From Continuous Integration to Continuous Assurance

James A. Kupsch, Barton P. Miller, Vamshi Basupalli, and Josef Burger
Computer Sciences Department, University of Wisconsin, Madison, WI 53706
Software Assurance Marketplace (SWAMP), 330 N. Orchard St., Madison, WI 53715

***Abstract*—Continuous assurance extends the concept of continuous integration into the software assurance space. The goal is to naturally integrate the security assessment of software into the software development workflow. The Software Assurance Marketplace (SWAMP) [1] was established to support continuous assurance, helping to simplify and automate the process of running code analysis tools, especially static code analysis (SCA) tools. We describe how the SWAMP can be integrated easily into the continuous assurance workflow, providing direct access from integrated development environments (IDEs) such as Eclipse, source code management systems such as git and Subversion, and continuous integration systems such as Jenkins.**

I. Introduction

The software development process takes us from the specification and design phase through deployment. *Continuous integration systems*, such as Jenkins [2] or Travis CI [3] try to capture much of the workflow of the development process into a tool that allows the developers to control and monitor the process. Each time a programmer commits changes to a common code repository, there must be corresponding merging, building and testing of the code. Given the increased awareness and importance of developing safe and secure software, this continuous integration process must also include running tools that analyze the code for security properties; adding such tools to the continuous integration process introduces the concept of *continuous assurance*.

Software assurance (SwA) defines the steps that we use during the development process to increase the confidence that the software is safe and correct. With a SwA process in place, a software system is more likely to operate as intended and have fewer vulnerabilities. SwA takes many different forms, from manual reviews and practices to automated tools. SwA processes should be used during all phases of the software development, and the more that is done, the greater will be the level of SwA. Unfortunately, SwA is not used as much as it should be, often because the cost to perform SwA, in both time and money, often exceeds the perceived benefits.

The Software Assurance Marketplace (SWAMP) [1] was established in 2012 to support continuous assurance, helping to simplify and automate the process of running software assurance tools, especially static code analysis (SCA) tools. SCA tools inspect the source code of the software for weaknesses without executing the code, and supports SwA during the implementation and testing phase. The ability to easily apply multiple analysis tools to a software system and view integrated results has been a strong point of using the SWAMP for software development. In its initial stages, interactions with the SWAMP were solely through a web-based user interface. While this interface is effective, it is separate from normal tasks of the programmer, including edits and compiles (often in an integrated development environment (IDE)), commits to a code repository, and control of workflow using a continuous integration tool. In this paper, we describe how the SWAMP's new web APIs and tool plug-ins provide direct access for submitting assessment runs and fetching results from the assessments. These APIs have been used to integrate IDEs (Eclipse), CI tools (Jenkins), and source code management systems (git and Subversion) into the SWAMP. Such integration allows the natural incorporation of continuous assurance without changing the developer's workflow.

We first present an overview of common software development methodologies, then describe the SWAMP services. Next, we present how we used the new SWAMP APIs in current tool integration and describe our plans for future integrations. Last, we discuss how the SWAMP's integration with software development systems compares to other available mechanisms to run SCA tools from software development systems.

II. Software Development Practices

Software development is a human activity that can be performed using a variety of processes. At its core, developing software is the process of writing source code in a computer programming language that is translated into a software system that functions correctly. The software system can be a single program, or multiple programs that are dependent on each other and possibly other software systems. Depending on the size and complexity of the software system, the development may be performed by a single developer or a team of developers. To organize this process, developers use methodologies and tools during the software development process.

The rest of this section briefly discusses software development methodologies and commonly used implementation and testing practices of these methodologies. The discussion is to illustrate where such assurance tools can be integrated; it is not meant to be an exhaustive description of these topics.

A. Software Development Methodologies

Software methodologies are the processes that a developer or team follows when producing a software system. For small projects and small teams, the process may be ad-hoc

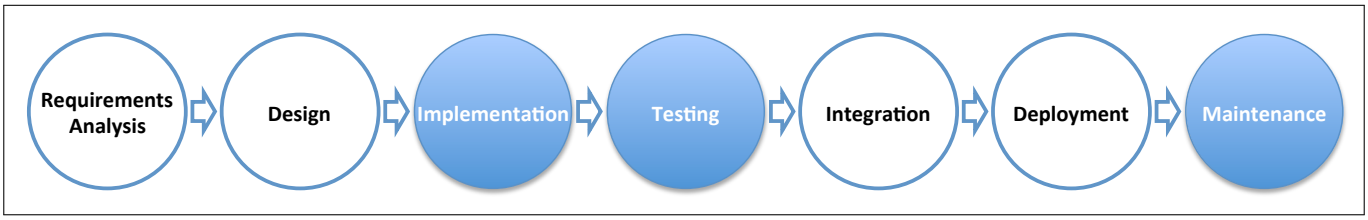

Fig. 1. The Waterfall Model of software development. In this methodology each stage is completed before proceeding to the next stage. The SWAMP supports the implementation, testing and maintenance stages.

and informal, but will often share characteristics of formal methodologies. As projects and teams become larger, a formal methodology is often used to impose structure on the development process. This allows the team to more efficiently work towards the goal of creating a correctly functioning system integrated from the individually developed components.

