Intelligently Transparent Software Ecosystems

James Herbsleb, Christian Kästner, and Christopher Bogart,
Carnegie Mellon University

// Intelligently transparent services will support rapid development of innovative products while helping developers manage risk and issuing them early warnings of looming failures. An infrastructure will apply analytics to data from all phases of the life cycle of open source projects. //

INNOVATIVE TRANSPARENT environments such as GitHub, LaunchPad, and BitBucket are profoundly influencing how a new generation thinks about software development. They continue a widespread trend toward openness. It's no longer surprising that individuals and commercial firms can form communities that develop and maintain valuable and freely available software assets. What's new is the combination of distributed version control and social-media features that create transparent environments that can scale up software ecosystems well into the millions of repositories and developers.[1] This trend will accelerate as large-scale data analytics adds transformative intelligent services.

Increasingly, development is a matter of selecting useful libraries, frameworks, and other components and quickly wiring them together. The result is impressive functionality produced rapidly. But this approach comes with serious risks, as developers use code without thoroughly understanding it and create new combinations with potentially dangerous interactions, and as strangers contribute hard-to-evaluate code.

Yet these risks create a business opportunity. The data that transparent environments generate could contain the fodder for novel ideas that will further speed development and help manage risk. We call this idea *intelligent transparency*.

A Hypothetical Scenario: Big Tax Data

Imagine that 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 exploit the insights and predictions this database enables.

TaxCoders, a firm supplying tax software and services, swings into action. It uses its extensive knowledge of customers' tax needs to dream up innovative applications, such as a tool that lets small businesses benchmark their tax burden, deductions, and credits against similar businesses nationally and by region. Although TaxCoders has extensive experience with tax services and enterprise data, it isn't confident it has the tools and infrastructure to engineer services using data on this national scale.

Design begins by seeking an appropriate language, Web framework, and database and appropriate visualization and data manipulation libraries. Knowing that having the quickest time to market will confer a first-mover advantage, TaxCoders engages a business we envision—an *intelligent software assurance and monitoring* (ISAM) provider. This provider delivers evidence-based

0740-7459/16/$33.00 © 2016 IEEE

component recommendations and, employing data supplied by existing customers, issues a range of early warnings as customer applications and open source components evolve.

Recommending Ensembles

The ISAM provider has extensive experience with open source projects and knows established and rising candidates in many fields. Advances in code search based on recent research in semantic search[2] and query reformulation[3] help it find a pool of candidates. Then comes the step of choosing among these candidates and finding suitable components, which used to be an arduous, highly uncertain manual process.

The ISAM provider applies analytics to its proprietary database of software-in-use to quickly identify the best candidate *ensembles*—stacks or sets of software that work well together. It maintains extensive logs of its customers' past use, which reveal a substantial history of components used together in a variety of combinations. From runtime data and test results supplied by past and current customers, the ISAM provider uses 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 its predictions to the customers' environments.

Besides evaluating the technical characteristics of components and ensembles, the ISAM provider effectively uses the detailed activity and social data available from open source hosting environments. Customers want to avoid immature and volatile components. They want well-managed projects, meaning that issues and code contributions are handled promptly and professionally. They want projects supported by a dedicated, skilled community that will continue to remain vibrant. On the basis of extensive research on success in online communities[6] and specifically of knowledge of open source communities,[7] the ISAM provider applies analytics to a variety of community variables. These variables include activities within the community and its code forks, as well as the developers' profiles and commit histories. The results are predictions of community sustainability, technical experience and proficiency, and responsiveness.

The ISAM provider identifies several potential ensembles that will serve TaxCoders's needs, but with slightly different quality and risk profiles. For example, a component in one candidate ensemble recently gave rise to a sharp increase in bug reports when a new database version was introduced, but those reports 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 have been trending upward. Stability analysis of a third ensemble, based on development observed in forks, discussions about the need for specific changes, and a growing list of related feature requests, hinted at future backward-compatibility issues. But all the recommended ensembles have risk levels acceptable to TaxCoders, which quickly chooses an ensemble.

Ongoing Vigilance

Over the next four years, TaxCoders flourishes, and the new services generate big revenue. But software evolution brings new risks, especially because key components aren't directly in TaxCoders's control. Notification services alert TaxCoders to emergent risks of community deterioration, such as

- reduced 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 refocusing of activity from the current repository to one of its forks.

