ChatGPT

CAST Highlight: Overview of Features

CAST Highlight is a SaaS-based portfolio analysis tool that examines source code across multiple dimensions. It uses static analysis "code insights" to quantify software health, security, open-source risks, and technical debt. Cast Highlight provides a **Software Health** score (0–100) averaging three factors – *Resiliency*, *Agility*, and *Elegance* – that measure coding best practices related to robustness, maintainability, and complexity [1] [2]. The tool also computes an **Open Source Safety** index (0–100) combining component *Security*, *License* and *Obsolescence* sub-scores [3]. Behind the scenes, Highlight's **Software Composition Analysis (SCA)** engine detects all open-source/third-party components (via a ~118M–component database) and cross-references known CVEs (~150K in NVD) [4]. For security, it tags any component CVE that appears in CISA's "Known Exploited Vulnerabilities" database [5], and can notify users of newly published vulnerabilities affecting their apps without re-scanning [6]. By default, Highlight does *not* perform deep SAST on proprietary code (unlike tools like Checkmarx); its security focus is on open-source components and coding patterns that impact reliability (e.g. "avoid generic catch" rules in **Software Resiliency**) [2]. In summary, CAST Highlight reports on:

- **Code Quality:** Through its *Software Health* score, it identifies code issues (unused methods, nested loops, poor naming, etc.) affecting maintainability and complexity [7] [8]. Each issue ("code insight") contributes to *Elegance*, *Agility*, or *Resiliency* sub-scores, providing high-level and drill-down views of code quality by file or technology [1] [2].
- **Open-Source Risk:** It scans for all libraries and frameworks, checking versions, CVEs, license types, and age. Its *Open Source Safety* score averages: (1) CVE-based security (number of CVEs per component, weighted by severity) [9]; (2) license risk (counts of low-/medium-/high-risk licenses); and (3) obsolescence (gap to latest version) [10]. It supports SBOM import (CycloneDX) to instantly analyze components, produce a BOM, and flag vulnerabilities or license issues [11]. License obligations are surfaced, and users can define custom license-compatibility rulebooks to catch conflicts (e.g. GPL vs Apache) in the generated BOM [12]. The product also offers portfolio-level advisors that prioritize apps by open-source risk and even "predictive" vulnerability analysis (detecting flawed open-source code not yet in CVE databases [13]).
- **Technical Debt:** CAST Highlight quantifies technical debt in developer-time units. Each code insight has an associated remediation effort (in minutes/hours), and Highlight sums *occurrences* × *effort* across all insights to give total debt for an app [14]. The result is expressed in minutes (or aggregated to hours/person-days) in the UI [14]. This model replaced an older proprietary-dollar metric to be more transparent and customizable. Users can override the effort estimates per issue and re-calculate portfolio debt. Dashboards show debt by factor (Resiliency vs Agility vs Elegance) and help pinpoint the files or findings driving most of the debt. A "Portfolio Advisor for Technical Debt" visualizes debt layers (e.g. by health factor, priority, technology, code insight) and highlights the top 5 most indebted apps [15].

CAST Highlight surfaces all of these metrics in a unified UI, letting managers track health and risk across a portfolio. It **does not** natively check for secrets or configuration issues, nor does it scan running applications or inject code (unlike DAST or IAST tools). Instead, it offers a high-level "software intelligence" dashboard: scores, charts, and customizable reports of *code quality*, *security/vulnerability posture* (mostly via open source CVEs), *license compliance*, and *technical debt*. Key references to CAST's features include its documentation and tutorials [14] [3] [2] [12].

