DRIVER - A platform for collaborative framework understanding

Nuno Flores, Ademar Aguiar
INESC-TEC & Dept. of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
nuno.flores@fe.up.pt, ademar.aguiar@fe.up.pt

***Abstract*—Application frameworks are a powerful technique for large-scale reuse but often very hard to learn from scratch. Although good documentation helps on reducing the learning curve, it is often found lacking, and costly, as it needs to attend different audiences with disparate learning needs. When code and documentation prove insufficient, developers turn to their network of experts. The lack of awareness about the experts, interrupting the wrong people, and experts unavailability are well known hindrances to effective collaboration. This paper presents the DRIVER platform, a collaborative learning environment for framework users to share their knowledge. It provides the documentation on a wiki, where the learning paths of the community of learners can be captured, shared, rated, and recommended, thus tapping into the collective knowledge of the community of framework users. The tool can be obtained at http://bit.ly/driverTool.**

I. Introduction

Frameworks are a powerful technique for large-scale reuse that helps developers to improve quality and to reduce costs and time-to-market. To be able to reuse a framework effectively, developers have to invest considerable effort on understanding it. Especially for first time users, frameworks can become difficult to learn, mainly because its design is often very complex and hard to communicate, due to its abstractness, incompleteness, superfluous flexibility, and obscurity.

Good quality documentation is crucial for the effective reuse of object-oriented frameworks [1]. Producing such documentation can be costly as it needs to be easy to use, to cover different audiences, and to present different types of documents using different notations. But even if the documentation is produced with quality standards, learners need to acquire knowledge from it and their cognitive needs must be attended. Learning about a framework is guided according to [2]:

- *Goal*. What does the learner expects to do with the framework (select, instantiate, evolve)?
- *Cognitive profile*. How does the learner instinctively tackles with the information (top-down vs. bottom-up, verbal vs. visual, sequential vs. global [3])?
- *Abstraction level*. Depending on the goal, it might be required to navigate up and down different abstraction levels of the framework. Which?
- *Knowledge availability*. Will the documentation suffice to learn how to use the framework?

To better support this learning process and to help on a more effective and efficient building of the mental model of the learner, we may suggest best practices (or patterns) [4]. But even with a process behind learning a framework, the documentation and the framework itself may not be sufficient to provide solutions (on how to use the framework) in a time-effective way.

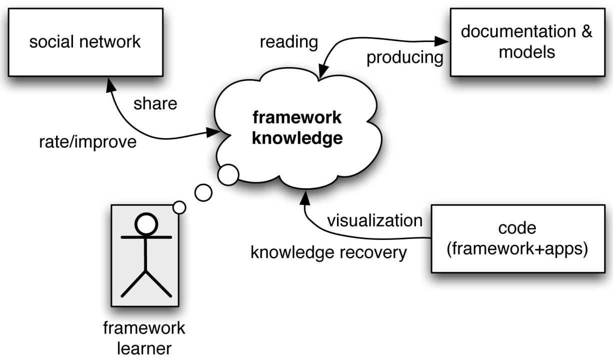


Fig. 1: Framework learning activities and actors

Software development is a highly social activity. As with software in general, a framework learner looks at the code, reads the documentation, visualises information and asks her colleagues for help, while going through the process of understanding how to use the framework (Figure 1). However, asking the team for help may present obstacles [5] [6]:

- *Availability*. Knowledgeable team mates are, most often, busy and not available to help due to task and time constraints.
- *Intrusion*. Interrupted developers lose track of parts of their mental model, resulting in laborious reconstruction or bugs and discouraging more frequent interruptions.
- *Tacit knowledge*. Developers spend vast amounts of time gathering precious, demonstrably useful information, but rarely recording it for future developers, rendering it useless.
- *Selfish ownership*. "Knowledge is power" is a commonly observed philosophy, specially in expert developers, afraid of loosing their status-quo.

