International Journal of Software Engineering
and Knowledge Engineering
Vol. 27, No. 5 (2017) 699–725
© World Scientific Publishing Company
DOI: 10.1142/S0218194017500267

World Scientific
www.worldscientific.com

Learning Frameworks in a Social-Intensive Knowledge Environment — An Empirical Study

Nuno Flores* and Ademar Aguiar†

Department of Informatics Engineering & INESC TEC
University of Porto - Faculty of Engineering
Rua Roberto Frias s/n, Porto, Portugal
**nuno.flores@fe.up.pt*
†ademar.aguiar@fe.up.pt

Received 1 April 2016
Revised 12 May 2016
Accepted 6 June 2016

Application frameworks are a powerful technique for large-scale reuse, but require a considerable effort to understand them. Good documentation is costly, as it needs to address different audiences with disparate learning needs. When code and documentation prove insufficient, developers turn to their network of experts. Nevertheless, this proves difficult, mainly due to the lack of expertise awareness (who to ask), wasteful interruptions of the wrong people and unavailability (either due to intrusion or time constraints). The DRIVER platform is a collaborative learning environment where framework users can, in a non-intrusive way, store and share their learning knowledge while following the best practices of framework understanding (patterns). Developed by the authors, it provides a framework documentation repository, mounted on a wiki, where the learning paths of the community of learners can be captured, shared, rated, and recommended. Combining these social activities, the DRIVER platform promotes collaborative learning, mitigating intrusiveness, unavailability of experts and loss of tacit knowledge. This paper presents the assessment of DRIVER using a controlled academic experiment that measured the performance, effectiveness and framework knowledge intake of MSc students. The study concluded that, especially for novice learners, the platform allows for a faster and more effective learning process.

Keywords: Collaborative learning; frameworks; tools.

1. Introduction

Frameworks are a powerful technique that enables large-scale reuse, helping developers to improve software quality and reduce costs and time-to-market. In its definition, a framework is a reusable design together with an implementation. It consists of a collection of cooperative classes, both abstract and concrete, which embody an abstract design for solutions to problems in an application domain

[9, 10, 26]. 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 [5].

Providing an accompanying good quality documentation is thus crucial for the effective reuse of object-oriented frameworks. Producing such documentation is no trivial task [2, 42], as it needs to be easy to use, to cover different audiences, and to present different types of documents using different notations. Assuming the documentation is produced with quality standards, the framework users still need to go through the process of acquiring knowledge from its contents. This process is, usually, guided according to a set of interdependent aspects:

- *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 versus bottom–up, verbal versus visual, sequential versus global [11])?
- *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, there are best practices (or patterns) [13] that one can follow. But even with a process behind learning a framework, the documentation and the framework itself may not be sufficient to provide solutions in a time-effective way.

Software development is, besides knowledge-intensive, a highly social activity. Like software developers in general, a framework learner looks at the code, reads the documentation, visualizes information and asks her colleagues for help, as part of the process of understanding how to use the framework (Fig. 1). However, asking the team for help may prove hard, due to its inherent human nature [30, 37]:

- *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 a laborious reconstruction or *bugs* and discouraging more frequent interruptions.
- *Tacit knowledge.* Developers spend vast amounts of time gathering precious, demonstrable useful information, but thus rarely record it for future developers, rendering it useless.
- *Selfish ownership.* "Knowledge is power" is a commonly observed philosophy, especially in expert developers, afraid of loosing their *status-quo*.

