Developer Dashboards: The Need for Qualitative Analytics

Olga Baysal, Reid Holmes, and Michael W. Godfrey, University of Waterloo

// Interviews with Mozilla developers show a need for qualitative dashboards designed to improve developers' situational awareness by providing task tracking and prioritizing capabilities, presenting insights on the workloads of others, listing individual issues, and providing custom views to help manage their workloads while performing day-to-day development tasks. //

MANY ORGANIZATIONS HAVE adopted data-driven decision-making processes and technologies. Embedding analytics into an organization's culture can enable a competitive advantage.[1] Analytic approaches strive to provide actionable real-time insights and can be both quantitative and qualitative in nature. Whereas quantitative analytics can highlight high-level data trends, qualitative analytics enable real-time decision making for day-to-day tasks.

Most analytics approaches focus on quantitative historical analysis, often using chart-like dashboards (see examples in Figure 1). These dashboards are often geared toward helping project managers monitor and measure performance—for example, "to provide module owners with a reliable tool with which to more effective[ly] manage their part of the community."[2]

David Eaves, a member of the Mozilla Metrics team who worked on community contribution dashboards (a quantitative analytic tool), states, "We wanted to create a dashboard that would allow us to identify some broader trends in the Mozilla Community, as well as provide tangible, useful data to module owners particularly around identifying contributors who might be participating less frequently."[2]

Figure 1a demonstrates quantitatively oriented dashboards for the IBM Jazz development environment (https://jazz.net/products/rational-quality-manager) and Figure 1b for the Mozilla Metrics project (https://metrics.mozilla.com). The Jazz quality dashboard provides high-level charts describing various project statistics such as how issues have been open and closed (bottom left) and how various test cases execute (bottom middle). The Mozilla dashboard provides a quantitative description of various community contributions, including a graph at the top showing where these contributions were made and a table below that describes per-developer statistics. Although both of these dashboards effectively convey information about specific aspects of the project, the information is geared more toward managers than to individual developers as they perform their daily tasks (for additional information, see the "Related Work in Software Development Awareness" sidebar).

Our own study suggests that current dashboards poorly support developers' day-to-day development tasks.[3] As Ehsan Akhgari, a Mozilla developer, says, "What I really want

0740-7459/13/$31.00 © 2013 IEEE

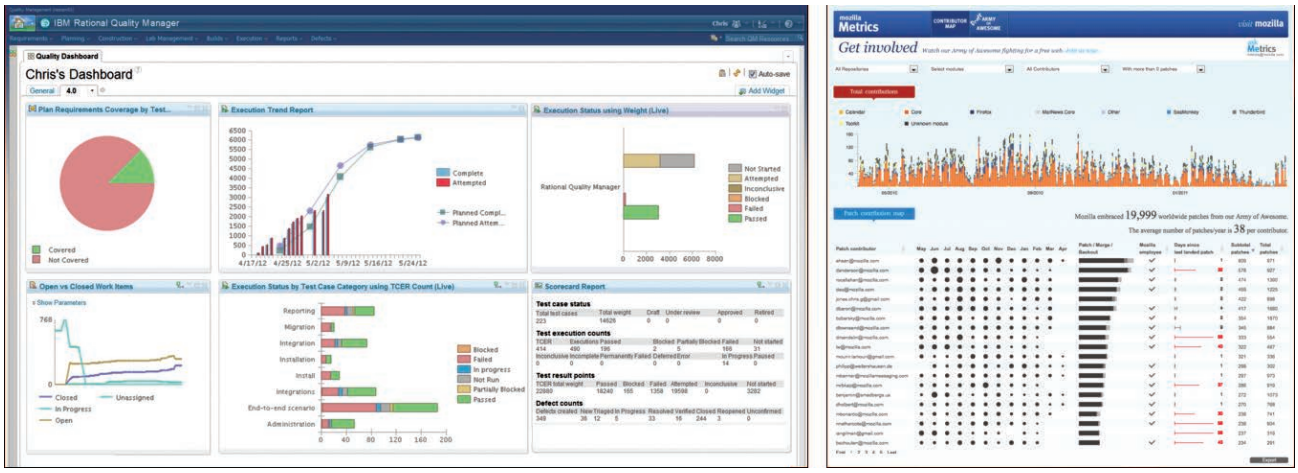

FIGURE 1. Quantitative dashboards: (a) IBM Rational Quality Manager and (b) Mozilla Metrics. These dashboards provide aggregate views of project activity that are most suitable for project managers.