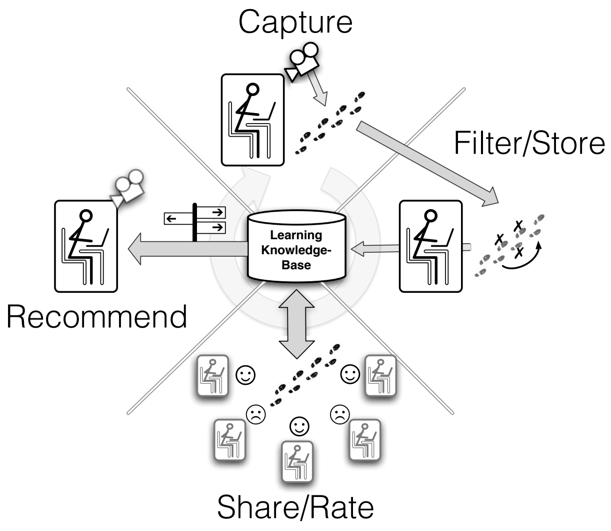


Fig. 2: DRIVER's 4-step learning knowledge cycle

With these issues in mind, and to support the activities taken during the learning process, the authors propose to help the learner in two ways: (1) providing a "guide" or "map", in the form of patterns, of the best way to "drive" the learning process and (2) allowing the learner to tap into the knowledge of the learning community through the use of appropriate tools. Both strategies are integrated into a collaborative, shared data-driven environment named DRIVER.

DRIVER enables the capture, storage, sharing, rating, and recommendation of learning knowledge, namely *learning paths*, i.e., the steps the learner took (while going through the documentation) that enabled her to build a solution to her problem. This environment is built upon a wiki that provides documentation artifacts about the framework, which is configurable enough to allow knowledge acquisition in several ways.

II. Overview

DRIVER is a platform that enables users to effectively learn how to use a framework in a collaborative, user-friendly, knowledge-intensive environment. It promotes social learning within a community of framework users, with different levels of experience, motivated to find answers to their problems and at the same time to share them for the benefit of all. Its architecture relies on the notion of a Collective Knowledge System [7], supporting knowledge quality evolution through social interaction. Its features include:

- *Collective knowledge management*. The learning knowledge is captured and maintained by the community in a non-intrusive way. Learners can search and rate available knowledge and get recommendations on the best course of action.
- *Best practices support*. Previous work by the authors resulted in a set of patterns [4] for assisting in framework understanding. These are available to the platform user.
- *Collaborative documentation*. The framework documentation artifacts are available for editing and updating by the community of learners.
- *Social Classification*. Tagging and folksonomies are at the basis of the learning knowledge classification.
- *Extensibility*. The platform is open for extension to accommodate new features that might appear in the future.

These features cover a set of requirements, collected by the authors, deriving from previous research and bibliographic studies on collective intelligence and collaborative learning [8]. At its core, the DRIVER toolset supports a process of social learning described next.

A. Learning cycle

The authors believe that providing a learner with the steps others (learners) took to solve their problems (a.k.a. *learning path*), can improve the learning experience and produce better and quicker outcomes. The motto is: *Show me how you learnt it yourself.*

The goal is to non-intrusively capture the learning steps a framework user takes, store them in a sharable knowledge-base, which other users can access. This knowledge relies on the community's potential to maintain its relevance and quality, by rating it and allowing the system to recommend possible next steps that aid on the learning task. This can be described as a 4-step cycle as follows (see Figure 2):

- *Capture*. The learner begins her learning quest to find knowledge that might solve her problem. The trail of steps is captured as she browses through the artifacts, trying to find the relevant knowledge that might help her.
- *Filter/Store*. The learner improves the captured learning path by trimming off those steps that, despite taken, didn't lead to the required knowledge. This prevents other learners from running in circles or hitting dead-ends. Afterwards, the pruned and grafted learning path is stored in a shared knowledge-base.
- *Share/Rate*. The learners access the knowledge-base, searching for learning paths that might help them. They evaluate its usefulness (taking the steps, a.k.a. walking through or just inspecting the visited artifacts) and rate them according to its effectiveness.
- *Recommend*. This step enables the recommendation of possible next steps (on a learning path that is being currently captured), based on previous learning paths other learners have took. Actually, this step occurs, in parallel, during the Capture step. The more learning paths, the better the recommendation becomes, motivating the community to participate.