Another primary risk of reliance on open source components is that ensemble elements will become incompatible. TaxCoders could get stuck on old versions, without benefiting from new features, bug fixes, or security updates. Or, it would have to manually apply changes and resolve conflicts. However, with new notification mechanisms, TaxCoders developers get tailored information whenever a component evolves so that they can react before technical debt mounts. Basing such notifications on development forks, well ahead of product releases, TaxCoders can upgrade quickly on the release date or even contact developers to negotiate the changes' direction.

In addition, the ISAM provider issues notifications of suggested changes that have attracted many comments, hinting at controversy or complexity, or of substantial new development in forks. This allows a peek into the project's future. Through timely notification, TaxCoders has had the opportunity to influence developers, avoid disruption, and advocate for the inclusion of useful features.

The biggest benefits are that TaxCoders avoids being taken by surprise and has the time to plan a response to risks that arise. In the next sections, we sketch the two major design ideas that support this scenario.

Enhancing Transparency with Analytics

We borrow Ethan Bernstein's definition of transparency: "accurate observability, of an organization's low-level activities, routines, behaviors, output, and performance."[8] Transparent environments such as GitHub let anyone

- 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
- find detailed information about people, their activities, and their social connections.

Transparency makes it much easier for developers to find useful code supported by viable communities and to monitor code they're using to detect changes that create problems or opportunities.[9] On the downside, transparency can present developers with overwhelming amounts of information.

Characteristics developers care about, such as a component's quality attributes, often aren't 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 the software. This practice is currently imperfect and time-consuming.

Intelligently transparent environments will streamline and expand these capabilities. Inferences' speed and accuracy will be enhanced by computational agents that quickly summarize the information developers want to see. For example, when choosing among candidate libraries or frameworks, developers look for signals that the project

- is "alive,"
- has a group of people committed to it,
- evolves without frequent disruptions to downstream projects,
- is skillfully managed, and
- has been well received by the community.

A variety of signals are useful for assessing these hidden qualities.[10] Such signals include

- the commit velocity,
- the diversity of frequent committers,
- the project's number of stars or likes,
- the number of test cases,
- the history of continuous-integration results, and
- a history of issues and pull requests being quickly addressed.

It's laborious for developers to manually examine all of the many signals they wish to see to assess and compare projects' suitability. However, a computational agent can acquire the data and present it in a terse format, such as a dashboard visualization. Many people are experimenting with such visualizations (for a collection of visualizations, see GitHub Visualizer; http://ghv.artzub.com). Such tools can transform tedious tasks into tasks that quick perusal of a visual display can resolve.

Customers want well-managed projects, meaning that issues and code contributions are handled promptly and professionally.

Future analytics research will also bring novel forms of information that help navigate the challenges of evolving components. Intelligent transparency can help developers discover interesting changes among a sea of constantly evolving projects. It can also help identify changes that encourage further actions, such as breaking interface changes or new useful features. Tech Angels' Gemnasium (https://gemnasium.com) is in this spirit. It notifies users of updates and security vulnerabilities in any of their dependencies so that they can take appropriate action without constantly monitoring all changes in all their dependencies. In our scenario, such awareness functionality lets TaxCoders avoid disruptions and failures as the open source projects on which it depends evolve.

Another problem being addressed is information overload. Approaches such as YooHoo[11] and NeedFeed[12] have shown that developers can significantly reduce the notification clutter in systems such as GitHub by using simple mechanisms to filter important messages. For example, YooHoo identifies those changes that break binary compatibility in library code—a mere 7 percent of all changes. NeedFeed uses code ownership and past code changes as simple heuristics to identify relevant changes.

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

Different facets of stability can be inferred from many sources. For example, developers can simply declare the intent that an interface won't change or will remain backward compatible. In addition, developers can observe historical stability, both averages and trends, by mining the repository.[13]

Furthermore, by analyzing dependencies in an ecosystem, developers can derive signals about the context in which a component is being used. For example, components that are used by other components that are intended to be stable should evolve more conservatively. Use of components not intended to be stable indirectly indicates that the using component will likely change. A situation in which components that are intended to be stable use historically unstable components 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. Such information 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.