Software development processes have evolved over time, producing methodologies with different philosophies and characteristics. The type of methodology used to develop a software system is ultimately selected by the development team and organization based on team member's experience, philosophy, the type of software, and budget. Although there are many practiced methodologies, there are inherent tasks to produce a correctly functioning software system – such as designing the system, writing code, and testing – that are present in some form in all software development methodologies.

Figure 1 shows the Waterfall model [4], [5], one of the earliest documented methodologies dates from 1970. This methodology largely has developers complete each stage in order. The stages with short descriptions are:

1) *Requirements.* Gather and document how the system should operate.
2) *Design.* Design the system at a high level from the overall system to individual components.
3) *Implementation.* Implement the software system by creating the source code.
4) *Testing.* Test the source code produced to verify that it meets the requirements.
5) *Integration.* Integrate the system with other systems.
6) *Deployment.* Deploy the system to computers to make it available for use.
7) *Maintenance.* Maintain the software system to fix issues discovered after deployment, and to enhance the system as needed. The maintenance stage is typically accomplished using the Waterfall Model stages on a smaller scale.

In practice, developers use a more robust model than the Waterfall model as it is too simple. Most other software development methodologies have roughly similar stages, although the order and exact stages may differ slightly. Other differences include iteration of the stages, and how each stage is carried out. All methodologies include both *implementation* and *testing* steps. Other commonly used methodologies include: Prototyping [6], Incremental, Iterative and Incremental [7], Spiral [8], Extreme Programming (XP) [9], Rapid Application Development (RAD) [10], Agile [11], [12], Test Driven Development (TDD) [13], Continuous Integration (CI) [14], [15], Multistage CI, Continuous Delivery [16] and Dev Ops [17]. A book with a brief overview of these models is available [18].

B. Implementation Practices

The implementation step involves creating or modifying source code written in a programming language, and transforming the source code into a program or set of programs that form the software system (also known as *building* the software). Development teams also need to manage source code changes over time and between team members. Development tools assist in these activities, and most of them can be extended to perform other tasks through the use of plug-ins.

Text Editors and **IDEs** are used to create the source code. IDEs such as *Eclipse* [19], [20], [21] and *IntelliJ* [22], [23] provide the functionality of a text editor, plus allow a developer to perform most of the development tasks, including testing, all within a common graphical user interface.

Build Automation Systems such as *make* [24] *ant* [25] and *maven* [26] automate the often complex task of transforming the source code into executable programs.

Source Code Management System (SCMSs) such as *git* [27] and *subversion* [28] allow development teams to manage revisions to the source code. They also allow teams to easily merge changes created by different team members together to allow concurrent development of the source code.

C. Testing Practices

The testing step verifies that the software system functions as intended (correctness) and has no unintended functionality (lacks security weaknesses). As software is complex and difficult to create without errors, testing is an essential step in developing software. Here, we concentrate on testing for weaknesses. Testing can be categorized as static or dynamic. These approaches differ in the types of problems they can discover and the amount of effort that developers must expend to create the tests.

Static Analysis tools inspect the software's source, byte, or binary code without actually running the software. Compilers are our first line of defense, as they determine if the source is valid and produce warnings about questionable constructs. The quality of checking in compilers has radically improved over the last several years.

Static analysis tools may require modification to the build system or development process to run the tool in a way that

assures the code that is being assessed is the same as the code being built. There are some subtle traps here, as we must insure that we account for the language version, compiler options, and which files to assess [29]. While many commercial tools help automate this process, open source tools do not have such support. However, assessment runs in the SWAMP benefit from its automation support for all tools, commercial and open source.

Static analysis tools can find many type of flaws in the code, however are limited by the sophistication of the tool's semantic analysis algorithms, the time it takes to analyze the code, and the complexity of the code. There are several code features that can make the code too complex to be analyzed precisely. Such features include the use of pointers, both data and code, and inter-procedural and inter-compilation-unit behaviors. As a result, tools will often quietly ignore behaviors they do not understand (leading to false negatives) or make conservative assumptions about the code behavior (leading to false positives).

Dynamic Analysis tools test the software by running the code. Such tools will find actual weaknesses in the program (there are no false alarms), but the results are only as complete as the coverage generated by the test input. An additional challenge with dynamic tools is that the software must build, be installed and configured (along with other necessary software such as a database or web server), and have a test suite that executes the software. The drawback of dynamic analysis is that it is only as good as the test suite developed, or in the case of randomly generated test cases [30] having the randomly generated data uncover the flaw.