B. Components

The DRIVER platform is composed of the following components:

- *Wiki*. Being lightweight, semi-structured, extensible and quite popular, a wiki served as a foundation for harbouring the collaborative environment and its components.

- *Framework documentation artifacts repository (FDAR)*. The wiki contents consist mainly of a set of documentation artifacts[1] about the framework in question. These contents will have to be produced up to an extend[2] and incorporated into the wiki.
- *Patterns*. The best practices for framework understanding in pattern form.
- *Learning cycle support sub-components*. A set of sub-components that support the implementation of the 4-step learning cycle.

C. Usage scenarios

Thus, the intended audience for the DRIVER platform are framework learners, that is, framework users trying to effectively use the framework. In terms of using the platform, it can be briefly summed up into two main scenarios: *Warm up* and a *typical usage*.

1) Warm Up: This scenario happens when setting up the platform and making it available to the users (potential framework learners). By itself, the platform does not have the documentation about the framework, thus it needs to be produced and added to the wiki. Similarly, there is no previous learning paths data on the knowledge-base, therefore, it requires some usage time[3] so that subsequent learners may benefit from previous knowledge on how to learn about the framework.

2) Typical usage: During a typical usage, the user will just need to login into the platform and signal the beginning of its learning *quest*. Immediately, the 4-step learning cycle sub-components become available and its steps subsequently captured. A depiction of a typical screen of the platform can be seen in Figure 3. Navigation is free throughout the framework documentation (i.e, the wiki) until the user has satisfied her learning needs. At any given moment, the user can *share* a set of steps it has taken (deciding it is a learning path by itself and *storing* it in the knowledge-base), *search* for existing learning paths or continuing to browse the documentation, taking ”hints” from the *recommendation* component. Even without entering a ”capturing” state, a user can, at any time, make a tag-based search over the learning paths knowledge base, being able to *walk-through* the resulting learning paths, and *rating* them according to its usefulness. An illustration of a typical usage scenario can be found at **http://bit.ly/driverTool**.

III. Community impact

The presented approach tackles with the intrusiveness the learning process can have when directly asking for help, usually resulting in disregarding the request, exhibiting non-availability and, progressively forgetting useful learning knowledge (the issue of loosing tacit knowledge). Intrusiveness is thus mitigated by resorting to a shared and maturing knowledge-base of learning knowledge. Not only the *asking* learner isn't interrupting her colleagues, but the *knowledgeable* learner can, without disrupting his normal functions, capture and store the grafted learning path, as part of the common procedure of learning. This also contributes in diminishing the loss of tacit knowledge.

A. Towards a collective knowledge system

An expected (by the authors) major impact in the software engineering community is the use of the collective knowledge of the community itself. Tapping into the collective intelligence was a step yet to be taken when it comes to framework learning. Thus DRIVER has its toolset based on the notion of a Collective Knowledge System [7]. This concept has a set of key properties and supporting components which are described next, along with DRIVER's covering of such features.