is a dashboard to show the important stuff I need to see—review status on the dashboard, assigned bug with a new comment or change since the last time I looked at it, bugs that have a certain status could go on the dashboard, etc."[4]

By definition, a qualitative property is described rather than measured. Whereas quantitative dashboards provide statistical summaries of various development metrics, qualitative dashboards emphasize the attributes and relationships of a set of artifacts that are of interest to the developer. Thus, qualitative dashboards provide developers with the means to define, organize, and monitor their personal tasks and activities. Unlike quantitative dashboards, which address aggregate developer questions such as, "How many bugs are pending on me?" or "How well did I perform last quarter?," qualitative dashboards provide insights into the specific items developers are working on: "Who has the ball?," "What do we do next?," and "What has changed since the last time I looked at [this bug]?" Regarding the issues they're interested in, developers want the ability to get information based on what has changed since the last time they looked at them. Being able to keep abreast of the volume of changes taking place on active development teams can be challenging; by helping developers focus on the evolution of their issues, along with those in which they're interested, they can better prioritize their own tasks.

Research Study

To understand how developers engage and interact with the Bugzilla issue-tracking system, we performed a qualitative study of interviews with 20 core Mozilla developers.[4] The study captures developers' insights into the strengths, weaknesses, and possible future enhancements of the Bugzilla platform, the primary collaboration platform for Mozilla developers. We applied a grounded theory methodology on the interview transcripts; we had no predefined themes or categories. We first split the 20 interview transcripts into 1,213 individual comments. We performed two rounds of independent card sorting and reported the inter-coder reliability (degree of agreement) to ensure the card sort's integrity. We calculated Krippendorff's alpha (α), a reliability coefficient to measure the agreement among coders achieved when categorizing textual units into groups, categories, and themes. We achieved good inter-coder reliability with $\alpha = 0.865$ ($\alpha = 1$ indicates maximum reliability; $\alpha = 0$ indicates the absence of reliability). On average, our two coders agreed on the coding of the content 98.5 percent of the time.

Four high-level themes emerged from the data, among which situational awareness emerged as one of the major missing pieces developers requested from Bugzilla (19 of the 20 developers provided 208 quotes in support of issues surrounding situational

RELATED WORK IN SOFTWARE DEVELOPMENT AWARENESS

The research community has recognized the problem of maintaining developer awareness on projects. Mauro Cherubini and his colleagues looked at how and why developers use drawing during software development.[1] Christoph Treude and Margaret-Anne Storey investigated the role of awareness tools such as Jazz dashboards and feeds in supporting development activities.[2] Thomas Fritz and Gail Murphy studied how developers assess the relevancy of these feeds to help users deal with the vast amount of information flowing to them in this form.[3]

Several tools assist developers with daily tasks and activities. FASTDash offers an interactive visualization to enhance team awareness during collaborative programming tasks.[4] The workspace awareness tool Palantir follows a similar approach by providing insight into workspaces of other developers, in particular, regarding artifact changes.[5] Mylyn is a task management tool for Eclipse that integrates various repositories such as GitHub, Bugzilla, JIRA, and so on.[6] Mylyn offers a task-focused interface to developers to ease activities such as searching, navigation, multitasking, planning, and sharing expertise. Yoohoo monitors the development activity from all of the projects the developer's code depends on (such as libraries and external systems); it then creates customized notifications highlighting changes that are likely to impact the code the developer is responsible for.[7] Similarly, Crystal increases developer awareness of version control conflicts during collaborative project development.[8]

Although these tools provide insight into the current collaborative activities on a project, there is still a need for a custom view of the project to help developers maintain awareness of their own working context.

