Intelligently Transparent Software Ecosystems

James Herbsleb, Christian Kästner, Christopher Bogart

Carnegie Mellon University

Abstract

Today's social coding tools foreshadow a transformation of the software industry, as it increasingly relies on open libraries, frameworks, and code fragments. Our vision calls for new "intelligently transparent" services that support rapid development of innovative products while managing risk and receiving early warnings of looming failures. Intelligent transparency is enabled by an infrastructure that applies analytics to data from all phases of the lifecycle of open source projects, from development to deployment, bringing stakeholders the information they need when they need it.

Keywords

D.2.6 - Programming Environments, D.2.8 - Metrics, K.6.3 - Software Management

Introduction

Innovative transparent environments such as GitHub, LaunchPad, and BitBucket, are exerting a profound influence on how a new generation thinks about software development. They continue a widespread trend toward openness – it is no longer surprising that individuals and commercial firms can form communities that develop and maintain valuable and freely available software assets. What is new is the combination of distributed version control and social media features that create "transparent" environments, capable of scaling up ecosystems well into the millions of repositories and developers [1]. This trend will accelerate as large-scale data analytics add transformative intelligent services.

Increasingly, software development is a matter of selecting useful libraries, frameworks, and other components, and quickly wiring them together. The result is impressive functionality produced very rapidly. But this approach comes with serious risks, as code is used without being thoroughly understood, new combinations with potentially dangerous

interactions are created, and hard-to-evaluate code contributions are offered up by strangers.

Yet these risks create a business opportunity. The data potentially generated by transparent environments contain the fodder for novel ideas that will further speed development and help manage risk. We call this idea "intelligent transparency" and sketch how it could work. We begin with a scenario . . .

An Opportunity: The IRS Offers Up Big Tax Data

In 2020 the IRS announces it will release an anonymized version of its tax return database. Immediately, the race is on to create applications and services that will take advantage of insights and predictions enabled by this database. TaxTech, a firm supplying tax software and services, swings into action. TaxTech uses its extensive knowledge of customers' tax needs to dream up innovative applications, such as a tool for small businesses that lets them benchmark their tax burden, deductions, and credits against similar businesses nationally and by region. While they have extensive experience with tax services and enterprise data, they are not confident they have the right tools and infrastructure to engineer services using data on this national scale. Design begins by seeking an appropriate language, web framework, database, and libraries for visualization and data manipulation. Knowing time-to-market will confer first mover advantage, TaxTech engages a business we envision – an intelligent software assurance and monitoring (ISAM) firm. ISAMs provide evidence-based component recommendations, and, using data provided by existing customers, provide a wide range of early warnings as customer applications and open source components evolve.

Recommending ensembles

ISAMs have extensive experience with open source projects and know established and rising candidates in many fields. In addition, advances in code search based on recent work in semantic search [2] and query reformulation [3] support them in finding a pool of candidates. Then comes the difficult step of choosing among these candidates and finding suitable components, which used to be an arduous and highly uncertain manual process. But the ISAM applies analytics to its proprietary database of software-in-use to quickly identify the best candidate *ensembles*, i.e., stacks or sets of software that work well together. ISAMs maintain extensive logs of their customers' past use, which reveals a substantial history of components used together in a great variety of combinations. From run-time data and test results supplied by past and current customers, the ISAMs use a variety of analytic techniques [4, 5] to identify and eliminate buggy components, avoid architectural mismatch, predict fault-prone ensembles, estimate the cost of glue code

development for each potential ensemble, and tailor their predictions to their customers' environments.

In addition to evaluating the technical characteristics of components and ensembles, ISAMs also make effective use of the detailed activity and social data available from open source hosting environments. Customers want to avoid components that are immature and volatile. They want projects that are well managed, meaning that issues and code contributions are handled promptly and professionally. They want projects supported by a dedicated and skilled community that will continue to remain vibrant. Based on extensive research on success in online communities [6] and on knowledge of open source communities specifically [7], ISAMs apply analytics to a variety of community variables, including activities within the community and its code forks as well as the profiles and commit histories of its developers, to predict community sustainability, technical experience and proficiency, and responsiveness.

