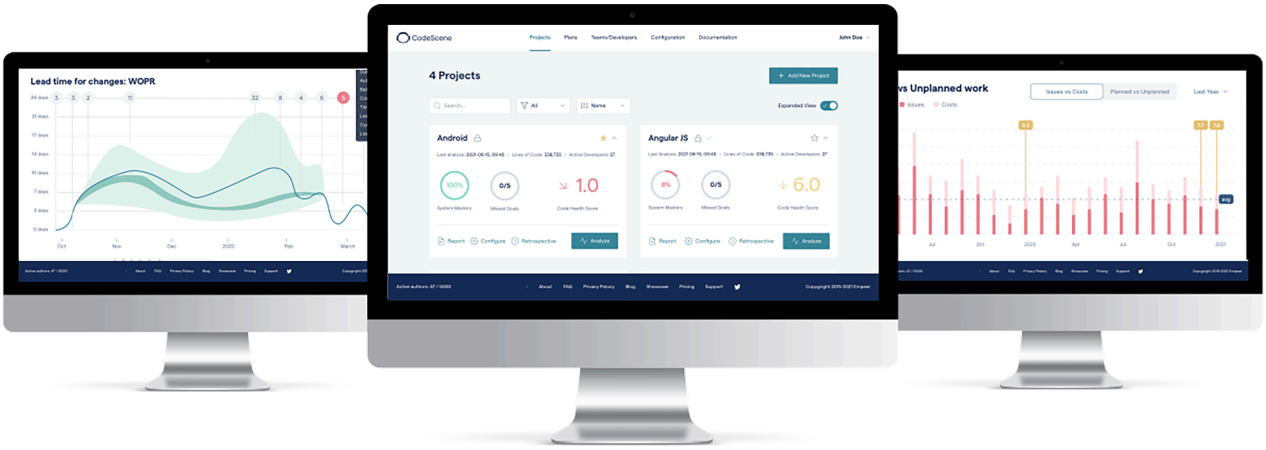
CodeScene's X-Ray is a programming language aware analysis tool that operates on the function/method level of your code. Thus, X-Ray is able to provide deep and detailed information on what's happening inside a large hotspot. The X-Ray analysis is used to prioritize actionable refactorings inside larger hotspots.

Hotspots | Internal Change Coupling | External Change Coupling | External Change Coupling Details

Function	Change Frequency	Lines of Code	Cyclomatic Complexity	Defect Commits	Overloaded Functions?
ZooKeeperServer View Complexity Trend | View Function Code	29	69	8	3	7
loadData View Complexity Trend | View Function Code	17	33	2	1	1
processPacket View Complexity Trend | View Function Code	16	84	9	4	1
processConnectRequest View Complexity Trend | View Function Code	15	102	17	2	1
startup View Complexity Trend | View Function Code	15	3	1	1	1
shutdown View Complexity Trend | View Function Code	13	59	12	1	2
processSasl View Complexity Trend | View Function Code	11	63	9	1	1

Figure 8. X-Ray runs a hotspot analysis on a function level to provide actionable refactoring targets.

QA, Testers and DevOps Organizations



CodeScene Analyses

- Branch Analyses
- Risk Predictions
- Defect Mining
- Hotspots
- Code Biomarkers

Recommended Usage Frequency

- Daily monitoring of upcoming delivery risks.
- Weekly supervision of hotspots in the test automation code and infrastructure.
- Weekly supervision of implicit dependencies on architectural level.
- Sprint-based priority of exploratory testing targets.

Predict and Detect Delivery Risks

Many organizations work on feature branches and employ practices like continuous integration/delivery. To work in practice, those feature branches have to be kept short-lived. By applying behavioral code analysis, we're able to visualize the branching activity, measure lead times, and even predict the delivery risk of individual branches. The most important information here is CodeScene's automatic prediction of delivery risk as shown in figure 9.