Continuous Integration (CI) Systems are used to automate the building and testing of a software system. A typical CI system allows the user to specify how to get the source code (from an SCMS or local files), when to build and test the package (each SCMS revision, periodically, or manually triggered), how to build the software package, and what tests and other actions to perform. By frequently integrating, building and testing the project's source code, developers can discover problems early and identify the revision that caused the build or tests to fail. CI tools provide a dashboard to view the current state of the software. To support new SCMS systems and new testing tools, CI system support a plug-in mechanism. Example CI systems include Jenkins SonarQube [31], and Travis CI.

III. The Software Assurance Marketplace

The SWAMP is a resource that automates the assessing of software with more than 30 available software assurance tools, both open source and commercial. The SWAMP comes in two versions, the SWAMP cloud service, called *MIR-SWAMP*, and a downloadable local version, called *SWAMP-in-a-Box, (SiB)*. Users of the MIR-SWAMP upload their software to the SWAMP, and have the software analysis tools applied without further interaction. The MIR-SWAMP currently supports static code analysis tools for software written in C, C++, Java, Python, Ruby, PHP, JavaScript, CSS, HTML and XML; and static binary analysis tools for Java Bytecode and Android APKs. When an analysis completes, users can review the merged results of multiple static analysis tools together in one of the result viewers. Access to the MIR-SWAMP is free and openly available (though some restrictions apply to the use of the commercial tools) at https://www.mir-swamp.org.

For organizations that do no want to (or can not) upload their software to the SWAMP service, SWAMP-in-a-Box is available for download and installation at a user's site from https://continuousassurance.org/swamp-in-a-box. This local version allows access to the functionality of the SWAMP while keeping the user's source code on-site. To use commercial tools with SWAMP-in-a-Box, the tools must be licensed from their provider (a *bring your own license (BYOL)*) model.

To use either version of the SWAMP, a user loads their package's source code along with a description of how to build it, selects operating system platform(s) and assurance tool(s); the SWAMP then builds the code and runs the assessment producing results to view. Once the source code and build description are uploaded all applicable tools can be used with no other user input or modifications to the software to build [29].

Users can interact with the SWAMP using a web browser to upload packages and their configuration, configure and perform assessments, and view results using an integrated version of Secure Decisions' Code Dx [32]. Denim Group's ThreadFix [33], or the basic SWAMP Native Viewer. The SWAMP can also be accessed using a web API.

Using the web-based interface, the SWAMP supports CI as the software is built and assessed using one or more static analysis tools. The status of the build and the results of static analysis are accessible from the web-based user interface. The next section describes how SWAMP assessment runs can easily be integrated with the software development workflow described in the Section II.

IV. SWAMP Integration with Software Development Methodologies

The SWAMP has been designed to integrate its assessment functionality with tools that support the software development process. While many users directly access SWAMP functionality via the standard web interfaces, the SWAMP also supports web APIs that can be used by plug-in modules for various software development tools.

The standard web interfaces allow the user to:

- Sign in, either with a local SWAMP identity or one from a federate identity management system such as inCommon, github or Google.
- Create a new project and upload the software to be assessed.
- Start an assessment, choosing from the large variety of tools provided in SWAMP.
- View the assessment results in the default results view or more sophisticated ones such as Secure Decisions' Code DX and Denim Group's ThreadFix.

The SWAMP web APIs provide programmatic interfaces to conduct these same basic functions. Using these web APIs,