The ISAM identifies several potential ensembles that will serve TaxTech's needs, but with slightly different quality/risk profiles. A component in one candidate ensemble, for example, recently gave rise to a sharp increase in bug reports when a new database version was introduced, but those appear to have been resolved. Another ensemble used a new web services framework with a too-small and too-inexperienced community of contributors, but community membership and experience levels were trending upward. Stability analysis of a third, based on development observed in forks, discussions about the need for specific changes, and a growing list of related feature requests, hinted at future issues with backward compatibility. But all the recommended ensembles have levels of risk acceptable to TaxTech, which quickly makes its choice.

Ongoing Vigilance

Through the following four years, TaxTech flourishes, and the new services generate big revenue. But software evolution brings new risks, especially since key components are not directly in TaxTech's control. Notification services alert TaxTech to emergent risks of community deterioration, such as reduction in repository activity, rising numbers of unaddressed bug reports and pull requests, the development of controversy in mailing lists and comments, an increase in changes breaking backward compatibility, and a change of focus of activity from the current repository to one of its forks.

Another primary risk of reliance on open source components is that elements of an ensemble will become incompatible. TaxTech could get stuck on old versions, without benefiting from new features, bug fixes, or security updates, or would have to manually apply changes and resolve conflicts. With new notification mechanisms however, TaxTech developers get tailored information whenever a component evolves so they can react before technical debt mounts. Basing such notifications on development forks, well ahead

of product releases, TaxTech can upgrade quickly on release date, or even contact developers to negotiate the direction of the changes. In addition, the system triggers notifications of suggested changes that have attracted many comments, hinting at controversy or complexity, or when substantial new development occurs in forks. Both allow a peek into the future of the project. Through timely notification, TaxTech has had the opportunity to influence developers, avoid disruption, and advocate for inclusion of useful features.

The biggest benefit is to avoid being taken by surprise, and to have time to plan a response to risks that arise. In the next sections, we sketch the two major design ideas that support this scenario.

Enhance Transparency with Analytics

We borrow Bernstein's definition [8] of transparency: "accurate observability, of an organization's low-level activities, routines, behaviors, output, and performance" (p. 181). Transparent environments such as GitHub allow anyone to easily fork and manipulate code in any repository, examine all commits in forks and master repositories, see all comments linked directly to the artifacts they refer to, and to find detailed information about people, their activities, and their social connections. Transparency makes it much easier to find useful code supported by viable communities, and to monitor code one is using to detect changes that create problems or opportunities [9]. On the down side, transparency can present developers with overwhelming amounts of information.

Characteristics developers care about, such as quality attributes of a component, are often not directly observable, so developers use things they can see as signals from which they infer hidden software qualities as well as the durability and responsiveness of the community maintaining it. This is currently an imperfect and time-consuming practice.

Intelligently transparent environments will streamline and expand these capabilities. The speed and accuracy of inferences will be enhanced by computational agents that quickly summarize the information developers want to see. For example, when choosing among a number of candidate libraries or frameworks, developers look for signals that the project is "alive," that it has a group of people who are committed to it, that it evolves without frequent disruptions to downstream projects, that the project is skillfully managed, and that the project has been well-received by the community. A variety of signals are useful for assessing these hidden qualities [10], including the commit velocity, the diversity of frequent committers, the number of "stars" or "likes" the project has, the number of test cases, the history of continuous integration results, and a history of issues and pull requests being quickly addressed.

While it is laborious to manually examine all of the many signals one wishes to see to assess and compare the suitability of projects, it is feasible for a computational agent to acquire the data and present it in terse form, such as a dashboard visualization. Indeed, many people are experimenting with such visualizations now (see, e.g., GitHub Visualizer, for a collection of visualizations http://ghv.artzub.com). Such tools can transform tedious tasks into tasks that can be resolved by quick perusal of a visual display.

Future analytics research will also bring novel forms of information that help navigate the challenges of *evolving* components. Intelligent transparency can support developers in discovering interesting changes among a sea of constantly evolving projects, and in identifying changes that encourage further actions, such as breaking interface changes or new useful features. Tech Angels' product Gemnasium (https://gemnasium.com/), for example, is very much in this spirit. It notifies users of updates and security vulnerabilities in any of their dependencies so they can take appropriate action without having to constantly monitor all changes in all their dependencies. In our scenario, this kind of awareness functionality allows TaxTech to avoid disruptions and failures as the open source projects on which they depend evolve.

