Making the Most of What's Available

Using Jenkins and SonarQube in a scalable and certifiable verification process

Michael Baron, Emenda

Three key areas of the software verification process (release management, technical debt and management reporting) often go underutilised and unvalued. With increasing numbers of free and open source tools able to improve productivity, reduce costs and provide an insight into a projects' development cycle, this article looks at what is required of a modern verification process and how to make the most of what is available.

Release Management

When Continuous Integration (CI) tools, such as Jenkins, first appeared in the industry the focus was on build management, however since then the industry has transitioned to the more complicated and involved task of release management and associated Development Operations (DevOps).

Release management concerns the process from development to release, with optimal processes being automated through CI, with constant feedback to both the developer(s) and Quality Assurance (QA) teams. A well-designed release management process would mean that if one line of code was changed then the automation system should be able to handle the verification of the new release quickly and with no added human interaction. We should also aim to design a release management process that; is simplistic in design, functions as quick as possible to reduce the time to release, is reliable, should handle the configuration of the build automatically, optimise the developers' time, optimise the quality assurance activities, and enforce software quality across the project. The result of a well-functioning release management system is reduced time for product verification and shorter time to market.

To automate the whole release lifecycle, development teams have been adding in analysis and testing tools to achieve the quality assurance and verification of the product pre-release, thus, ensuring no surprises downstream. Generally, these tools are most effective when used in a 'per check-in' process but with an increased number of activities being integrated with the automation system, and combined with ever growing codebase sizes, the result is often a shift from fast 'per check-in' runs to periodic runs, every few hours, nightly or even in some cases over a weekend. The net result is that the release cycle is then slowed down significantly.

Some other common limitations when adding automated analysis and testing tools into the release cycle are:

- increased complexity in the configuration of the release management tool,
- long waiting times for testing results hindering developer and QA productivity,
- generation of large quantities of unnecessary data, reducing focus,
- vague, unrealistic, or undefined testing criteria and quality levels

Making the most of CI, and tools like Jenkins, allows us to outsource the test and analysis execution, and provide near real time feedback to the developer of the result. But, to succeed, additional test and analysis tools and steps also need to seamlessly connect into the CI platform provide appropriate and relevant data, and be both fast and scalable.

In addition, to make the most of Jenkins and SonarQube for release management, defining relevant quality gates and thresholds, ensures that we have a clear understanding of the progress toward the final release and allows us to continually ensure that we are compliant with the required quality standards – continuous compliance.

Technical Debt

In an ideal World, there wouldn't be any technical debt, however in practice that isn't realistic. Managing it effectively is the next best option.

Like other forms of debt, the cost of technical debt increases the longer it is held, and, of course, higher than if it were never acquired in the first place. Thus, preferably, the debt is resolved as soon as possible and by placing emphasis on providing fast feedback, closer to the development, when the debt is increased, it ensures that the costs are managed as best as they can be.

For example, if a defect is introduced into the code and the developer fixes it before check-in the relative technical debt is very low as it has only resulted in their time to re-work the code in a context they understand. If, however the defect makes its way into the system and is discovered before release in the quality assurance phase, the cost of the debt is both to recode and retest. Finally, if the defect makes its way into the release and it is discovered by the customer, the debt value is significantly higher due to it requiring recoding, possible branching from in development code, retesting and any financial implications or reputational costs of the software failure with the customer.

Strategies such as architectural de-coupling and change-based testing can help to reduce the impact of technical debt. For example, if build, test and analysis tools and processes can be configured to rebuild, retest and re-analyse only the modules and packages that need be - rather than everything - then removing technical debt becomes cheaper and thus the technical debt itself is reduced.

In addition to the CI platforms, such as Jenkins, tools like Docker, can help enable quicker analysis and configuration management.

However, there will still be technical debt, so using visualisation tools, such as SonarQube, that will help development and QA teams understand and share the technical debt results is vital. SonarQube also provides impact metrics to help find the 'low hanging fruit' enabling better allocation of resources for debt reduction.

Management Reporting

What is required from management reporting is an instant answer to "Can the product be released"? If the answer is no, then clear details on the metrics

that failed are needed, along with technical debt values to enable estimation of release dates and the resourcing required to get there.

Typically, the result from CI runs is used as the basis for many of these reports, and the results of each step and each tool within the cycle is collated in the relevant project page, for review, or alternatively failures are raised within a bug tracking system.

There are a wide range of limitations with using this type of reporting, for example:

- Jenkins focuses on the process rather than the results,
- the result of the process may be down to issues other than the test or analysis results,
- using multiple verification tools creates a fractured view of the overall result,
- results provided in differing formats,
- changing tools produces results in a different format again,
- reports typically list all the results and metrics and not just the new or changed issues,
- no details are provided for the overall technical debt for a given release

A dedicated 'dash-boarding tool', such as SonarQube, will pool together all the metrics from the various test and analysis tools, helping to mitigate these limitations. It also allows the thresholds and quality gates to be defined in one place and outputs one pass or fail result for clarity. The focus of SonarQube is on the results related to the quality (and security) of the code, meaning that it provides management and QA teams with only the data that they need, and not on build failures, etc.

SonarQube calculates technical debt from supplemented information and metrics provided by the test and analysis tools that it connects to, and allows for totals and breakdowns to be calculated across the full data set. Management and QA teams can then drill down into the results to focus in on the causes.

As the data from the external tools are collated and processed into a single project view, a change in verification tools generally has a lesser impact on the top-level results. The central dashboard also provides the ability to generate one overall compliance report based on the metrics from multiple tools.

Author

Michael Baron works as a technical consultant at Emenda, and is a long-term contributor of plugins to the Jenkins platform. Michael is experienced in providing DevOps consultancy to large businesses on the setup, configuration, and process design for the implementation of verification and validation tools within continuous integration (CI) and delivery (CD) development cycles. Michael has been a long-term supporter and evangelist for the utilisation of free and open source tools within the software development life cycle where possible and sensible, and he is also involved in numerous large-scale projects for Emenda around enterprise-wide technical debt (and technical wealth) reporting to board-level management teams.