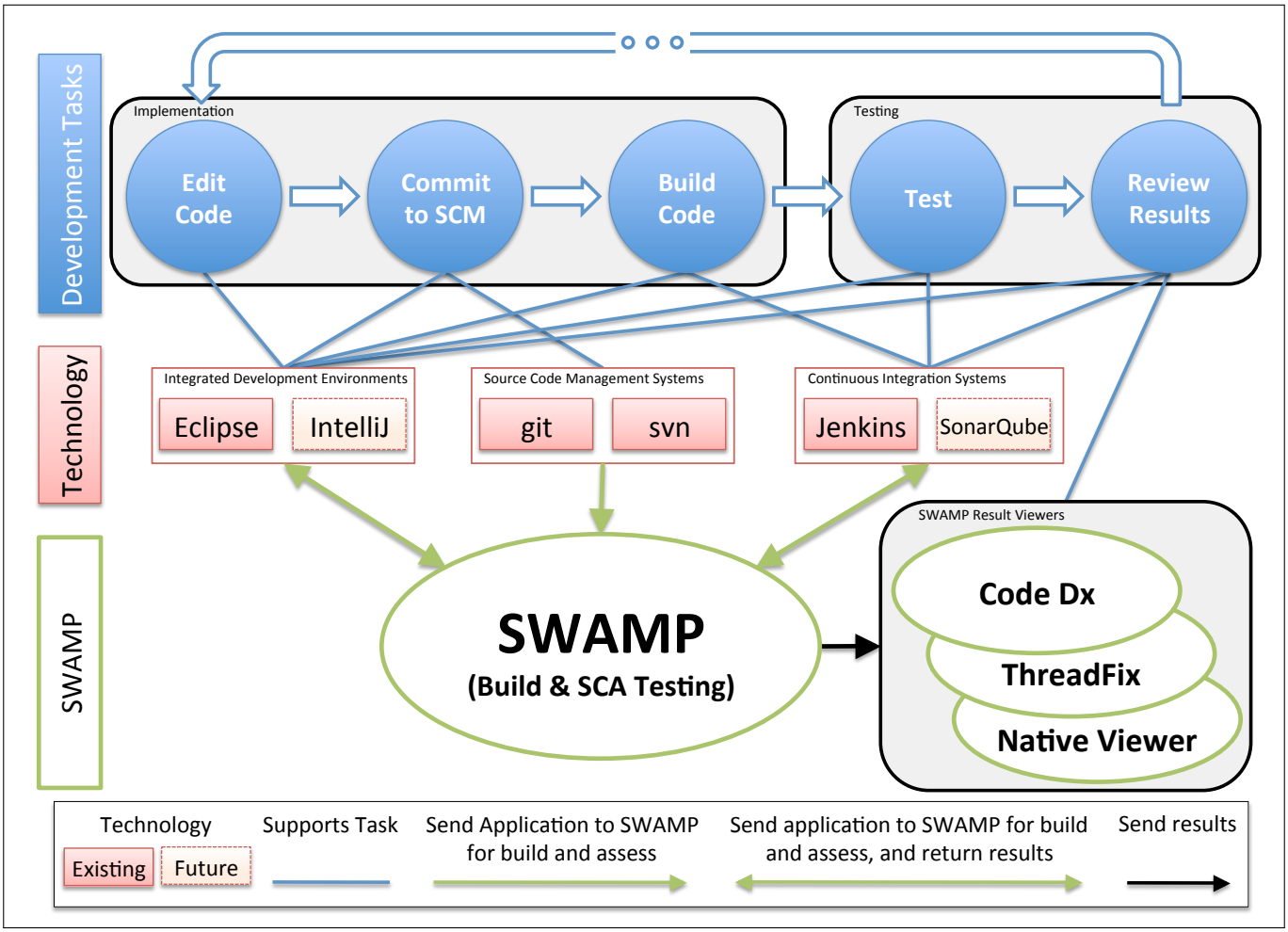

Fig. 2. Software development workflow, development tools and SWAMP integration points. Shows the workflow of tasks common to the typical implementation and testing steps of software development methodologies such as the Waterfall methodology shown in Figure 1, development tool technology commonly used by developers to perform the task, and current and future support to easily integrate this technology with the SWAMP to perform builds, run static assessment tools, and view results.

we have developed plug-ins and command line line tools that integrate the SWAMP with IDEs, SCMS, and CI systems. These plug-ins allow developers to gain the benefits of the SWAMP without disrupting their current workflow.

Figure 2 shows common development tasks within the implementation and testing steps, common types of technology used to perform these tasks, along with specific development technology where integration with the SWAMP is complete or planned, and how the SWAMP can integrate with the tasks or technology. The (blue) connections from Development Tasks to Technology indicate that the given technology is used to perform the corresponding task. The (green) arrows from the Technology to the SWAMP indicate a pathway to trigger builds and SCA assessments in the SWAMP, and potentially return the results. Lines with double-headed arrows indicate that the technology directly displays results produced by the SWAMP to the user. For all assessments performed, results can also be inspected using one of the SWAMP result viewers, which currently include Code Dx, ThreadFix, and the basic SWAMP Native Viewer.

Common to all the plug-ins is a one-time common configuration task for each plug-in to specify the URL to the SWAMP instance to use, the user's credential, the SWAMP project, the SWAMP package, and the tools and platforms to use for assessments.

The remainder of this section describes the integration available for existing software development tools to easily perform assessments in the SWAMP and to view results where applicable.

A. *Integrated Development Environments*

The ability to trigger assessments and view results directly in an IDE is advantageous as the developer stays within their work environment, and sees results directly in the interface where they can fix the reported problems. Figure 3 shows the user interface presented to view results after an assessment has been triggered.

A SWAMP plug-in for the Eclipse IDE is available. The plug-in currently supports software written in Java, C, or C++.