Information overload is also a problem currently being addressed. Existing approaches like YooHoo [11] and NeedFeed [12] have already shown that it is possible to reduce the notification clutter in systems like GitHub significantly when using simple mechanisms to filter important messages. For example, YooHoo identifies those changes that break binary compatibility in used library code, a mere 7% of all changes, and NeedFeed uses code ownership and past changes in the code as simple heuristics to identify relevant changes.

A further strategy to identify relevant changes when they happen, and even predict upcoming changes, will be to develop *indicators of stability,* derived from activity in forks, mailing lists, bug trackers, and other communication channels. Ecosystem environments are unpredictable, because the components one uses are controlled by others who can change them at will. There are no stability guarantees. Stability indicators can provide vital information both for component developers and users.

Different facets of stability can be inferred from many sources. For example, developers can simply declare an *intent* that an interface won't change, or will remain backwardly compatible. In addition, *historical stability*, both averages and trends, can be observed by mining the repository [13]. Furthermore, by analyzing dependencies within an ecosystem, signals can be derived about the *context* in which a component is currently being used, e.g., components used by components intended to be stable should evolve more conservatively. Using components not intended to be stable indirectly indicates that a component is likely to change. If historically unstable components are used by components intended to be stable, this suggests a *stability conflict* that should be addressed. Abrupt changes or mismatches among intended, historical, and contextual stability provide important,

actionable information for developers and users that can help them decide where to implement new functionality, what projects and APIs to use, how to avoid disrupting users, and what activity in other repositories requires immediate attention.

In the future, as analytic techniques are refined and very-large-scale data sets become available, research will push intelligent transparency beyond filtering and stability, to infer developer intent for a wide range of use cases and from a wide range of sources. These are all forms of additional notifications and reports that ISAMs can provide in the longer run, enabling firms like TaxTech to respond proactively.

Examples include:

- Indicators and signals to identify commits that deserve more attention or review, considering for example the developer's experience and the centrality of the code being changed [14] (see Figure 1 where such commits are highlighted in red),
- The probability that a specific pull request will be accepted,
- Likely future activity level (new features, bug fixes) for a project
- Over-dependence on continued contributions of few core developers, and
- A summary, derived from package managers and clone detection, of all uses of project code, broken down by type of user and key attributes of use, to help avoid disrupting users.

Results gathered automatically by intelligent transparency mechanisms can be visually integrated into platforms such as GitHub or provided as an independent service. While intelligence useful for selecting components has its basis in analysis of repositories and developer activity, critical monitoring services will also use runtime data.

Analyze Runtime Data to Provide Monitoring Services

Using free software grown in the wild exposes businesses to unpredictable failures and security threats. ISAMs also tackle these adoption risks. Some ISAM-like services currently exist for internal use for proprietary software, like Apple and Microsoft's operating systems, and some consultants have amassed a lot of experience in particular domains. Yet such services are not available, at sufficiently detailed level or breadth, for monitoring the ecosystems of diverse open-source software on which businesses often depend.

Fortunately, some users of software components are willing to accept more risk than others. Open source projects presumably follow a traditional adoption curve [15]. The first to take it up will be those with a high tolerance for risk coupled with a compelling business need for novel functionality. They may be writing mobile apps, for example, that do not touch sensitive data but require novel computation. If a new component has features that

attract early adopters, the attention helps to ensure that bugs will be discovered and fixed quickly. If developers can observe that a component is widely used in diverse contexts, and has become stable, it will be regarded as trustworthy. Earlier adopters act, in effect, as field testers for those who are more risk averse, but this has impact *only if their use is visible*. It would be to any adopter's advantage to know what position on the adoption curve any given project occupies, and exactly what sorts of "testing" have been carried out, in which domains, platforms, and configurations. It is this critical piece of information ISAM providers can supply.

ISAM providers monitor deployed *usage* of the software, by supplying their customers -- with their knowledge and consent, under appropriate confidentiality arrangements -- with instrumented versions of the software packages the ISAM thinks their customers will want. These instrumented versions will send data about use back to the ISAM provider. (See Figure 2.) This data captures what version of what software is deployed, in what hardware and software environment, how often, how it performs, and the circumstances of failures. It can even provide a monitoring platform for the vendor to capture domain-specific statistics of interest: for a web server software it might capture a characterization of traffic shape and performance; for a scientific algorithm, it might summarize statistics on the kind of data fed to it; for an IDE, it might record user settings and installed plugins. The runtime data can be visualized in several ways -- an example prototype is shown in Figure 3. The graph view (on the left) shows which packages the focal package (gtools) depends on (solid lines) or is frequently used with (dotted lines). The bar charts show upstream and downstream dependencies, and the frequency with which these packages are actually called in a set of runs.