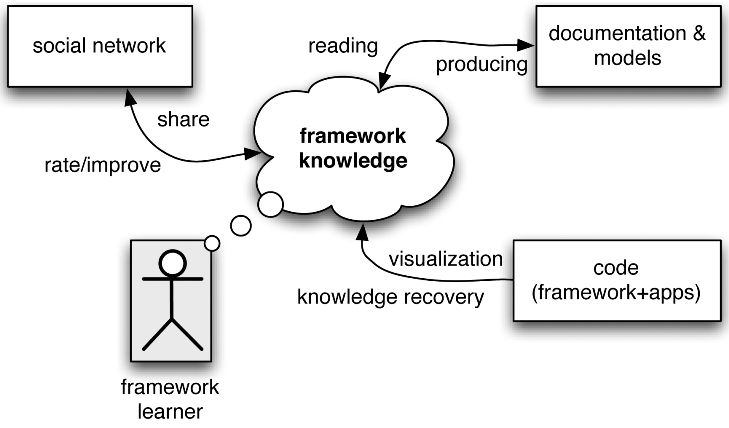


Fig. 1. Framework learning activities and actors.

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 undertake those activities 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 [15].

DRIVER includes a (sub-)set of tools that enable 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 toolset is built upon a wiki that provides documentation artifacts about the framework, and is configurable enough to allow knowledge acquisition (KA) in several ways.

Software engineering research comprises computer science issues, human issues and organizational issues. It is thus often convenient to use combinations of research approaches both from computer science and social sciences. The taxonomy described by Zelkowitz and Wallace [46] identifies 12 different types of experimental approaches for software engineering, grouped into three broad categories: *observational*, *historical* and *controlled*.

For evaluating the DRIVER platform, the choice fell on conducting a *controlled* replicated experiment. This experiment studies intermediate-experienced developers in understanding a framework, collecting time and knowledge acquisition metrics and comparing results from differently set-up learning environments. Its purpose was to find evidence that the presented collaborative approach helps novices and experts to improve their framework learning process.

The experiment took groups of similar MSc students and measured their performance, effectiveness and framework knowledge intake, while developing a set of tasks using a new framework. In parallel, a set of students already knew the framework, as to study the process of re-acquiring dormant framework knowledge. The final results support the hypothesis that the collaborative approach helps improving framework learning, especially for first-time learners and novices.

This paper is organized as follows: Section 2 elaborates on the motivation behind the design of the proposed platform. Section 3 describes the DRIVER platform. Section 4 describes the controlled experiment, its design, subjects and protocol. Results are reported and analyzed in Sec. 5. Section 6 discusses threats to the validity of the results. Section 7 reviews related work, and Sec. 8 summarizes our conclusions and points to future work.

2. Collaboratively Improving Learning

Software development is carried out by a group of developers, forming a community and engaging in collective creative knowledge work [36]. It is a social activity mediated through artifacts, which are, primarily, source code and documents. Although sharing knowledge and information within a community of developers being indispensable, the primary means for developers to obtain knowledge is not through communicating with their peers, but through artifacts. Developers invest great effort recovering implicit knowledge by exploring code and documents. If this fails, they turn to their social network [30]. There have been examples of software problems and issues, specifically while using frameworks, being solved resorting to the community of users [43].

In [40], Surowiecky presents an extensive analysis on how knowledge and reasoning in a group of people provide better results, on average, than an informed, expert individual. He states that, despite unawareness of it, "*we are collectively smart*" and intellectual superior to the isolated individual. He calls this notion *wisdom of the crowd*, built on four essential pillars a group of people usually contains: diversity, independence, decentralization and aggregation.

2.1. *Grasping the collective knowledge*

Effectively capturing expertise from several heterogeneous sources in a social environment is the goal of the Collaborative Knowledge Acquisition field of study, a spin-off of the KA domain. KA deals with the process of extracting, structuring, and organizing knowledge from human experts so that the problem-solving expertise can be captured and transformed into a computer-readable form in order to fuel expert systems [33, 44]. KA is a complex task with several identified issues that capturing techniques should address [33, 34], such as: (1) Most (but not all) knowledge is in the head of experts; (2) Experts have vast amounts of knowledge; (3) Each expert does not know everything; (4) Experts have a lot of tacit knowledge; (5) Experts are very valuable and busy people and (6) Knowledge has a "shelf-life".