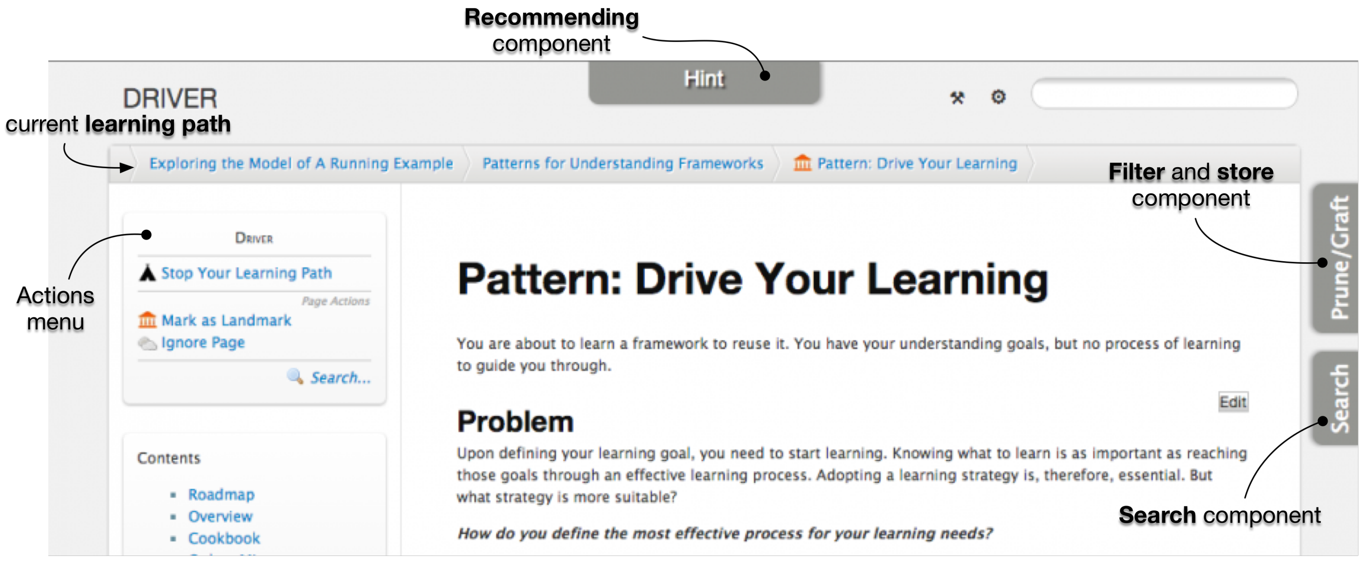
1) Key properties: A Collective Knowledge System provides the following key properties:

- **User generated content** - The bulk of the information is provided by humans participating in a social process. A traditional database or expert system, in contrast, gets the bulk of its information from a systematic data gathering or knowledge modeling process. In DRIVER, most of the content is created/edited/evolved by the community. Whether the documentation artifacts in the wiki or the learning path knowledge-base, the information always originates from the users and can be evolved by them.
- **Human-machine synergy** - The combination of human and machine provides a capacity to provide useful information that could not be obtained otherwise. These systems provide more domain coverage, diversity of perspective, and sheer volume of information than what it could be achieved by searching official literature or talking to experts. In DRIVER, automation of tasks such as collecting tags from the artifacts during browsing, presenting similar learning paths to the one just captured or providing recommendations, spares the user from collecting this information on her own.
- **Increasing returns with scale** - As more people contribute, the system becomes more useful. The system of rewards that attracts contributors and the computation over their contributions is stable as the volume increases. In contrast, a text corpus and simple keyword search engine does not get more useful when the volume of content overwhelms the value of keywords to discriminate among documents. Similarly, if the reward system encourages fraud or fails to bubble up the best quality content, the system will get less useful as it grows. In DRIVER, as more learners rate existing learning paths, the search tool brings up the most used and better rated results, contributing with higher valued information to the user.
- **Emergent knowledge** - The system enables computation and inference over the collected information, leading to answers, discoveries, or other results that are not

[1]These were based on a set of patterns for effectively documenting a framework [9].

[2]The minimal documentation that allows a user to effectively use the framework.

[3]The *pioneering* learners of the framework will, for a while, record their learning paths for future learners.



Pattern: Drive Your Learning

You are about to learn a framework to reuse it. You have your understanding goals, but no process of learning to guide you through.

Problem

Upon defining your learning goal, you need to start learning. Knowing what to learn is as important as reaching those goals through an effective learning process. Adopting a learning strategy is, therefore, essential. But what strategy is more suitable?

How do you define the most effective process for your learning needs?

Fig. 3: Screenshot of the DRIVER platform, showing the main access points to the toolset components.

found in the human contributions. This fourth property is what differentiates a *collective* from a *collected* knowledge system. In DRIVER, the system offers recommendations for guidance through the artifacts, based on existing learning paths in the knowledge-base, inferring new knowledge from the existing data. Of course, the recommendation heuristic can be extended to improve the results and to more suitably provide effective recommendations. Nevertheless, the basis is there.