As analytic techniques are refined and very large-scale datasets become available, research will push intelligent transparency beyond filtering and stability, to infer developer intent for a range of use cases from a range of sources. These analyses will produce additional notifications and reports that ISAM can provide in the longer run, enabling firms like TaxCoders to respond proactively.

Examples of such information include

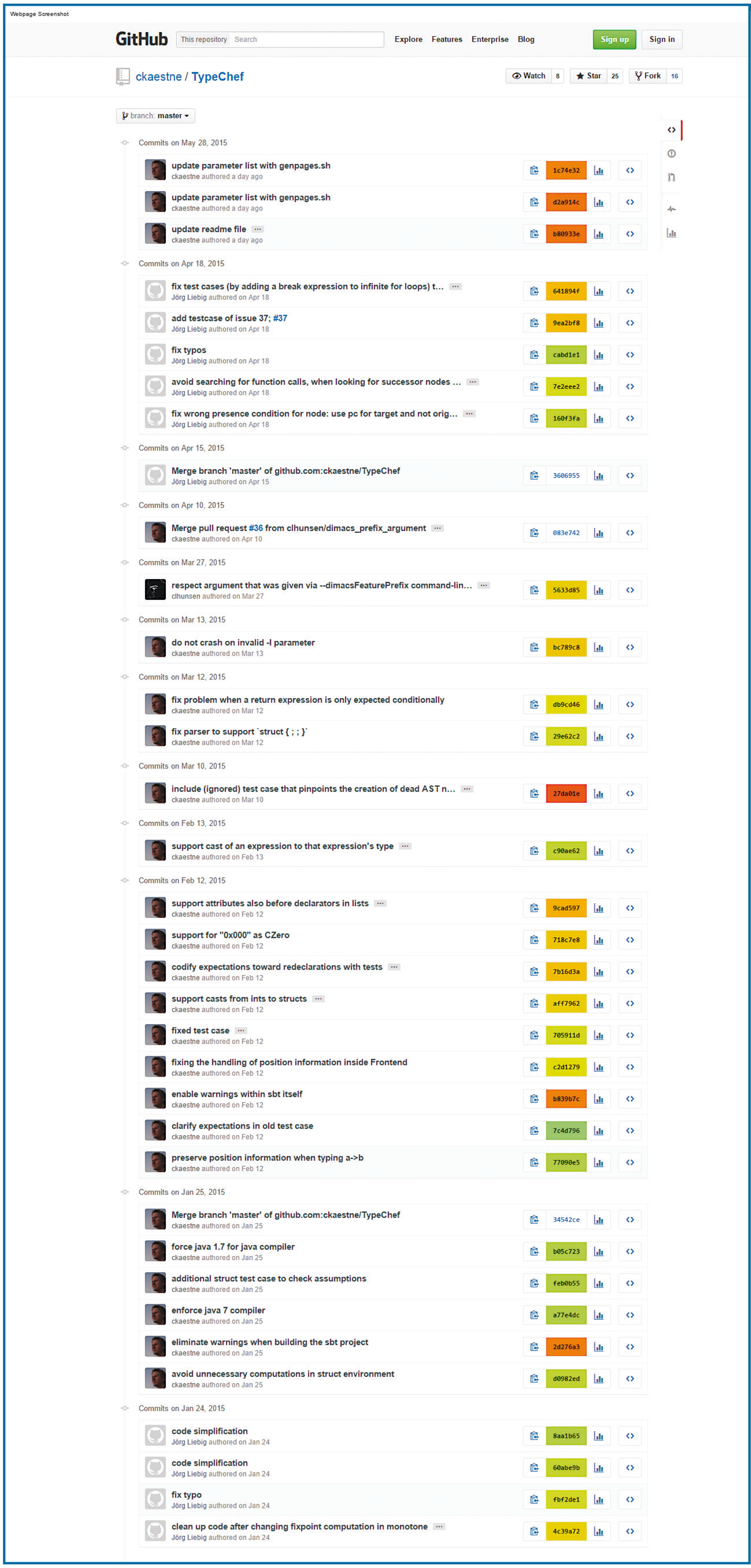
- indicators and signals to identify commits deserving more attention or review—for example, considering the developer's experience and the centrality of the code being changed[14] (see Figure 1, in which the dark orange highlighting indicates such commits);
- the probability that a specific pull request will be accepted;
- a project's likely future activity level (new features and bug fixes);
- overdependence on a few core developers' continued contributions; and
- a summary, derived from package managers and clone detection, of all uses of project code, broken down by user type and key use attributes, 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. Although intelligence useful for selecting components has its basis in analysis of repositories and developer activity, critical monitoring services will also use runtime data.