CAST Highlight Capabilities by Domain

Aspect	CAST Highlight Approach	Reporting & Output
Code Security	Scans open-source/3rd-party libraries for known CVEs (via NVD + CISA KEV) [5]. Uses an SCA DB (118M components) for detection [4]. Provides alerts for newly published CVEs impacting an app [6]. Highlights insecure code patterns under *Software Resiliency*.	Open Source Safety score (inc. "Security" sub-score) [16], lists of vulnerable components and CVEs by criticality, KEV-tagged CVEs, and notifications of new issues [5] [6].
Code Quality	Uses static code analysis ("code insights") to detect bad practices and complexity issues in source code. Calculates *Software Health* = average(Resiliency, Agility, Elegance) [1]. Hundreds of technology-specific rules cover dead code, nesting, naming, documentation, duplication, etc. [7] [8].	Software Health score (0–100) and its three sub-scores [1]. Drill-down reports show top "code insights" (problematic code constructs) by file or module, and trend charts.
Open-Source Risk Mgmt.	Detects all open-source components (via code scan or imported SBOM) and gathers metadata: version, license(s), age, vulnerabilities. Computes *Open Source Safety* = avg(Security, License, Obsolescence) [3]. Built-in license database (with customizable risk profiles) flags high-risk licenses and license conflicts [12]. Indicates outdated or abandoned libraries and suggests safer versions.	Open Source Safety score (0–100) with sub-scores for CVE exposure, license compliance, and obsolescence [10]. Tables of components with vulnerabilities, license risk, and upgrade advice. BOM exports (Excel/Word) include dependency trees, license compatibility warnings, and change-logs of new issues.
Technical Debt Estimation	Sum of fix efforts for all code issues. Each code insight has a default remediation time (minutes/hours); total debt = Σ(occurrences × effort) [14]. Effort defaults can be customized portfolio-wide, then debt is recalculated. Technical debt is reported in time units (days) per app.	A Technical Debt indicator (in person-days) per application and portfolio, with breakdown by contributing factor (Resiliency, Agility, Elegance). Portfolio Advisor visualizations show debt by factor, by rule priority, by technology, etc. [15]. Detailed "Top 5" apps by segment. Time and cost reports.

Comparing to Other Tools

The following summarizes how Checkmarx, Black Duck, TruffleHog, and SonarQube relate to CAST Highlight's domains:

Tool	Code Security	Code Quality	Open-Source Risk	Tech Debt	Primary Strengths	Integration
Checkmarx (CxSAST)	**Extensive SAST**: Performs deep static analysis on proprietary code to find security vulnerabilities (OWASP, CWE, logic flaws) [17]. It also includes Open Source Analysis (OSA) to flag vulnerable libraries and license issues [18].	*Weak:* Focus is security; does not report code quality scores or design issues.	*Partial:* Via OSA, it can identify open source components' CVEs and license compliance violations [18], but lacks the portfolio-level context and remediation suggestions of Highlight.	*No:* Checkmarx does not estimate "tech debt" in the Fowler sense; it tracks vulnerabilities per scan but not aggregated refactoring effort.	Very thorough vulnerability detection in custom code, low false positives (claimed 80% reduction), and IDE/ DevSecOps integration. Ideal for security gates.	Integrates with build systems (Maven/ Ant), SCM (GitHub, Azure DevOps), C (Jenkins, Bamboo), issue trackers (JIRA), and quality tool (SonarQub [19]. Offers IDE plugins for on-the-fly scanning

Tool	Code Security	Code Quality	Open-Source Risk	Tech Debt	Primary Strengths	Integration
Black Duck (SCA)	*Minimal:* Only for open-source components. Detects known vulnerable OSS libraries across code and binaries; no SAST on proprietary code.	*No:* Not designed for code smells or static analysis of original code. (Not an SAST tool.)	**Comprehensive SCA:** Industry-leading open-source management. Identifies declared/ undeclared dependencies (code, manifests, binaries) via multiple engines [20]. Reports on CVEs (security), license obligations, age, and even "code snippet" reuse [20]. Maintains a vast KnowledgeBase (2750+ licenses) [21]. Provides SBOM tools and policy enforcement across CI/CD [22] [23].	*No:* No concept of "technical debt."	Deep license & vulnerability coverage for OSS. It can pinpoint transitive and hidden deps, match code snippets to origins, and continuously monitor deployed apps [24] [25]. Useful for supply-chain security and IP compliance.	Integrates into SDLC with plugin (CI, IDE, package managers) and APIs. Feeds vulnerabilit alerts into issue trackers or policy gate

Tool	Code Security	Code Quality	Open-Source Risk	Tech Debt	Primary Strengths	Integration
TruffleHog	**Focused Security:** Scans code repositories (and other data stores) for high-entropy strings and known secret patterns [26] [27]. Detects leaked API keys, tokens, credentials in commit history, containers, etc.	*No:* Not a code quality tool.	*No:* Does not inspect libraries or licensing.	*No:* Does not measure code refactoring effort.	Very effective at finding secrets/ sprawling credentials anywhere in an SCM or artifact. Supports detection and verification pipelines.	Easy to run via CLI or a a pre-commit hook. Can hook into C to block commits with detected secrets.