2) Components: Conveying the presented key properties, a Collective Knowledge System can be composed of the following elements:

- **Community of motivated people with problems and solutions**. These contributors share their expertise and knowledge on the specific domain.
- **Larger population of intelligent people with similar problems**. Who actively search for personalised solutions to their problems.
- **Computer mediated social communication**. Whether through tagging, blogging or commenting, the social process is augmented and nurtured.
- **Semi-structured information repository**. Acting like a storage facility for a more long-term memory and where the solutions are collected and shared.
- **Socially clustered data knowledge-base**. Where the solutions are catalogued and clustered according to the social interaction and multidimensional analysis.
- **Faceted search engine**. So that the solutions seekers can look for personalised solutions, through contextual browsing.
- **Recommendation engine**. To keep the users in perspective and assisting in obtaining more rapid and effective answers to their specific issues.

All of these components are implemented/provided by the DRIVER platform, as depicted in Figure 4.

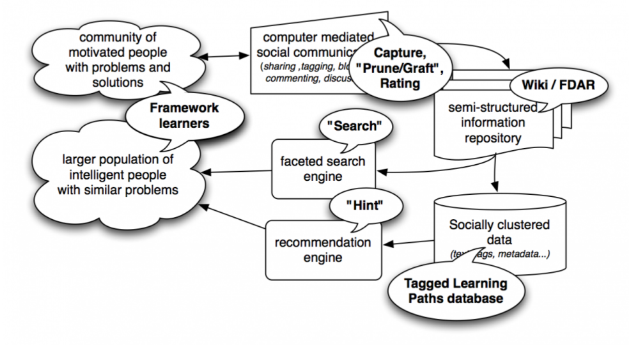

community of motivated people with problems and solutions
computer mediated social communic
(sharing ,tagging, bl commenting, discus
Capture, "Prune/Graft", Rating
Wiki / FDAR
Framework learners
semi-structured information repository
"Search"
larger population of intelligent people with similar problems
faceted search engine
"Hint"
Socially clustered data
(ta ags, metadata...)
recommendation engine
Tagged Learning Paths database

Fig. 4: The DRIVER toolset as a Collective Knowledge System, with a matching of the implemented components.

In short, the proposed collaborative environment provides an extensible scaffold for building and extending a collective knowledge acquisition system, where the community of learners can share their insight without too much effort and benefit from its collective wisdom.

IV. Related work

Regarding framework understanding, most solution proposals found in the literature converge to produce and enhance existing documentation, such as design patterns [10], pattern languages [11], cookbooks [12], hooks [13], exemplars [14] and minimalist documentation [1]. Nevertheless, the true impact of these techniques on framework understanding is still fuzzy. Even so, there are a few studies that deal with issues around effective framework reuse and understanding.

In [15], Schull et al. presented an evaluation of the role that examples play in framework reuse. Their study compared two approaches to framework reading: example-based approach and hierarchical-based approach. Their results suggested that examples are an effective learning strategy, especially for those beginning to learn a framework. Nevertheless, examples had issues: finding the small pieces of required functionality in larger examples; inconsistent structure and organisation; lack of design choice rationale and shallowness in understanding the framework internals.

In [16], Morisio et al. conducted an empirical study in an industrial context on the production of software using a framework, as to investigate quality and productivity issues and the effect of learning in framework-based object-oriented development. They observed higher quality and productivity levels in framework-based applications, due to a learning effect from repetition the same task over time. Yet, a more proficient developer has to engage on high level framework knowledge learning.

In [17], Kirk et al. conducted a research, through observation of both novice and experienced users, where they identified four fundamental problems of framework reuse: (1) Mapping the problem onto the framework, (2) Understanding functionality, (3) Understanding interactions and (4) Understanding the framework architecture. Applying both pattern languages and micro-architectures. Their results showed that the pattern language provided some support for mapping problems, particularly for those with no experience of the framework, by introducing key framework concepts and providing examples of framework use. Yet, experienced users of the framework discarded the pattern language, find it constraining. Despite believing in micro-architectures, these seemed relatively ineffective.