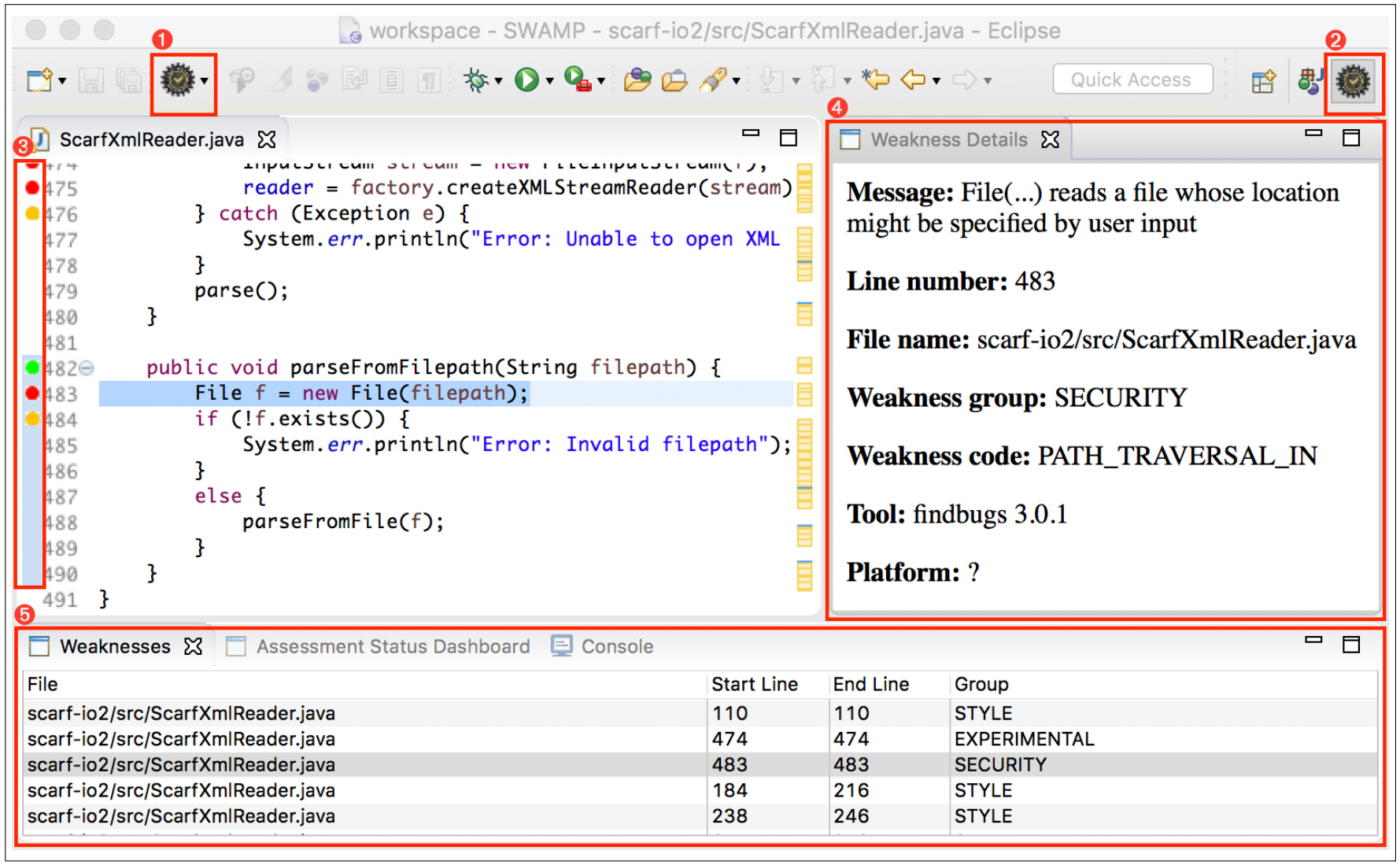

Fig. 3. Screenshot of Eclipse with the SWAMP plug-in displaying the SWAMP perspective (set of panels) after an assessment has been run: ❶ the SWAMP Logo button and menu used to trigger an assessment in the SWAMP of the current Eclipse project and to access SWAMP setting and functions, ❷ SWAMP perspective button used to switch to the set of SWAMP panels, ❸ circular markers indicating lines where weaknesses were found in the project by SWAMP SCA tools with colors indicating tool types, ❹ the Weakness Details panel displaying the currently selected weakness, and ❺ the Weakness tab containing a tabular view of the weaknesses found.

The SWAMP Eclipse plug-in supports assessing the source code with any or all of the tools available in the SWAMP on any or all of the operating system platforms available, and directly viewing results within Eclipse. The plug-in supports any of the platforms that Eclipse runs on, but the assessment platforms available in the SWAMP are currently all Linux based operating systems.

Besides the common configuration items, no other information is required. SWAMP assessments are performed by simply clicking the SWAMP button in the IDE or selecting a menu item. The software then is uploaded, assessed in the SWAMP, and the results are retrieved for display within the IDE.

The combined results of all the configured tools are displayed in the IDE with markers alongside lines with weaknesses, a table view, and a view that displays the details of a weakness.

Support for other IDEs such as IntelliJ are planned for the future.

B. Source Code Management Systems

The ability to trigger SWAMP assessments from an SCMS when an update occurs allows assessments to be performed on every commit, directly supporting one of the core tenants of continuous integration and continuous assurance. From this it is possible to determine when a broken build was introduced or a particular weakness was introduced.

SWAMP plug-ins for both git and Subversion are available. The plug-ins support any of the languages supported by the SWAMP. Since there is no user interface available for displaying results in an SCMS, viewing results must be done using the SWAMP's user interface.