There are, of course, potential privacy issues with runtime data, but there is reason to think they are not insurmountable (see sidebar).

The advantages for customer and ISAM provider are substantial. For the customer, monitoring could provide early warning or ideally allow them to avoid most failures and downtime. For example, if a failure is detected at one customer site, the ISAM provider could immediately contact the software's open source developers and work with them and the customer on a fix. By informing developers in detail about how their software is used, and how widespread the impacts of specific failures are likely to be, they help the developers establish priorities. It could also contact customers running a similar configuration, giving them a detailed warning of a possible failure, allowing the customer to prepare backup plans or workarounds.

In addition to early detection of defects and an early warning system, ISAM providers will develop proprietary algorithms that use the vast store of runtime data, and make custom assessment reports available to their customers, tailored to their tolerance for risk and priorities among quality attributes. Customers considering the use of a given component or

ensemble would benefit from the accumulated experience of other users similarly situated. These reports will be updated frequently, since as any given project gets used more, it produces more data and becomes more mature, stable, and secure. Customers receiving reports will be in a much better position to create for themselves a portfolio of software assets that balances their need for innovation and tolerance for risk.

Finally, ISAMs could make basic usage data public to benefit the overall ecosystem. Wider use of a software project compared to its competitors can increase its use further, much as receiving "likes" on news services tends to lead more people to read, generating still more likes. For software, more attention also encourages more people to fix bugs and offer new functionality. The accelerated pace of evolution sparked by the extra attention initiates a virtuous cycle of wider adoption and rapid improvement.

Conclusion

Open software ecosystems are a rich source of libraries, frameworks, and code fragments that can reduce programmer effort and accelerate development speed. Yet they require time and effort to evaluate, they expose users to the risks of poor selection, and introduce the uncertainties of becoming dependent on code whose evolution is controlled by someone else. By making use of analytics applied to the detailed activity and communication traces in transparent environments, and by acquiring and analyzing ongoing runtime data, ISAMs can provide a valuable service that supports well-informed choices and provides timely warnings that help the application developer negotiate for favorable changes in upstream software, or prepare to migrate to alternative components. Armed with solid, contextualized empirical data and analytics, software quality will improve, and development will become faster, cheaper, and more predictable.

Sidebar

Runtime monitoring. Software users are rightfully vigilant about software that communicates their activities. But there is an important place for open source software monitoring when it is done with the knowledge, consent, and for the benefit, of the users. For example, Fahey, McLay and Agrawal's XALT [S1] tracks use of both open source and commercial software on supercomputers, through shell wrappers that users can choose to use; XALT captures library and resource usage without special instrumentation of the individual packages. The Condor distributed batch system reported basic data of its own usage on 50,000 CPUs over 1000 sites [S2]. Other efforts have included usage statistics like the Debian Popularity Contest (http://popcon.debian.org/) and crash reporting as in Ubuntu, Mozilla [S3], and Microsoft products [S4].

In the scenario we envision, ISAM customers agree to use instrumented software versions and provide their runtime data and test results, under suitable confidentiality arrangements, as part of a contract that allows the ISAM to provide monitoring services that would not be possible without the usage data. We expect there will be many software firms willing to accept runtime monitoring in order to reap the benefits an ISAM can provide, and many of their customers who will consent, as many users do now for crash reporting and quality improvement.

Sidebar references:

[S1] M. Fahey, R. McLay, and K. Agrawal, *XALT Design and Installation Manual*, 2015.

[S2] D. Thain, T. Tannenbaum, and M. Livny, 2006 "How to measure a large open-source distributed system," In *Concurrency and Computation: Practice and Experience*, Wiley InterScience.

[S3] Socorro: Mozilla's Crash Reporting System, http://blog.mozilla.com/webdev/2010/05/19/socorro-mozilla- crash-reports

[S4] K. Kinshumann, et al., "Debugging in the (very) large: ten years of implementation and experience," *Com. of the ACM*, vol. 54, no. 7, 2011, pp. 111-116.