References

1. M. Cherubini et al., "Let's Go to the Whiteboard: How and Why Software Developers Use Drawings," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 07), ACM, 2007, pp. 557–566.
2. C. Treude and M.-A. Storey, "Awareness 2.0: Staying Aware of Projects, Developers and Tasks Using Dashboards and Feeds," *Proc. 32nd ACM/IEEE Int'l Conf. Software Eng.* (ICSE 10), vol. 1, ACM, 2010, pp. 365–374.
3. T. Fritz and G.C. Murphy, "Determining Relevancy: How Software Developers Determine Relevant Information in Feeds," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 11), ACM, 2011, pp. 1827–1830.
4. J.T. Biehl et al., "FASTdash: A Visual Dashboard for Fostering Awareness in Software Teams," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 07), ACM, 2007, pp. 1313–1322.
5. A. Sarma, Z. Noroozi, and A.V.D. Hoek, "Palantir: Raising Awareness among Configuration Management Workspaces," *Proc. 25th Int'l Conf. Software Eng.* (ICSE 03), IEEE CS, 2003, pp. 444–454.
6. M. Kersten and G.C. Murphy, "Using Task Context to Improve Programmer Productivity," *Proc. ACM-SIGSOFT Intl. Symp. Foundations of Software Eng.* (FSE 06), ACM, 2006, pp. 1–11.
7. R. Holmes and R.J. Walker, "Customized Awareness: Recommending Relevant External Change Events," *Proc. ACM/IEEE Int'l Conf. Software Eng.* (ICSE 10), vol. 1, ACM, 2010, pp. 465–474.
8. Y. Brun et al., "Crystal: Precise and Unobtrusive Conflict Warnings," *Proc. 19th ACM/SIGSOFT Symp. and 13th European Conf. Foundations of Software Eng.* (ESEC-FSE 11), ACM, 2011, pp. 444–447.

awareness). The complete results of the qualitative study are available online.[3]

Eighteen developers reported challenges maintaining awareness of the status of their own issues: "[I maintain a] gigantic spreadsheet of bugs I am looking at. It would be useful to know how the bugs have changed since I last looked at [the bugs] to track if any work was done [on them]."

In addition to their own issues, 15 developers expressed interest in the ability to unobtrusively observe the evolution of other issues without being forced to take an active role: "You don't want to cc yourself on every bug you triage."

Twelve developers also expressed a desire to easily gain an understanding of their colleagues' workloads, for instance, when requesting code reviews:

> *If you could see how many reviews are pending on a person on that [review] list, this would be a better way to load balance reviewers. It would be good to have an easy way to click on the name in some way and jump on their review queue to see if they have a lot of easy or hard issues to look at.*

Finally, 12 developers found it challenging to assess other developers' roles in the Mozilla organization:

> *People profiles—you should be able to know more about them, how long they have been in the system, what is their ranking, are they module owners or peers. We need to know who we are talking to. We need some way to figure out who you are so that we can treat each other better. We depend on people so much and Bugzilla is all about bugs, not people.*

We believe that the information that developers seek is often available in the tools they currently use, but the data isn't accessible in a format amenable to the tasks they're trying to perform. Just as we can generate quantitative

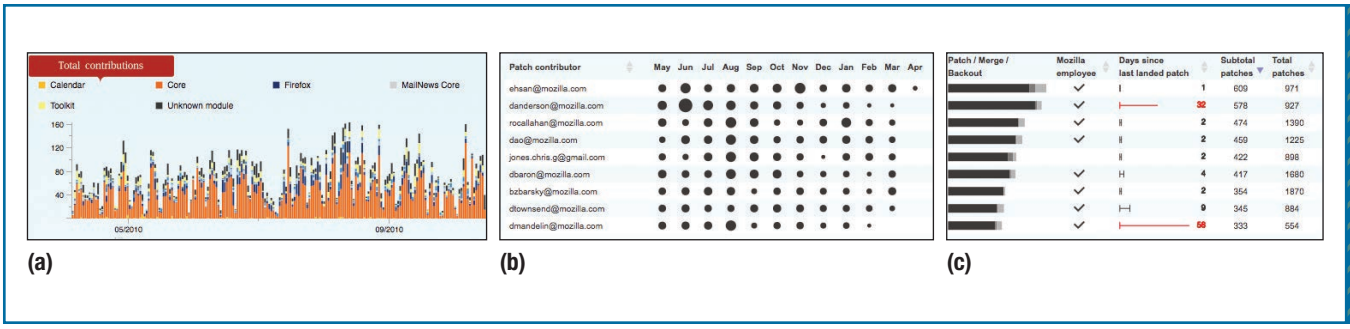

FIGURE 2. Mozilla Community Metrics, a view designed to aggregate trends regarding Mozilla community involvement: (a) total contributions, (b) patch contributions per developer, and (c) developer effectiveness.

dashboards from a project's issue tracker, we can also generate developer-specific qualitative dashboards.