KA in a social environment shares the same issues. Additionally, the developer has to rely on distributed knowledge resources (artifacts and people) where not everyone is an expert. This becomes even worse if the community scope goes beyond the team of developers and extends to the web, where other developers may have the answer for a specific problem regarding a well-known shared software artifact, API or framework.

The quality of the retrieved knowledge is evaluated by the behavior of the community towards that knowledge. *If it is useful, it is used, if not, it is abandoned.* One way of capturing this behavior is to give the community ways of expressing their intent, whether through rating or commenting. Otherwise, there are ways of implicitly capturing the community behavior, like *page hits*[a] or *social bookmarking.* This is known as Collaborative KA [32], as it gathers information from several heterogeneous sources, such is the morphology of the Internet. Systems that enable this kind of KA are denominated *Collective Knowledge Systems.*

2.2. *Collective knowledge systems*

In [20], Tom Gruber states

> "The web, as a community, is not yet a collective intelligence, rather a collected intelligence. This comes from the fact that there is no new level of understanding. User-generated content is being shared, gathered and collected in domain-specific sites. We can find what things are more popular or what are the current fads. However, while popularity is one measure of quality, it is not a measure of veracity. Mass authoring is not the same thing as mass authority."

This classification of *collected versus collective* intelligence of the web renders a definition of what a Collective Knowledge System can be and its key properties are summarized below:

- *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.
- *Human-machine synergy.* The combination of human and machine enables 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, when compared to results obtained by searching official literature or talking to experts.
- *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, do not get more useful when the volume of content overwhelms the value of keywords to discriminate among documents.

[a] The number of web users that visit that page.

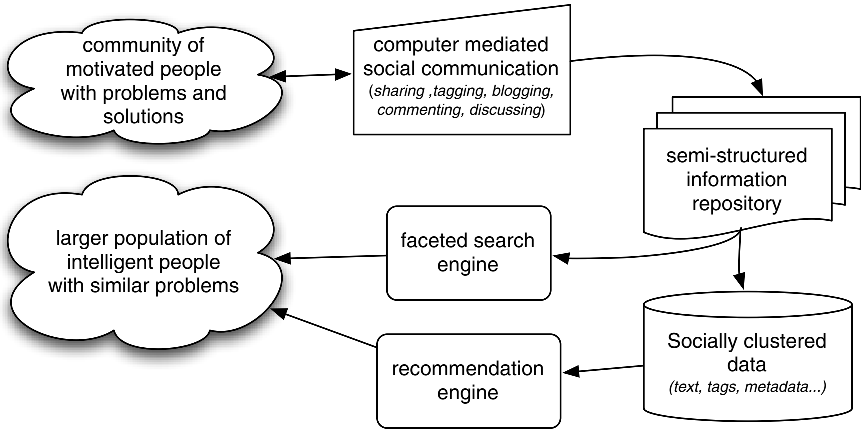

community of motivated people with problems and solutions
computer mediated social communication
(sharing ,tagging, blogging, commenting, discussing)
semi-structured information repository
larger population of intelligent people with similar problems
faceted search engine
Socially clustered data
(text, tags, metadata...)
recommendation engine

Fig. 2. Composing elements of a Collective Knowledge System (adapted from [20]).

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.

- *Emergent knowledge.* The system enables computation and inference over the collected information, leading to answers, discoveries, or other results that are not found in the human contributions. *This fourth property is what differentiates a collective from a collected knowledge system.*

Conveying these key properties, a Collective Knowledge System can be composed of the following elements, depicted in Fig. 2:

- *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 personalized 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 cataloged and clustered according to the social interaction and multidimensional analysis.
- *Faceted search engine.* So that the solutions seekers can look for personalized 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.

It might be relevant to say that not all of these elements need to be present for a system to be considered of Collective Knowledge. At least, the knowledge quality evolution through social interaction and its access need to be enforced.