Several studies have been led by Hou et al.[18][19][20], regarding framework usage and understanding. They unveiled issues regarding design (tight-coupling, delocalised concerns, excessive special cases), documentation (doc-driven understanding fares better the reverse engineering the code) and learner profile (Novice learners rely on the existing documentation first, make a shallow study using available examples and, in distress, ask for help). They also shown that novice learners tend to favour functional aspects over non-functional and that spending some time (up-front) learning about the design of the framework is beneficial to a more effective framework reuse, impacting on the application of examples, that should be functionality-driven. They propose a set of tool requirements for reuse that rely on better communication, better semantic search, improved tracking of intermediate results and better IDE-integration.

When its comes to tooling for support framework learning specifically, there is not much beyond that previously stated. Nevertheless, looking at the type of platform presented by DRIVER, it is relevant to mention other existing tools that might, somehow, relate to DRIVER when it comes to software knowledge management.

One such tool is MyLyn [21], currently integrated in the Eclipse IDE [22]. MyLyn provides support to the notion of Degree of Interest (DOI) [23], where its allows a filtered view over the relevant[4] artifacts inside Eclipse, allowing a more focused and decluttered development environment. These views (called *contexts*) can be stored and retrieved according to the task at hand, providing an easier and faster context switch between tasks. While MyLyn provides a *snapshot* of the most relevant artifacts, DRIVER provides a *pathway* of the most relevant artifacts. Both approaches rely on the relevancy (DOI) of the artifacts, yet, for learning purposes, DRIVER relies on the progress throughout the artifacts, while MyLyn only show the latest state of DOI. Yet the underlying motto behind both tools is the same: *focus*.

Another issue worth mentioning is the fact that the concept of learning path can be compared to what is known as *clickstream*, a web analytics metric. According to WAA[5], Web Analytics is the measurement, collection, analysis and reporting of Internet data for the purposes of understanding and optimizing Web usage. Clickstream tracks by which order the visitor of a site navigated through its contents[6]. A learning path can be seen as a constrained form of clickstream, where only the relevant clicks (navigating between documents and liking sections) are captured.

V. Conclusions and Future Work

This paper presented DRIVER, a collaborative environment that supports the framework learning process. Not only it relies on an easy, sharing, lightweight, editable platform (wiki), that provides documentation artifacts about the framework, but it promotes knowledge acquisition by enabling capture, storage, sharing, rating and recommendation of learning knowledge.

This learning knowledge takes the form of learning paths. These show how other learners tackled with similar problems by presenting which documentation artifacts they went through and by which order. The presented toolset supports this kind of learning process through a series of components that seamlessly integrate the documentation infra-structure. These learning paths are stored and shared by the community of learners, who rate the level of usefulness this learning data has, allowing the information to mature and improve its quality and applicability throughout the community.

An experiment was conducted [2] within a controlled experimental environment to validate the usefulness of DRIVER environment, where the final results support the hypothesis that the collaborative approach helps novices to more effectively learn about a framework.

As future work, enhancements to the DRIVER platform are currently under way, focusing on improving recommendation heuristics, gamification techniques to improve user adhesion, increase learner's profile awareness and convergence to the semantic web. Also, further studies in industrial settings with a different time-scale and a broader community of learners are being designed as to reinforce the current evidence and to study the impact on framework understanding and software learning in general.

[4]Those actually viewed and edited, within the artifacts set, i.e. those with a high DOI.

[5]Web Analytics Association

[6]The click stream is the sequence of mouse clicks the user performed on the contents of the site, but 37 usually, only the navigation (link clicks) is recorded.

REFERENCES

[1] A. Aguiar, “Framework documentation – a minimalist approach,” Ph.D. dissertation, FEUP, September 2003.

[2] N. Flores, “Patterns and tools for improving framework understanding: a collaborative approach,” Ph.D. dissertation, University of Porto, Faculty of Engineering, 2012, http://paginas.fe.up.pt/ñflores/dokuwiki/lib/exe/fetch.php?media=research:nuno_flores_phd_thesis.pdf [Online; accessed April 2014].