Situational Awareness

Situational awareness is a term from cognitive psychology referring to a state of mind in which a person is aware of the elements of his or her immediate environment, has an understanding as to the environment's greater meaning, and can anticipate (or plan to change) these elements in the near future.[5] The term is used in engineering, for example, to describe how air traffic controllers work as they track and route air traffic. It is also an apt description of how software developers must maintain awareness of what's happening on their projects as they manage a constant flow of information and react accordingly.

Developers often find themselves trying to identify the status of an issue (What is the issue waiting on? Who is working on it? What are the workloads of others? Who is the best person to review a patch?) as well as trying to track their own tasks (Which bugs do I need to triage, fix, review, or follow up on? Which issue should I work on next?).

Our study suggests that supplementing quantitative dashboards with more developer-specific qualitative data can improve developers' situational awareness of their working context. This awareness will enable developers to keep better track of the ever-increasing number of issues involved in complex software systems. From our own experience in analyzing the data available in Bugzilla, we also believe the data required to build developer-specific qualitative dashboards already exists; much like quantitative analytics, the data just needs to be extracted, analyzed, and presented in an appropriate format so developers can easily make use of it.

Qualitative Dashboards: Task-Oriented Views

Qualitative dashboards can enhance a developer's daily activities, such as issue tracking and prioritization, patch submission, personal workload management, colleague workload estimation, maintaining lists of issues that need follow-up, and so on. Although most of the data necessary to generate qualitative dashboards is present within the Bugzilla database, it's not easily accessible for several reasons. First, the Bugzilla user interface is notoriously unintuitive and wasn't designed to support process management, presenting too much information to the user and little direction, as noted by seven developers from our study: "The Bugzilla interface is bad; too many fields." Second, Bugzilla's slow performance hinders real-time data exploration: "The speed of Bugzilla is the major issue"; "Bugzilla is too slow; this is wasting a lot of time—very frustrating." Third, developers are often unable to correctly formulate queries to access and correlate various pieces of metadata: "Running searches on Bugzilla is kind of scary sometimes"; "Querying in Bugzilla is hard; [the user] has to spend a few minutes to figure out how to do the query … [there's] no good way to query certain information."

Mozilla has applied quantitative analytics through two initiatives: Community Management Metrics provides insights on the evolution of the community contributions,[2] and Bugzilla Anthropology analyzes project performance.[4] These initiatives have created a series of dashboards that use historical information to monitor community contributions (see Figure 2) or to measure and track trends in bug-fixing efforts (see Figure 3). These dashboards support tasks such as monitoring patch contributions and identifying bug trends (status- or priority-based). However, they're tailored toward project managers and their activities; they don't provide developers with any useful support for their typical daily tasks.

Contrasting Qualitative and Quantitative Dashboards

For the Mozilla team, qualitative analytics can generate developer-specific views of the Bugzilla repository. The qualitative approach filters important

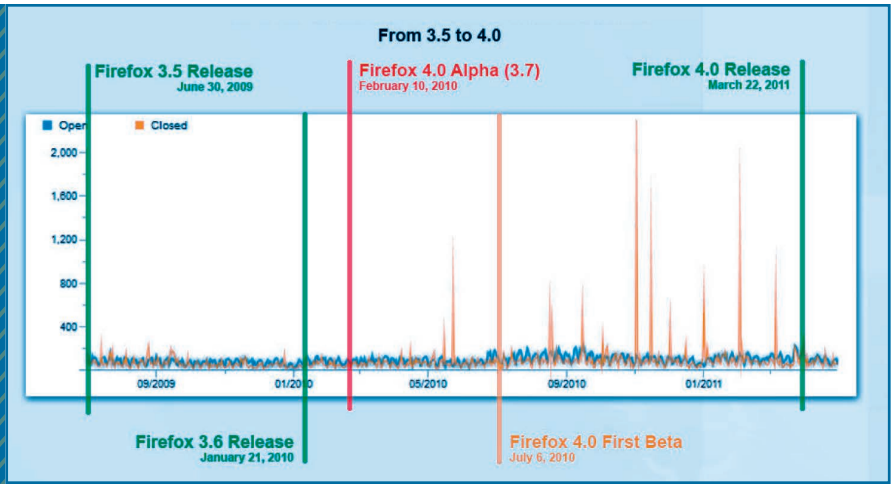

FIGURE 3. Daily open versus closed bug changes in Firefox over time.