Tool	Code Security	Code Quality	Open-Source Risk	Tech Debt	Primary Strengths	Integration
SonarQube	**Some Security:** SonarQube includes security rules (detecting certain vulnerabilities, OWASP Top 10, CWE patterns) in all editions [28] [29]. In Enterprise+ editions (Advanced Security), it adds taint analysis SAST and now incorporates SCA for open-source via the new Tidelift acquisition [30].	**Core Focus:** Rich static analysis for code quality. Detects bugs, code smells, duplications, coverage gaps, style and complexity issues across 25+ languages. Gives a **"Maintainability" rating** and remediation time (technical debt) for issues.	**Partial (new):** Sonar's new Advanced Security (post-2025) bundles SAST + SCA: it can now identify vulnerable OSS dependencies and track license usage (enabling SBOM generation) [30]. However, in standard Community/ Developer editions, SCA is not built-in (apart from simple dependency checks).	**Yes:** Sonar computes *Technical Debt Ratio* based on remediation cost of issues vs development cost [31]. It reports "Technical Debt" (in developer hours) on projects and aggregates this on portfolios.	Mature code-quality insights and "quality gates" for CI. Sonar provides drill-down dashboards, historical trends, and developer-friendly issue navigation. It's highly extensible (plugins, rules) and offers a free community edition.	Integrates with virtually an CI/CD (Jenkins, Azure DevOps, GitLab, GitHub actions), and has ID plugins (Eclipse, VS Code, IntelliJ). Als integrates with other quality/ security tools. Supports export of metrics and has a rich REST API.

Summary: CAST Highlight's unique value is **portfolio-level software intelligence** combining multiple axes. **Checkmarx** would cover only the *security scanner* aspect of Highlight (but far more deeply in proprietary code). **Black Duck** covers nearly all open-source risk aspects (more comprehensively) but not code quality or debt. **TruffleHog** adds a narrow capability (secrets detection) that Highlight lacks. **SonarQube** overlaps heavily on code quality and technical debt (providing similar "code insights" and time-based debt), and with its new SAST+SCA it is moving into security and open-source risk domains (SBOM, CVE scanning) [30]. However, Sonar is typically applied per-project, whereas CAST Highlight is SaaS-oriented to scan many applications and compute aggregate scores. Each tool has its niche: use Checkmarx for in-depth vulnerability analysis, Black Duck for OSS compliance, TruffleHog for secret scanning, and SonarQube for developer-centric code quality. None of them by itself replicates Highlight's integrated portfolio dashboard, but together they cover most of its functions.

Building a Custom Toolchain

One could **orchestrate these tools** to emulate CAST Highlight. For example, set up a CI/CD pipeline (e.g. in Jenkins or GitLab) that, for each application repository, runs:

1. **SonarQube Scan**: Performs static analysis, computing quality metrics and technical debt. (Set up a Sonar Server and use SonarScanner; it outputs metrics and an API for debt values.)
2. **Checkmarx Scan**: Runs the Checkmarx SAST on the source, yielding a list of detected vulnerabilities. (Use the Cx CLI or CI plugin; results can be pushed to Checkmarx's DB or retrieved via API.)
3. **Black Duck Scan**: Analyzes the SBOM or source tree for OSS components. (Trigger Black Duck SCA via CLI or API; it returns a BOM and vulnerability/license report.)
4. **TruffleHog Scan**: Scans the repo for secrets. (Invoke TruffleHog CLI on the repo and record any findings.)

Each run would output structured results (JSON, XML, or database entries). A **data aggregation layer** would ingest these outputs into a centralized store. For instance:
- SonarQube's PostgreSQL database or its REST API can provide project health scores and "technical debt" (remediation time) by project.
- Checkmarx APIs yield lists of CVEs/flaws per project.
- Black Duck provides a consolidated list of open-source components with CVEs, license issues, and an SBOM via its API.
- TruffleHog can output a JSON of discovered secrets.

This data could be loaded into a reporting database or Elasticsearch index. A custom **reporting layer** (e.g. Kibana dashboards or a BI tool) would then correlate metrics across tools and projects. For example, one might:

- Map Sonar's debt estimates (minutes) into portfolio totals per app and technology.
- Combine Checkmarx and Black Duck CVE counts (weighted by severity) into an "Open-Source Safety" metric per app.
- Flag any app where TruffleHog found secrets.
- Apply license-risk heuristics (e.g. assign numeric scores to license categories) and compute a compliance index.