As an SCMS is generic with respect to its contents, it is not possible to automatically configure the package to be built in the SWAMP as is done with IDEs. The user must create a small package configuration file that describes how to build the package in the SWAMP, in addition to the common configuration. Once the configuration file is created it rarely needs to be modified unless the package's build process changes. As large projects have a high volume of SCMS commits, fine grained control of the commits that trigger an assessment is necessary. Use cases include a developer wanting assessments for all commits to their SCMS repository, or a development team wanting assessments for merges to a set of development branches of their shared team repository. The plug-in supports triggering based on branch names, and on selecting between a commit that is local versus being pushed from another SCMS repository.

C. Continuous Integration Systems

The ability to trigger SWAMP assessments from a CI system allows users of a CI system to take advantage of tools and platforms available within the SWAMP. It also allows the source code assessed in the SWAMP to use the CI system to acquire the source code and to determine when assessments are performed. If a development team is already using a CI system, they can continue to use it and easily gain access to the analysis provided by the SWAMP.

A SWAMP plug-in for Jenkins that can upload source code and perform assessments is currently available. When assessments complete, Jenkins displays trend graphs of the discovered weaknesses, and allows the user to view the results from the tools including seeing the source code context of individual weaknesses. Figure 4 shows three aspects of the user interface provided by Jenkins with the SWAMP plug-in after the assessment of ten versions of the project.

Besides the common configuration, users also need to specify how to build the package in the SWAMP. This information should rarely change unless the packages build process changes.

Support for other CI systems such as SonarQube and Travis CI are planned for the future.

D. Command Line Interface (CLI) for Other Systems

Besides the plug-ins above, the underlying technology for the plug-ins is available as a command line interface. The SWAMP CLI allows developers to interact with the SWAMP from scripts or systems that are not already directly supported by an existing plug-in.

To use the SWAMP CLI, users need to provide the common configuration along with a specification of how to build the package, and an archive of the package's source code. The configuration information should rarely change unless the packages build process changes.

V. Related Techniques

IDEs, SCMSs, and CI systems also provide a means to directly run SCA tools (as opposed to using the SWAMP cloud service or a locally-installed SWAMP-in-a-Box). In this section, we contrast the direct running of an SCA tool with the use of the SWAMP as an integrated assessment service.

The first, and most obvious benefit of using the SWAMP is the direct access to an extensive collection of SCA tools (as described in Section III). The SWAMP supports the assessment of a large variety of programming languages and multiple tools for each language. Good software assurance practice encourages that we run multiple tools to get the most comprehensive results and coverage of potential weaknesses that may expose vulnerabilities.

Along with the benefit of having access to multiple tools in the SWAMP, comes ease of use. In the SWAMP, there is no need for a user to download, install, or configure a tool.

One minor disadvantage of running on the SWAMP cloud service is an assessment will take a longer than locally running the same tool on a user's computer due to SWAMP's overhead. When assessing an application with multiple SCA tools, this cost differential is reduced or eliminated due to the SWAMP's ability to concurrently run many simultaneous assessments. This performance difference is less of an issue when running on the locally-installed SWAMP-in-a-Box.

The Eclipse and IntelliJ IDEs both support a limited number of static analysis tools to be run locally within the IDE.