Acknowledgments

The authors gratefully acknowledge support from National Science Foundation grants 1064209, 1322278, and 1111750, and a grant from the Alfred P. Sloan Foundation.

References

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Leveraging Transparency," *IEEE Software,* vol. 30, no. 1, 2013, pp. 37-43.

[2] K.T Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," *ACM Trans. on Software Eng. and Methodology (TOSEM)*, vol. 23, no. 3, 2014, 26:1-26.

[3] L. Martie, T. D. LaToza, and A. van der Hoek, "CodeExchange: Supporting Reformulation of Code Queries in Context", *Proc. 30th Int'l Conf. on Automated Software Eng. (ASE)*, to appear, 2015.

[4] T. Menzies, and T. Zimmermann, "Software analytics: so what?," *IEEE Software*, vol. 30, no. 4, 2013, pp. 31-37.

[5] H. Cleve, and A. Zeller, "Locating causes of program failures," *Proc. of the 27th Int'l Conf. on Software Eng*, 2005, pp. 342-351.

[6] R.E. Kraut and P. Resnick (Eds.), *Buildling Successful Online Communities: Evidence-Based Social Design*, 2011, Cambridge, MA: MIT Press.

[7] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/Libre open-source software development: What we know and what we do not know," *ACM Comp. Surveys (CSUR)*, vol. 44, no.2, 2012, art. 7.

[8] E.S. Bernstein, (2012) "The transparency paradox a role for privacy in organizational learning and operational control," *Admin. Sci. Quart.*, vol. 57, no. 2, 2012, pp. 181-216.

[9] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, (2012) "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," *Proc. ACM Conf. on Comp.-Supported Cooperative Work,* 2012, pp. 1277-1286.

[10] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in GitHub," *ACM Int'l Conf. on Software Eng*., 2014, pp. 356-366.

[11] R. Holmes and R.J. Walker, "Customized awareness: recommending relevant external change events," *ACM Int'l Conf. on Software Eng.*, 2010, pp. 465-474.

[12] R. Padhye, S. Mani, and V.S. Sinha, "NeedFeed: taming change notifications by modeling code relevance," *Int'l Conf. on Automated Software Eng*., 2014, pp. 665-676.

[13] G.A. Hall and J.C. Munson, "Software evolution: code delta and code churn," *J. of Sys. and Software*, vol. 54, no. 2, 2000, pp. 111-118.

[14] M. Zhou, and A. Mockus, "Developer fluency: Achieving true mastery in software projects," *Foundations of Software Eng.,* 2010, pp. 137-146.

[15] E.M. Rogers, (1995) *Diffusion of Innovations (4th ed.)*, 1995, New York, NY: The Free Press.

Biographical Sketches

James Herbsleb is a Professor in the Institute for Software Research in the School of Computer Science at Carnegie Mellon University, where he serves as Director of the PhD program in Societal Computing. His research interests lie primarily in the intersection of software, computer-supported cooperative work, and socio-technical systems, focusing on such areas as geographically distributed teams and large-scale open production communities. He holds a PhD in psychology, a JD, and an MS in computer science. Contact

him at Carnegie Mellon University, ISR–5216 Wean Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA; jdh@cs.cmu.edu.

Christian Kästner is an assistant professor in the School of Computer Science at Carnegie Mellon University. He is interested in controlling the complexity caused by variability in software systems. He develops mechanisms, languages, and tools to implement variability in a disciplined way, to detect errors, and to improve program comprehension in systems with a high amount of variability. He holds a Doctoral degree in computer science from the University of Magdeburg, Germany. Contact him at Carnegie Mellon University, ISR–5216 Wean Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA; kaestner@cs.cmu.edu.

Chris Bogart is a postdoctoral researcher at Carnegie Mellon University's Institute for Software Research. His research interests include how and why scientists go about creating software, and human-computer interaction aspects of software development. He received his PhD in computer science from Oregon State University in 2013. Contact him at Carnegie Mellon University, ISR–5216 Wean Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA; cbogart@cs.cmu.edu.