[3] R. Felder and J. Spurlin, “Applications, reliability, and validity of the index of learning styles,” *International Journal of Engineering Education*, vol. 21(1), pp. 103–112, 2005.

[4] N. Flores and A.Aguiar, “Patterns for framework understanding,” in *15th Pattern Languages of Programming Conference (PLoP’08)*, Nashville, USA, October 2008.

[5] K. Nakakoji, Y. Yamamoto, and Y. Ye, “Supporting software development as knowledge community evolution,” in *Proceedings of the CSCW Workshop on Suporting the Social Side of Large Scale Software Development*, 2006.

[6] T. D. LaToza, G. Venolia, and R. DeLine, “Maintaining mental models: A study of developer working habits,” in *Proc. of the International Conference of Software Engineering (ICSE’06)*, Shanghai, China, 2003.

[7] T. Gruber, “Collective knowledge systems: Where the social web meets the semantic web.” *Journal of Web Semantics*, 2007.

[8] N. Flores and A. Aguiar, “Understanding frameworks collaboratively : Tool requirements,” *International Journal on Advances on Software*, vol. vol. 3, 2010.

[9] A. Aguiar and G. David, “Patterns for effectively documenting frameworks,” in *Transactions on pattern languages of programming II*, J. Noble and R. Johnson, Eds. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 79–124. [Online]. Available: http://portal.acm.org/citation.cfm?id=1983735.1983740

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns — Elements of reusable object-oriented software*. Addison-Wesley, 1995.

[11] R. Johnson, “Documenting frameworks using patterns,” in *Proceedings of the OOPSLA’92, SIGPLAN notices*, vol. 27(10), 1992, pp. 63–76.

[12] G. E. Krasner and S. T. Pope, “A cookbook for using the model-view-controller user interface paradigm in smalltalk-80,” *Journal of Object-Oriented Programming*, vol. 1(3), pp. 26–49, 1988.

[13] G. Froehlich, H. Hoover, L. Lui, and P. Sorenson, “Hooking into object-oriented application frameworks,” in *Proceedings of the 19th International Conference on Software Engineering*, 1997, pp. 491–501.

[14] D. Gangopadhyay and S. Mitra, “Understanding frameworks by exploration of exemplars,” in *Proceedings of CASE-95*, I. C. Society, Ed., 1995, pp. 90–99.

[15] F. Schull, F. Lanubile, and V. Basil, “Investigating reading techniques for object-oriented framework learning,” *IEEE Transactions on Software Engineering*, vol. 26(11), 2000.

[16] M. Morisio, D. Romano, and I. Stamelos, “Quality, productivity, and learning in framework-based development: An exploratory case study,” *IEEE Transactions on Software Engineering*, vol. 28(9), pp. 876–888, 2002.

[17] D. Kirk, M. Roper, and M. Wood, “Identifying and addressing problems in framework reuse,” in *Proceedings of the 13th International Workshop on Program Comprehension (IPWC’05)*, 2005, pp. 77–86.

[18] D. Hou, K. Wong, and H. J. Hoover, “What can programmer questions tell us about frameworks?” in *Proceedings of the 13th International Workshop on Program Comprehension (IPWC’05)*, 2005, pp. 87–96.

[19] D. Hou, “Investigating the effects of framework design knowledge in example-based framework learning,” in *IEEE International Conference on Software Maintenance*, 2008, pp. 37–46.

[20] D. Hou and L. Li, “Obstacles in using frameworks and apis: An exploratory study of programmers’ newsgroups discussions,” in *IEEE 19th International Conference in Program Comprehension (ICPC)*, 2011.

[21] “Eclipse mylyn open source project,” July 2015, http://www.eclipse.org/mylyn/ [Online; accessed July 2015]. [Online]. Available: http://www.eclipse.org/mylyn/

[22] “Eclipse project,” July 2015, http://www.eclipse.org [Online; accessed July 2015]. [Online]. Available: http://www.eclipse.org

[23] M. Kersten and G. Murphy, “Mylar: a degree-of-interest model for ide’s,” in *International Conference on Aspect Oriented Software Development*, 2005, pp. 159–168.