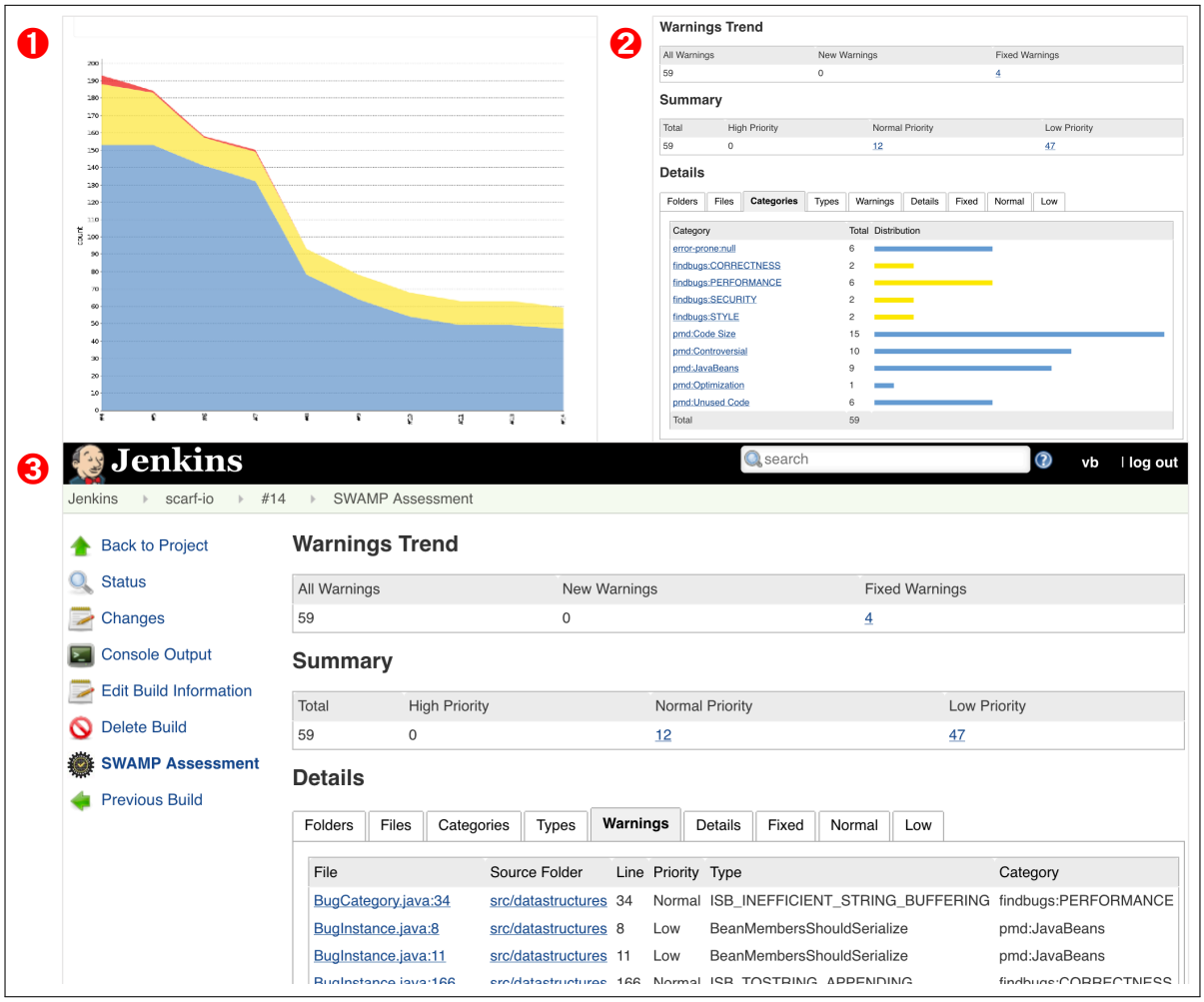

Fig. 4. Three screenshots of Jenkin with the SWAMP plug-in after ten versions of the project have been assessed. ❶ The weaknesses trend graph, displayed on Jenkins' project status page, showing the developer reducing weaknesses with each successive commit. The weaknesses are categorized from bottom to top as Low, Normal and High priority weaknesses shown in blue, yellow and red respectively. ❷ The SWAMP Assessment view of weaknesses by category showing the results of the tenth assessment. The colors are the same as above. ❸ Whole Jenkins screenshot showing the SWAMP Assessment view as a individual warnings (all 59 warnings are viewable by scrolling the web browser). The table is sortable and clicking on a individual rows shows the source code or warnings filtered by directory.

Both IDEs provide the best support for Java, with plug-ins for the FindBugs and PMD SCA tools. Eclipse also provides support for cppcheck and SonarQube's analysis tool. For other languages, the number of static analysis tools supported is smaller or nonexistent.

The git and Subversion SCMSs both provide hooks for invoking a program when an event such as a new commit occurs. The commit could be made to a developer's working branch or when merging to an integration branch. The SWAMP provides hooks for both git and Subversion to automatically upload the source code to the SWAMP and start an assessment run. While there are a few readily available recipes and scripts to add hooks to git and Subversion to directly trigger an SCA tool to run, the user must still install and configure these tools themselves.

Both Github and GitLab, widely used repository services based on git, can trigger an HTTP request to a remote server when a version is committed or merged. Since most CI systems (such as Jenkins or SonarQube) can accept the github HTTP request, this request can be used to trigger an assessment (as we describe below). SWAMP support for handling such HTTP requests is currently in the release process.

Jenkins supports many of the SCA tools provided by the SWAMP for Java, C and C++. There are three aspects to running an SCA tool assessment from Jenkins. First, the user must configure the build system to include a new target for running an assessment. Such a configuration is simple for maven but more complex for other build systems. Second, the user must install a plug-in from the Jenkins marketplace for each SCA tool that they want to run. The plug-in translates the tool output into a format that Jenkins can process and display. Third, each SCA tool must be locally installed.

SonarQube is a CI system that offers a no-cost community edition and a for-purchase professional edition. SonarQube

supports more than twenty languages with about half of them being available at no cost. For all the languages, SonarQube provides their own SCA tool that looks for weaknesses and collects metrics. For four of the languages, additional add-on tools are available from the SonarQube marketplace with some incurring an additional fee. Unlike Jenkins, the software is not built and assessed on the SonarQube server. Instead SonarQube requires the user to run a language specific SonarQube Scanner on another host. The Scanner installs (for most tools) and runs the SCA tool, and afterwards uploads the results to the SonarQube server for viewing. The Scanner manages much of the complexity of installing and running an SCA tool.