Analyzing Runtime Data to Provide Monitoring

Using free software grown in the wild exposes businesses to unpredictable failures and security threats. ISAM providers will also tackle these adoption risks. Some ISAM-like services already exist for internal use for proprietary software, such as Apple's and Microsoft's OSs, and some consultants have amassed a lot of experience in particular domains. Yet such services aren't available at sufficient detail or breadth for monitoring the ecosystems of diverse open-source software on which businesses often depend.

Fortunately, some software component users are willing to accept more risk than others. Open source projects presumably follow a traditional adoption curve.[15] The first to take them up will have high risk tolerance coupled with a compelling business need for novel functionality. For example, these adopters might be writing mobile apps that don't touch sensitive data but require novel computation. If a new component has features that attract early adopters, the attention helps ensure that bugs will be discovered



ckaestne / TypeChef

branch: master

Commits on May 28, 2015

update parameter list with genpages.sh
ckaestne authored a day ago — 1c74e32

update parameter list with genpages.sh
ckaestne authored a day ago — d2a914c

update readme file …
ckaestne authored a day ago — b80933e

Commits on Apr 18, 2015

fix test cases (by adding a break expression to infinite for loops) t… …
Jörg Liebig authored on Apr 18 — 641894f

add testcase of issue 37; #37
Jörg Liebig authored on Apr 18 — 9ea2bf8

fix typos
Jörg Liebig authored on Apr 18 — cabd1e1

avoid searching for function calls, when looking for successor nodes … …
Jörg Liebig authored on Apr 18 — 7e2eee2

fix wrong presence condition for node: use pc for target and not orig… …
Jörg Liebig authored on Apr 18 — 160f3fa

Commits on Apr 15, 2015

Merge branch 'master' of github.com:ckaestne/TypeChef
Jörg Liebig authored on Apr 15 — 3606955

Commits on Apr 10, 2015

Merge pull request #36 from clhunsen/dimacs_prefix_argument …
ckaestne authored on Apr 10 — 083e742

Commits on Mar 27, 2015

respect argument that was given via --dimacsFeaturePrefix command-lin… …
clhunsen authored on Mar 27 — 5633d85

Commits on Mar 13, 2015

do not crash on invalid -I parameter
ckaestne authored on Mar 13 — bc789c8

Commits on Mar 12, 2015

fix problem when a return expression is only expected conditionally
ckaestne authored on Mar 12 — db9cd46

fix parser to support `struct { ; ; }`
ckaestne authored on Mar 12 — 29e62c2

Commits on Mar 10, 2015

include (ignored) test case that pinpoints the creation of dead AST n… …
ckaestne authored on Mar 10 — 27da01e

Commits on Feb 13, 2015

support cast of an expression to that expression's type …
ckaestne authored on Feb 13 — c90ae62

Commits on Feb 12, 2015

support attributes also before declarators in lists …
ckaestne authored on Feb 12 — 9cad597

support for "0x000" as CZero
ckaestne authored on Feb 12 — 718c7e8

codify expectations toward redeclarations with tests …
ckaestne authored on Feb 12 — 7b16d3a

support casts from ints to structs …
ckaestne authored on Feb 12 — aff7962

fixed test case …
ckaestne authored on Feb 12 — 705911d

fixing the handling of position information inside Frontend
ckaestne authored on Feb 12 — c2d1279

enable warnings within sbt itself
ckaestne authored on Feb 12 — b839b7c

clarify expectations in old test case
ckaestne authored on Feb 12 — 7c4d796

preserve position information when typing a->b
ckaestne authored on Feb 12 — 77090e5