and relevant information from the repository and presents it to the developers in support of common tasks such as bug fixing, feature implementation, code review, and triage. Qualitative analytics let users access data from the existing issue-tracking system to enhance Bugzilla with the means to increase developers' situational awareness of the project.

We now describe four tasks that one of Mozilla's current quantitative dashboards provides support for and outline how these fall short in helping developers with their daily tasks.

Assessing Community Effectiveness and Evolution

The current Mozilla community management dashboard provides insight into community patch contributions. Figure 2a shows the number of contributions submitted to various Mozilla modules; contributors can be sorted according to their involvement in the project (employees or volunteers). This view is useful for identifying broader trends in the Mozilla community and assessing how much various members contribute to a given project.

This dashboard is designed to help Mozilla module owners more effectively gauge and manage their contributors and to support strategic decisions around community engagement. However, developers are unlikely to benefit from the aggregate statistics of the received contributions.

Measuring Developer Contributions

Figure 2b illustrates a developer's patch contributions in a given month. While this information is useful to the project managers to explore developer activity, it doesn't provide concrete support for the developers' daily tasks. At the same time, our study found that developers desired better transparency on others' workloads, which largely leverages the same data as the quantitative graph. Developers wanted this data in order to determine who the "right" reviewer is to request a review from: "It's hard to know when you request a review, which of these five people has time." The right person to send a patch for review might be someone with a faster review turnaround or a shorter review queue. To identify the best-suited reviewer for approving a patch, developers need to be informed about reviewers' workloads and average response time. Although review queues are frequently used "to see who might be quickest," Bugzilla poorly supports this task: "You can check queues one at a time but it's a lot of work"; "The current system punishes the guy who is the most responsible or that is doing the best job." Apprising developers about the review queues of the key reviewers for a module is one way of balancing workloads.

Being able to see reviewers' workloads is particularly important if a developer isn't familiar with the component reviewers: "When submitting a patch for another component, it's more difficult; [I] have to try to figure out who is good in that component [and] look up their review info." Knowing what others are working on can enable developers to assign more relevant tasks to people as well as enable better load balancing.

Developers also want to be able to track their own patch activity, as well as determine which patches are awaiting reviews or who's blocking their reviews. Figure 4 illustrates a partial view of a qualitative dashboard that enables developers to track their patches (under the first tab, called Patches).

Presenting the list of submitted patches and sorting them by last touched date can help developers stay aware of the recent changes on their patches (such as new comments), review flag changes, and reassign the patch reviewer. In addition, a patch's status can be made more transparent, for example, by displaying the name of the reviewer on whom the patch is pending.

Measuring Developer Effectiveness

Quantitative dashboards are often used to monitor developer productivity. Figure 2c demonstrates a developer's productivity by providing details on how many patches he or she contributed, how many of them successfully landed in the project's code base, and how recently the developer contributed to the project (days since last landed patch).

Our study suggests that developers

find it difficult to determine what has happened since the last time an issue was examined (as noted by 12 developers). Bugzilla's design makes it hard for developers to track their tasks, such as bug fixing or code review. Developers want to be able to see the list of issues they need to fix, review, or follow up on by having task-specific views: "It would be cool if Bugzilla people could be assigned with views that would set up good defaults for the task that they are working on." As Figure 4 shows, the dashboard displays current and past code review tasks (under the Pending and Completed tabs, respectively) to increase developer awareness of what tasks are blocking others or what issues were recently resolved.

Qualitative dashboards such as these can organize information the way developers want, perhaps by displaying issues developers report, follow, and need to resolve, or listing patches submitted for review, as well as discussions on the issues (posted comments). Ordering issues by last touched date, for example, allows developers to better monitor recent changes on their tasks and activities.