Figures

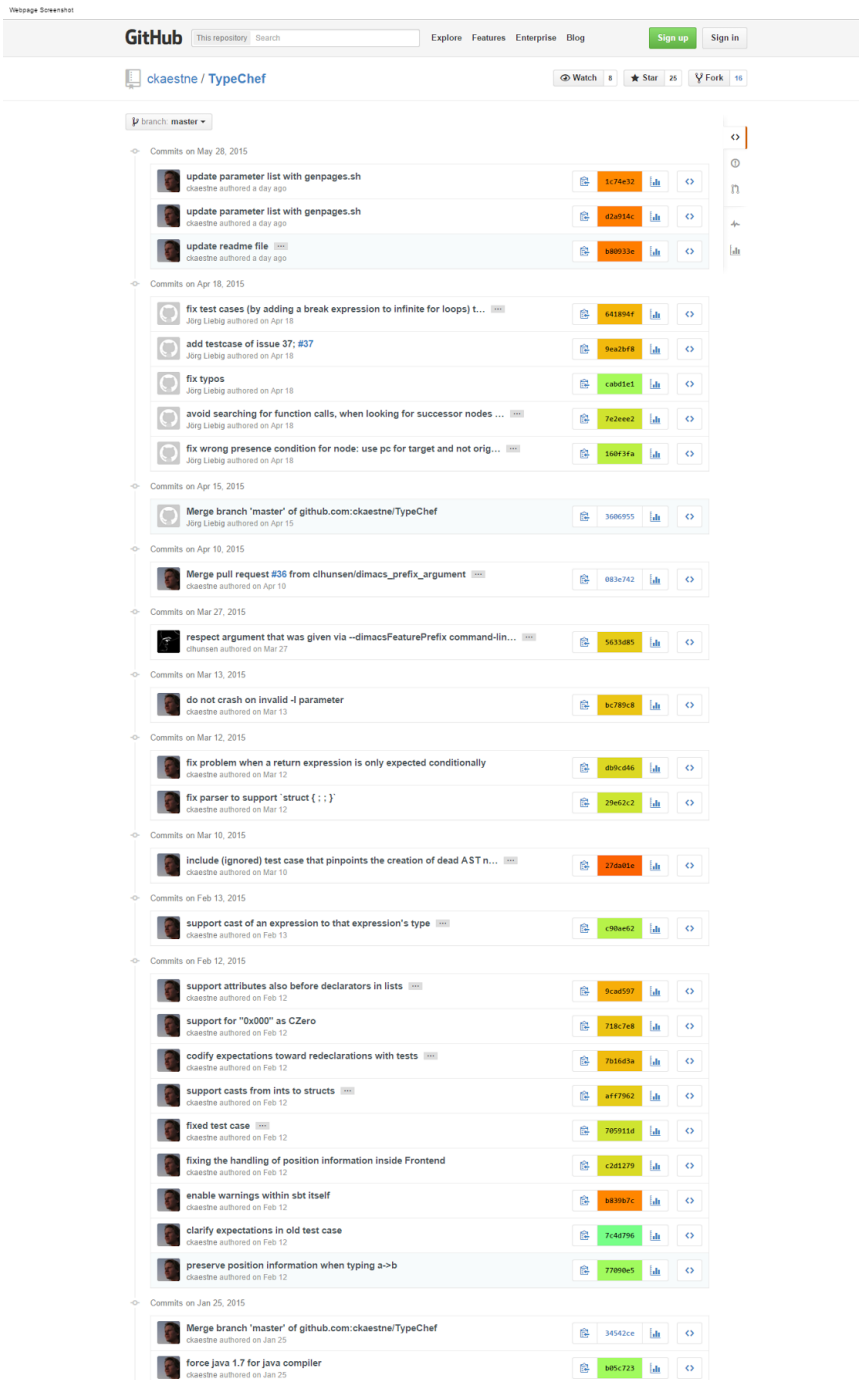

Figure 1: Color coding to highlight commits in a GitHub repository.