Commits on Jan 25, 2015

Merge branch 'master' of github.com:ckaestne/TypeChef
ckaestne authored on Jan 25 — 34542ce

force java 1.7 for java compiler
ckaestne authored on Jan 25 — b05c723

additional struct test case to check assumptions
ckaestne authored on Jan 25 — feb0b55

enforce java 7 compiler
ckaestne authored on Jan 25 — a77e4dc

eliminate warnings when building the sbt project
ckaestne authored on Jan 25 — 2d276a3

avoid unnecessary computations in struct environment
ckaestne authored on Jan 25 — d0982ed

Commits on Jan 24, 2015

code simplification
Jörg Liebig authored on Jan 24 — 8aa1b65

code simplification
Jörg Liebig authored on Jan 24 — 60abe9b

fix typo
Jörg Liebig authored on Jan 24 — fbf2de1

clean up code after changing fixpoint computation in monotone …
Jörg Liebig authored on Jan 24 — 4c39a72

FIGURE 1. Color coding to highlight commits in a GitHub repository. Red highlighting indicates commits that deserve more attention or review. (Source: GitHub; used with permission.)

and fixed quickly. If developers can observe that a component is widely used in diverse contexts and has become stable, they'll regard it as trustworthy.

Earlier adopters act, in effect, as field testers for those who are more risk averse, but this has an 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 "testing" has been carried out and in which domains, platforms, and configurations. ISAM providers will supply this critical information.

ISAM providers will monitor the software's deployed usage by supplying their customers—with their knowledge and consent, under appropriate confidentiality arrangements—with instrumented versions of the software packages the providers thinks their customers will want. These versions will send usage data back to the providers (see Figure 2). This data will capture what version of what software is deployed, in what hardware and software environment, how often, how it performs, and the circumstances of failures. It will even provide a monitoring platform for the providers to capture domain-specific statistics of interest. For webserver 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 the algorithm. For an IDE, it might record user settings and installed plug-ins.

Customers will be able to visualize the runtime data in several ways; Figure 3 shows a prototype. The graph view (see Figure 3a) shows which other packages the gtools package depends on (the solid lines)

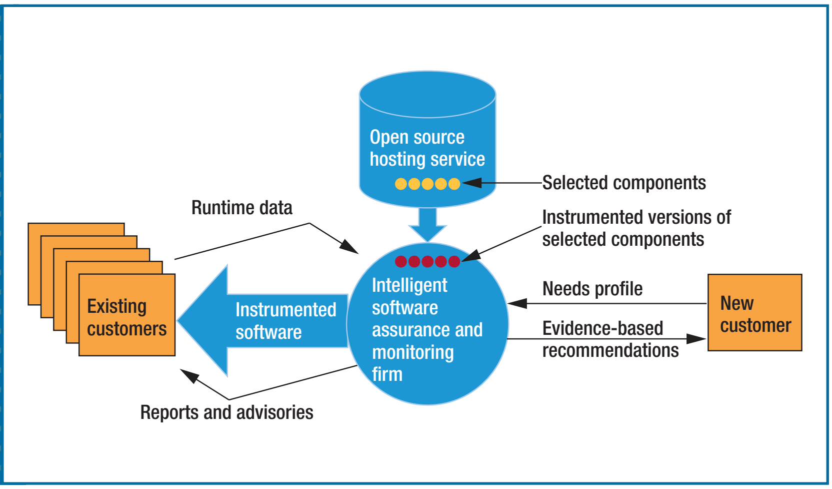

Open source hosting service
Selected components
Instrumented versions of selected components
Runtime data
Existing customers
Instrumented software
Intelligent software assurance and monitoring firm
Needs profile
New customer
Evidence-based recommendations
Reports and advisories

FIGURE 2. How intelligent software assurance and monitoring (ISAM) providers will function. ISAM providers will deliver evidence-based component recommendations. Using data supplied by existing customers, they'll also issue a range of early warnings as customer applications and open source components evolve.