Determining Performance Trends

Figure 3 shows a quantitative view displaying bug trends by status (open versus closed) and by priority (high versus low). This view can be helpful for managers who are trying to get a sense of project momentum leading up to various deadlines or to assess the project's overall health.

Again, our study found that developers are interested in status but in a different way: they want to be able to quickly determine how their bugs have evolved recently. They also want to be informed by the changes to the issues relevant to their work: "You want to get info based on what has changed since the last time you looked at. You wouldn't need to rely on bug mail; it's too easy to miss things in bug mail [and] could be embarrassing."

Private and public watch lists can help developers organize the large number of emails they need to filter. Public lists display issues that a developer needs to resolve or wants to monitor (being a module owner or quality assurance person). Developers often want to track the issues in which they're interested: "If there is something interesting, I will cc

Patches | Pending | Completed

Patch ID	Bug ID	Flag	Flag Setter	Requestee	Last Touched
712204	839088	review+	peterv		Yesterday
712205	839088	review+	peterv		Yesterday
712206	839088	review+	peterv		Yesterday
713367	840898	review+	trev.saunders		2 days ago
713367	840898	approval-mozilla-aurora+	lsblakk		2 days ago
713002	840614	review+	josh		1 week ago
711501	839116	review+	Ms2ger		1 week ago
711502	839116	review+	Ms2ger		1 week ago
711503	839116	review+	Ms2ger		1 week ago

FIGURE 4. An example of the qualitative dashboards supporting patch tracking and code review tasks.

ABOUT THE AUTHORS

OLGA BAYSAL is a PhD candidate at the David R. Cheriton School of Computer Science at the University of Waterloo. Her research interests include empirical software engineering, mining software repositories, software development analytics, and human aspects of software engineering. Baysal received an MMath in computer science from the University of Waterloo. Contact her at obaysal@cs.uwaterloo.ca.

REID HOLMES is an assistant professor in the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests include understanding how software engineers build and maintain complex systems. Holmes completed a Natural Sciences and Engineering Research Council of Canada postdoctoral fellowship at the University of Washington. Contact him at rtholmes@cs.uwaterloo.ca.

MICHAEL W. GODFREY is an associate professor at the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests include software evolution, mining software repositories, reverse engineering, program comprehension, and software clone detection and analysis. Godfrey received a PhD in computer science from the University of Toronto. Contact him at migod@uwaterloo.ca.

myself on it." These lists also allow developers to communicate interests on issues without taking ownership, as developers "would like to have a personal list of bugs without claiming them."

Public lists are visible to anyone on the project, whereas private lists are created by the developers themselves. As one developer puts it, "We need a way for people to set their own priorities on bugs, so a team has one, and the product has another, and security yet another. Each one of the initiatives should be represented. If it's P1 [high priority] for all three, then you know it's a big deal." Prioritizing issues and daily tasks can improve developers' time management when they accomplish important tasks first.

Developer-oriented qualitative dashboards don't aim to supplant quantitative approaches; both kinds of dashboards exist to provide specialized views into the incredible wealth of information available concerning how modern software systems are developed and maintained. We believe that by considering more task-specific views that are oriented toward common development tasks, these dashboards can improve developers' abilities to keep pace with the evolution of the issues they care about.

References

1. S. LaValle et al., "Big Data, Analytics and the Path from Insights to Value," *MIT Sloan Management Rev.*, vol. 52, no. 2, 2011, pp. 21–31.
2. D. Eaves, "Developing Community Management Metrics and Tools for Mozilla," blog, 7 Apr. 2011; http://eaves.ca/2011/04/07/developing-community-management-metrics-and-tools-for-mozilla.
3. O. Baysal and R. Holmes, "A Qualitative Study of Mozilla's Process Management Practices," tech. report CS- 2012-10, David R. Cheriton School of Computer Science, Univ. Waterloo, 2012; www.cs.uwaterloo.ca/research/tr/2012/CS-2012-10.pdf.
4. M. Best, "Bugzilla Anthropology," Mar. 2012; https://wiki.mozilla.org/Bugzilla_Anthropology.
5. M.R. Endsley, "Toward a Theory of Situation Awareness in Dynamic Systems: Situation Awareness," *Human Factors*, vol. 37, no. 1, 1995, pp. 32–64.