Ready-to-use support for running SCA tools in Travis CI is limited. The only tool is Coverity Scan [34], and only if your software meets Coverity's eligibility requirements.

VI. Conclusion

Since 2012, the SWAMP has been promoting the inclusion of software assurance as an intrinsic part of the software development process. We introduced the idea of *continuous assurance* as a natural extension of the continuous integration process. The key has been to not only integrate the use of assessment tools into the development process, but to do so in a way that reduce the obstacles faced by the programmer. By doing so, we reward the programmer for using such tools and create an environment where software assurance is considered natural and essential. Note that all the software components produced by the SWAMP project are open source, so as to encourage the broadest adoption of these practices in both the commercial and research communities. This is an ongoing process and we (and many other groups) continue to strive to develop new ways to streamline the inclusion of software assurance into the development process.

Acknowledgments

This work was supported in by U.S. Department of Homeland Security under AFRL Contract FA8750-12-2-0289. We are grateful to Miron Livny, James Basney, and Von Welch for their feedback and comments, and for the efforts and support of the entire SWAMP team at the Computer Sciences Department at the University of Wisconsin, the University of Illinois, the Morgridge Institute of Research, and Indiana University.

We appreciate the thoughtful comments and suggestions on our early manuscript from Kevin Greene (SWAMP DHS Project Sponsor).

References

[1] *Software Assurance Marketplace (SWAMP) Web Site*, Software Assurance Marketplace, https://www.continuousassurance.org/.

[2] J. F. Smart, *Jenkins A definitive Guide: Continuous Integration for the Masses*. O'Reilly Media, July 2011, http://shop.oreilly.com/product/0636920010326.do.

[3] *Travis CI*, Travis CI, GmbH, http://travis-ci.org/.

[4] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, pp. 328–338. [Online]. Available: http://dl.acm.org/citation.cfm?id=41765.41801

[5] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?" in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 61–68. [Online]. Available: http://dl.acm.org/citation.cfm?id=800253.807650

[6] F. P. Brooks, Jr., *The Mythical Man-month*, anniversary ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[7] C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, vol. 36, no. 6, pp. 47–56, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1204375

[8] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, May 1988. [Online]. Available: http://dx.doi.org/10.1109/2.59

[9] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley Professional, 2004.

[10] J. Martin, *Rapid Application Development*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 1991.

[11] J. Shore and S. Warden, *The Art of Agile Development*, 1st ed. O'Reilly, 2007.

[12] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.

[13] Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[14] G. Booch, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[15] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Addison-Wesley Professional, 2007.

[16] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[17] J. Davis and K. Daniels, *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*, 1st ed. O'Reilly Media, Inc., 2016.

[18] B. K. Jayaswal and P. C. Patton, *Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software*. Prentice Hall PTR, 2006.

[19] *Eclipse IDE*, The Eclipse Foundation, http://eclipse.org/.

[20] *Eclipse Platform Technical Overview*, International Business Machines Corp., http://eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf.

[21] S. Holzner, *Eclipse: Programming Java Applications*. O'Reilly Media, 2004, http://shop.oreilly.com/product/9780596006419.do.

[22] *IntelliJ IDEA*, Jet Brains, https://www.jetbrains.com/idea/.

[23] J. Krochmalski, *IntelliJ IDEA Essentials*. Packt Publishing, December 2014, https://www.packtpub.com/application-development/intellij-idea-essentials.

[24] R. M. Stallman, R. McGrath, and P. D. Smith, *GNU Make Reference Manual*, 2nd ed. Free Software Foundation, May 2016, https://www.gnu.org/software/make/manual/make.pdf.

[25] *Apache Ant Project*, Apache Software Foundation, http://ant.apache.org/.

[26] *Apache Maven*, Apache Software Foundation, http://maven.apache.org/.

[27] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*, 2nd ed. O'Reilly Media, August 2012, http://shop.oreilly.com/product/0636920022862.do.

[28] C. M. Pilato and B. Collins-Sussman, *Version Control with Subversion: Next Generation Open Source Version Control*. O'Reilly Media, June 2004, http://shop.oreilly.com/product/9780596004484.do.

[29] J. A. Kupsch, E. Heymann, B. Miller, and V. Basupalli, "Bad and good news about using software assurance tools," *Software: Practice and Experience*, pp. n/a–n/a, 2016, spe.2401. [Online]. Available: http://dx.doi.org/10.1002/spe.2401

[30] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, Dec. 1990. [Online]. Available: http://doi.acm.org/10.1145/96267.96279

[31] G. A. Campbell and P. P. Papapetrou, *SonarQube in Action*, November 2013, https://www.manning.com/books/sonarqube-in-action.

[32] *Code Dx*, Secure Decisions, https://codedx.com/.

[33] *ThreadFix*, Denim Group, https://www.threadfix.it/.

[34] *Coverity Scan*, Synopsis, https://scan.coverity.com/.