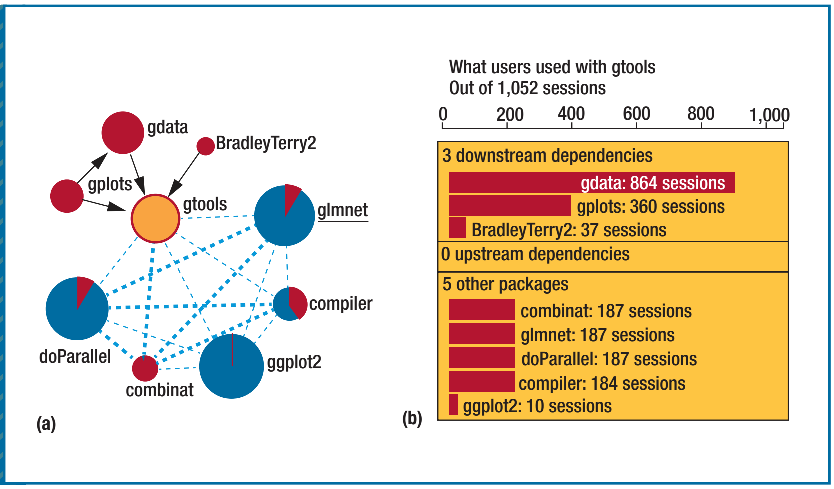


FIGURE 3. Visualizing runtime data. (a) Software packages used together. Solid lines indicate which packages gtools depends on; dotted lines indicate which packages gtools is used with. (b) Package dependencies and the frequency with which the packages are called in a set of runs.

or is frequently used with (the dotted lines). The bar charts (see Figure 3b) show upstream and downstream dependencies and the frequency with which the packages are called in a set of runs.

Of course, runtime data poses potential privacy issues, but they might well be surmountable (see the sidebar).

The advantages for customers and ISAM providers will be substantial. For customers, monitoring could provide early warning or ideally let them avoid most failures and downtime. For example, if a provider detected a failure at one customer site, it 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, ISAM providers will help them establish priorities. The providers could also contact customers running a similar configuration, giving them a detailed warning of a possible failure, letting them prepare backup plans or workarounds.

In addition, ISAM providers will develop proprietary algorithms that use the vast store of runtime data. They'll also make custom assessment reports available to their customers, tailored to their tolerance for risk and priorities among quality attributes. Customers considering using a given component or ensemble would benefit from the accumulated experience of other similarly situated users. These reports will be updated frequently because as any given project gets used more, it produces more data and becomes more mature, stable, and secure. Customers receiving the reports will be in a much better position to create a portfolio of software assets that balances their need for innovation and their risk tolerance.

Finally, ISAMs could make basic usage data public to benefit the overall ecosystem. Wider use of a software project compared to its competitors can further increase its use, 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 developers to fix bugs and offer new functionality. The extra attention accelerates evolution, initiating a virtuous cycle of wider adoption and rapid improvement.

Open software ecosystems are a rich source of libraries, frameworks, and code fragments that can reduce programmer effort and accelerate development. Yet they require time and effort to evaluate, and they expose users to the risks of poor selection. They also introduce the uncertainties of becoming dependent on code whose evolution someone else controls.

By applying analytics to the detailed activity and communication traces in transparent environments and by acquiring and analyzing ongoing runtime data, ISAM providers will support well-informed choices. They'll also issue timely warnings that help developers either negotiate for favorable changes in upstream software or prepare to migrate to alternative components. As developers become armed with solid, contextualized empirical data and analytics, software quality will improve, and development will become faster, cheaper, and more predictable.

RUNTIME MONITORING

Software users are rightfully vigilant about software that communicates their activities. But there's an important place for open source software monitoring when it is done with the users' knowledge and consent and is for their benefit. For example, XALT tracks the use of both open source and commercial software on supercomputers, through shell wrappers that users can choose to use.[1] It captures library and resource use without special instrumentation of the individual packages. The Condor distributed batch system reported basic data of its own usage on 50,000 CPUs over 1,000 sites.[2] Other examples include usage statistics such as the Debian Popularity Contest (http://popcon.debian.org) and crash reporting as in Ubuntu, Mozilla,[3] and Microsoft products.[4]

In our envisioned scenario (see the main article), *intelligent software assurance and monitoring* (ISAM) customers agree to use instrumented software versions and provide their runtime data and test results. This agreement occurs under suitable confidentiality arrangements, as part of a contract that lets ISAM provide monitoring that wouldn't be possible without the usage data. Many software firms will likely be willing to accept runtime monitoring to reap ISAM's benefits. Many of their customers will likely consent, as many users do now for crash reporting and quality improvement.