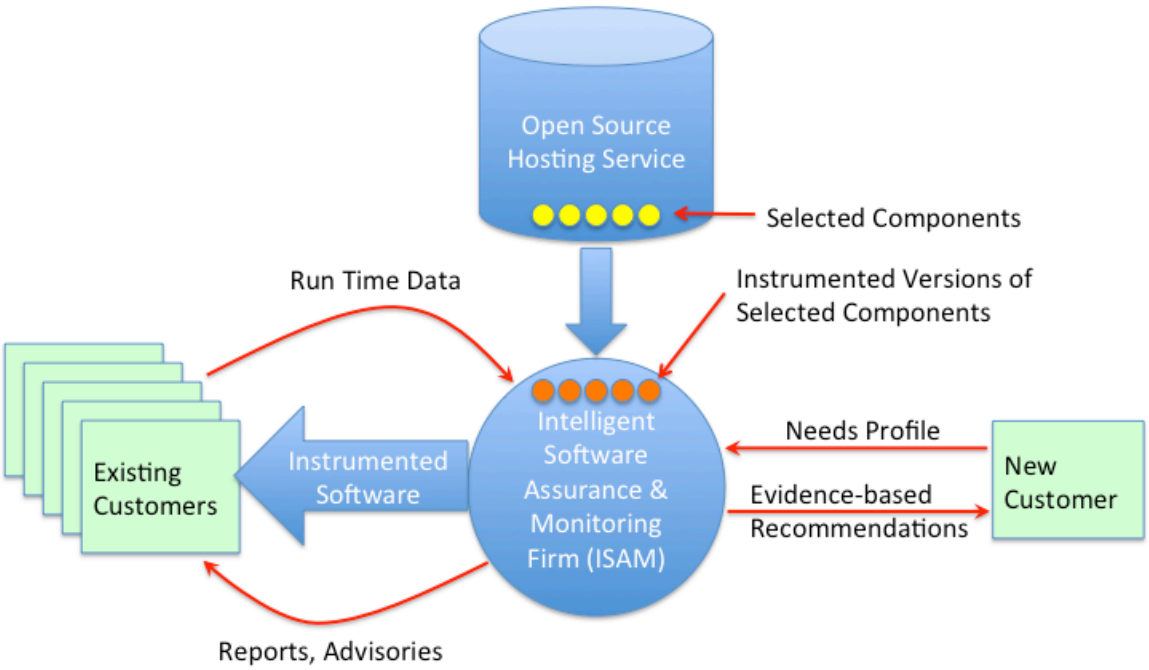

Figure 2. Intelligent Software Assurance and Monitoring firms.

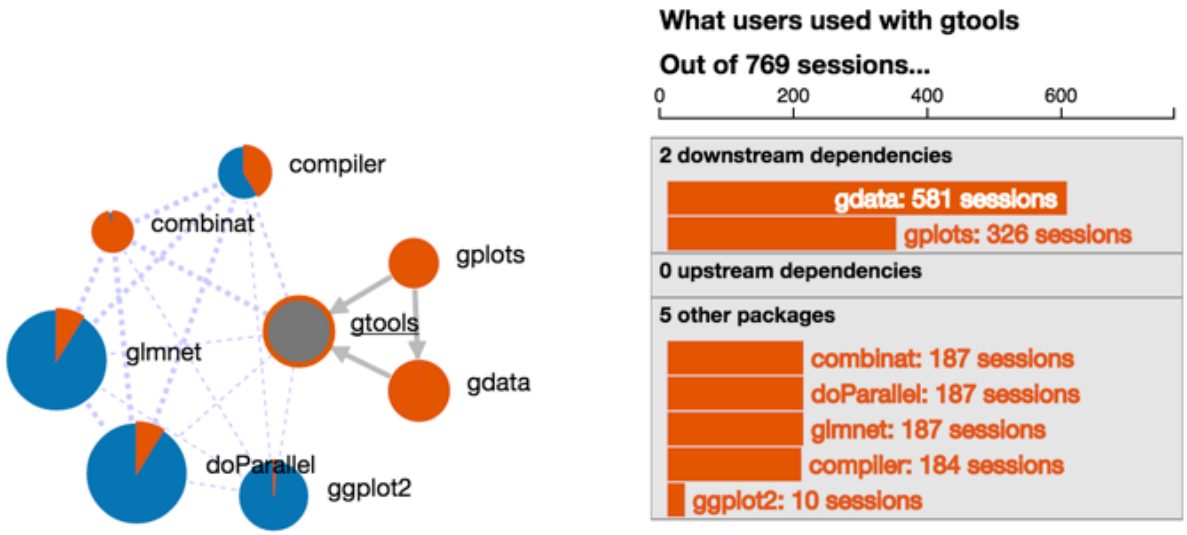

Figure 3: Software packages used together (left) and package dependencies (right).