The reporting front-end could mimic Highlight's dashboards: a portfolio summary table (health scores, debt, risk indexes for each app), visual charts (e.g. debt breakdown by factor or security risk pie charts), and drill-down detail pages for each application. You would likely need to build (or use) a front-end framework to display combined data, or integrate with an existing platform (e.g. Jira or Confluence dashboards, or a custom web app).

Integration considerations: Each tool supports automation but in different ways. Continuous integration (Jenkins pipelines, GitHub actions) can trigger all scans. Authentication and results retrieval require API keys or service accounts. Data formats differ (Sonar's reports vs Checkmarx's XML vs Black Duck's JSON), so custom parsers or connectors are needed. Results should be normalized (e.g. mapping CVSS scores to a unified criticality scale). Scheduling scans for large portfolios requires resource planning (these analyses can be time-consuming). Finally, maintenance entails keeping the tools' rule sets and vulnerability databases up to date (e.g. syncing Checkmarx and Black Duck feeds).

In summary, building a “Highlight-like” toolchain involves: running Sonar, Checkmarx, Black Duck and TruffleHog in a CI pipeline; consolidating their outputs (Sonar metrics, vulnerability lists, license reports, secrets) into a common data store; and providing a unifying dashboard. You would also implement formulas (e.g. Weighted CVE counts for “Security score”, and debt-summation) to generate composite indicators. With enough integration effort, this custom system could approximate CAST Highlight’s functionality, though it would require stitching together multiple UIs or creating a bespoke interface to replicate the seamless portfolio view and automated recommendations that Highlight provides.

Sources: The descriptions above draw on CAST Highlight’s official documentation and feature blogs [14] [3] [2] [12], as well as product literature for Checkmarx [17] [19], Black Duck [32] [20], TruffleHog [26] [27], and SonarQube [28] [30] [29]. These sources detail each tool’s capabilities, strengths, and integration points. The custom-architecture approach is based on typical DevSecOps patterns and the tools’ public integration APIs.

---

[1] Indicators & Methodology - CAST Highlight
https://doc.casthighlight.com/indicators-methodology/

[2] Software Resiliency - CAST Highlight
https://doc.casthighlight.com/software-resiliency/

[3] [9] [10] [16] Open Source Safety - CAST Highlight
https://doc.casthighlight.com/open-source-safety/

[4] Dealing with outdated third-party components | Documentation
https://doc.castsoftware.com/explore-results/learn/third-party/

[5] Feature Focus: CISA’s Known Exploited Vulnerability Insights - CAST Highlight
https://doc.casthighlight.com/feature-focus-cisas-known-exploited-vulnerability-insights/

[6] [11] [13] Product Tutorials & Third-Party Tools - CAST Highlight
https://doc.casthighlight.com/

[7] Software Elegance - CAST Highlight
https://doc.casthighlight.com/software-elegance/

[8] Software Agility - CAST Highlight
https://doc.casthighlight.com/software-agility/

[12] Feature Focus: Component License Compatibility - CAST Highlight
https://doc.casthighlight.com/feature-focus-component-license-compatibility/

[14] Feature Focus: Enhanced Technical Debt Estimates - CAST Highlight
https://doc.casthighlight.com/feature-focus-enhanced-technical-debt-estimates/

[15] Feature Focus: Portfolio Advisor for Technical Debt - CAST Highlight
https://doc.casthighlight.com/feature-focus-portfolio-advisor-technical-debt/

[17] [18] [19] Checkmarx SAST Overview
https://docs.checkmarx.com/en/34965-46311-checkmarx-sast-overview.html

[20] [21] [22] [23] [24] [25] [32] Black Duck Software Composition Analysis (SCA) tool | Black Duck
https://www.blackduck.com/software-composition-analysis-tools/black-duck-sca.html

[26] [27] What is TruffleHog? ◆ Truffle Security Co.
https://trufflesecurity.com/trufflehog

28 29 Code Quality, Security & Static Analysis Tool with SonarQube | Sonar
https://www.sonarsource.com/products/sonarqube/

30 Sonar Combines SAST and SCA Tools in Single Offer - DevOps.com
https://devops.com/sonar-combines-sast-and-sca-tools-in-single-offer/

31 Managing Outsourced Software Development: Exploring Strategies for Code Quality | Sonar
https://www.sonarsource.com/learn/measuring-and-identifying-code-level-technical-debt-a-practical-guide/