References

1. M. Fahey, R. McLay, and K. Agrawal, *XALT Design and Installation Manual*, 2015; http://sourceforge.net/projects/xalt/files.
2. D. Thain, T. Tannenbaum, and M. Livny, "How to Measure a Large Open-Source Distributed System," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 15, 2006, pp. 1989–2019.
3. L. Thomson, "Socorro: Mozilla's Crash Reporting System," blog, Mozilla, 19 May 2010; http://blog.mozilla.com/webdev/2010/05/19/socorro-mozilla-crash-reports.
4. K. Kinshumann et al., "Debugging in the (Very) Large: Ten Years of Implementation and Experience," *Comm. ACM*, vol. 54, no. 7, 2011, pp. 111—116.

Acknowledgments

We gratefully acknowledge support from US National Science Foundation grants 1064209, 1322278, and 1111750 and an Alfred P. Sloan Foundation grant.

References

1. L. Dabbish et al., "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. Software Eng. and Methodology*, vol. 23, no. 3, 2014, article 26.
3. L. Martie, T.D. LaToza, and A. van der Hoek, "CodeExchange: Supporting Reformulation of Code Queries in Context," to be published in *Proc. 30th Int'l Conf. Automated Software Eng.*, 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. 27th Int'l Conf. Software Eng.* (ICSE 05), 2005, pp. 342–351.
6. R.E. Kraut and P. Resnick, eds., *Building Successful Online Communities: Evidence-Based Social Design*, MIT Press, 2011.
7. K. Crowston et al., "Free/Libre Open-Source Software Development: What We Know and What We Do Not

ABOUT THE AUTHORS

JAMES HERBSLEB is a professor in the Institute for Software Research in Carnegie Mellon University's School of Computer Science, where he directs the Societal Computing PhD program. 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. Herbsleb received a PhD in psychology from the University of Nebraska. Contact him at jdh@cs.cmu.edu.

CHRISTIAN KÄSTNER is an assistant professor in Carnegie Mellon University's School of Computer Science. He's interested in controlling the complexity caused by software system variability. He develops mechanisms, languages, and tools to implement variability in a disciplined way, to detect errors, and to improve program comprehension in highly variable systems. Kästner received a doctorate in computer science from the University of Magdeburg. Contact him at kaestner@cs.cmu.edu.

CHRISTOPHER BOGART is a postdoctoral researcher at Carnegie Mellon University's Institute for Software Research. His research interests include how and why scientists create software, and software development's human-computer-interaction aspects. Bogart received a PhD in computer science from Oregon State University. Contact him at cbogart@cs.cmu.edu.

Know," *ACM Computing Surveys*, vol. 44, no. 2, 2012, article 7.

8. E.S. Bernstein, "The Transparency Paradox: A Role for Privacy in Organizational Learning and Operational Control," *Administrative Science Q.*, vol. 57, no. 2, 2012, pp. 181–216.
9. L. Dabbish et al., "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," *Proc. ACM 2012 Conf. Computer 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," *Proc. 36th Int'l Conf. Software Eng.* (ICSE 14), 2014, pp. 356–366.
11. R. Holmes and R.J. Walker, "Customized Awareness: Recommending Relevant External Change Events," *Proc. 32nd ACM/IEEE ACM Int'l Conf. Software Eng.* (ICSE 10), 2010, pp. 465–474.
12. R. Padhye, S. Mani, and V.S. Sinha, "NeedFeed: Taming Change Notifications by Modeling Code Relevance," *Proc. 29th ACM/IEEE Int'l Conf. Automated Software Eng.* (ASE 14), 2014, pp. 665–676.
13. G.A. Hall and J.C. Munson, "Software Evolution: Code Delta and Code Churn," *J. Systems and Software*, vol. 54, no. 2, 2000, pp. 111–118.
14. M. Zhou and A. Mockus, "Developer Fluency: Achieving True Mastery in Software Projects," *Proc. 18th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.* (FSE 10), 2010, pp. 137–146.
15. E.M. Rogers, *Diffusion of Innovations*, 4th ed., 1995, Free Press.