This risk classification predicts the risk for defects, and is given on the scale 1-10 where 10 is the highest risk. An organization uses this information to plan preventive measures such as extra code reviews and tests. In extreme cases, an organization may choose to postpone the merge of high-risk branches when close to a critical deadline. To provide visibility, CodeScene comes with an auto-updated dashboard that highlights delivery risks as they happen.

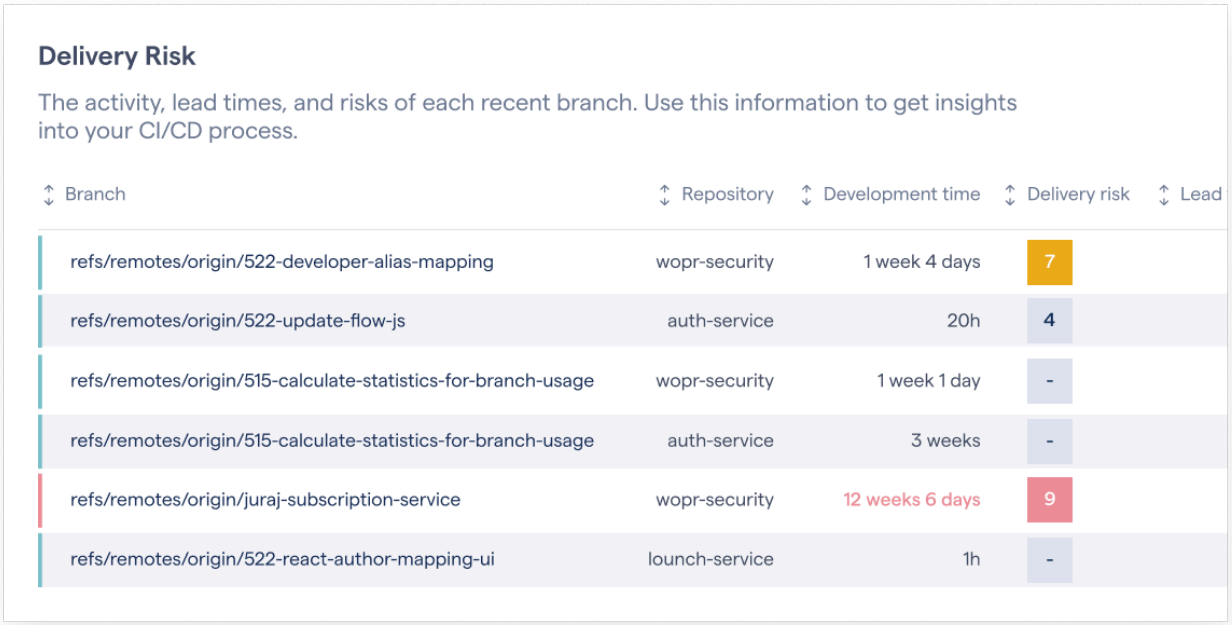


Figure 9. Automated prediction of delivery risks for ongoing work.

Evaluate Test Automation Efficiency

Technical debt isn't limited to application code, and frequently supporting code such as automated tests accumulate a high degree of technical debt. When that happens, the test code – intended to help an organization go faster – suddenly becomes a bottleneck. To avoid that situation, we recommend that you include all test code in the analyses as well. CodeScene specific detectors of code smells in its biomarkers analysis.

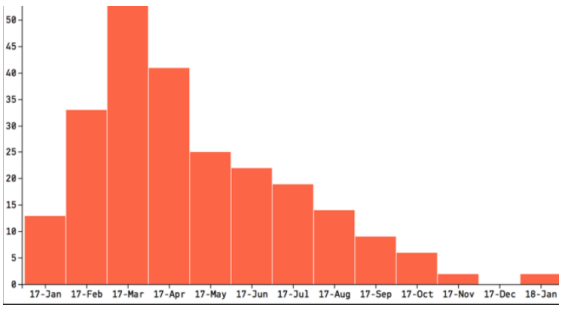
Further, an X-Ray analysis of a test hotspot might reveal opportunities to simplify the code. Specifically, we recommend the change coupling analysis combined with the code similarity view; That's CodeScene's copy-paste detector used on frequently co-changing tests:

Coupled Functions	Degree of Coupling (%)	Average Revisions	Similarity (%)
SetProperty_PropertyIsPreinitialized_DefaultValueAttributeDoesNothing SetProperty_PropertyIsPreinitialized_NoValue_DoesNothing	85	14	96
SetProperty_PropertyHasDefaultValue_DefaultValueAttributeDoesNothing SetProperty_PropertyIsPreinitialized_NoValue_DoesNothing	76	13	93
SetProperty_PropertyHasDefaultValue_DefaultValueAttributeDoesNothing SetProperty_PropertyIsPreinitialized_DefaultValueAttributeDoesNothing	92	13	90

Figure 10. An example from ASP.NET Core MVC where the X-Ray detects duplicated logic in the unit tests.

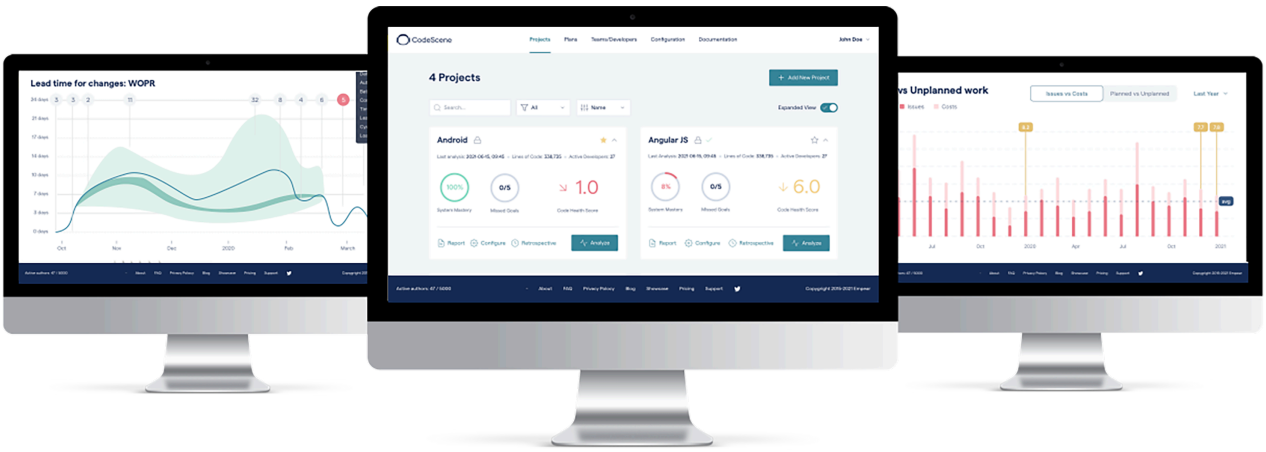
Guide Exploratory Testing, Prioritize Tests

Organizations that employ exploratory testing techniques use the hotspot maps to identify where the development activity has been over a particular period of time, e.g. the last sprint.



In addition, CodeScene also provides a Defect Density view. The defect density view shows how distributed the bug fixes are, which lets you identify defect-dense areas of the code and focus extra testing on those parts.

3. Get CodeScene as a SaaS, or host it in a Private Cloud or custom Data Center



CodeScene as SaaS

- CodeScene available as a SaaS product at https://codescene.io/

CodeScene as On-premise

- CodeScene is also available in an on-premise distribution that you can host in a private cloud or your own data center.
- The setup is easy since CodeScene is also distributed as a Docker container.

Contact

Contact: sales@codescene.com
Twitter: @codescene
LinkedIn: https://www.linkedin.com/company/codescene

Further Info, Blog And Articles

www.codescene.com
www.codescene.com/blog/