3. The DRIVER Platform

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 (Sec. 2.2), 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 least-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 [13] 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 also aim at covering a set of requirements that derived from previous research, by the authors, on collective intelligence and collaborative learning [14].

3.1. *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 shareable knowledge-base, where other users can access it. 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 Fig. 3):

- *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, did not lead to the required knowledge. This

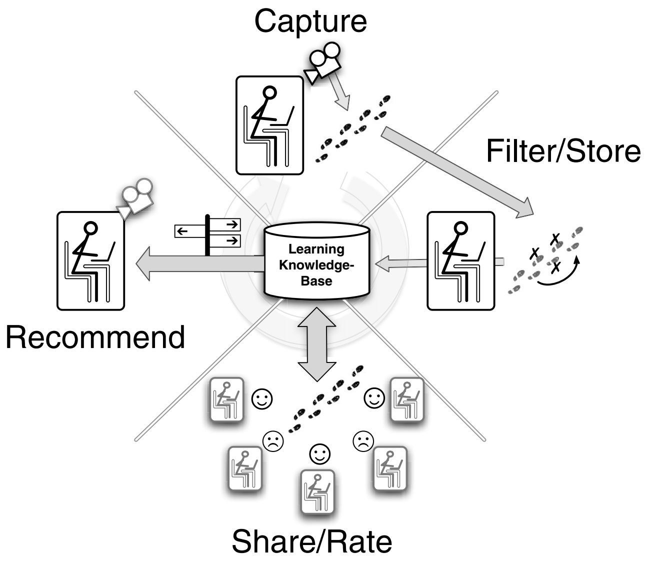


Fig. 3. DRIVER's 4-step learning knowledge cycle.

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.

3.2. *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 harboring the collaborative environment and its components.
- *Framework documentation artifacts repository.* The wiki contents consist mainly of a set of documentation artifacts[b] about the framework in question.

[b] These were based on a set of patterns for effectively documenting a framework [4].

- *Patterns.* The best practices for framework understanding in pattern form (developed by the authors [13], as previously stated).
- *Learning cycle support plug-ins.* A set of wiki plug-ins that support the implementation of the 4-step learning cycle.

Further details about the platform are available online at `bit.ly/driverTool`.

4. Experiment Design

The use of Empirical Studies With Students (ESWS) in software engineering helps researchers gain insight into new or existing techniques and methods. Scientific grounds and discussion for the usage of such *controlled* [46] methods for software engineering research validation can be found in [19, 28] and [39]. ESWSs can be valuable to the industrial and research communities if they are conducted in an adequate way, address appropriate goals, do not overstate the generalization of the results and take into account threats to internal and external validity [6]. As ESWSs are often used to obtain preliminary evidence in support of or against research hypothesis, this experiment was designed as an experimental package (available at [1]), to be performed in different locations, and by different researchers, in order to enhance the ability to integrate the results obtained and allow further meta-analysis on them.

4.1. *Subjects*

The experiment subjects were 23 MSc students from the Integrated Master in Informatics and Computing Engineering, lectured at the University of Porto, Faculty of Engineering. They were part of a fourth year class, attending an optional course on "Architecture of Software Systems". Its syllabus deals strongly with frameworks and patterns, therefore it was more than suitable to integrate this experiment into their course work.

4.1.1. *Group formation*

The subjects were divided into four groups, each with its own purpose.

- *Baseline* (BL). This group established the baseline for the experiment, serving as the control group. Its subjects used the framework with no aids but the documentation.
- *Experimental Group* 1 (EG1). This group used the framework having, besides the documentation, the patterns as an aid. The purpose was to compare results with the baseline group and provide evidence for the usefulness of the patterns.
- *Experimental Group* 2 (EG2). This group used the framework having the DRIVER platform (documentation, patterns and wiki plug-ins) as aid. The purpose was to compare results with the baseline group and provide evidence that the proposed collaborative environment improves framework understanding.

- *Experts* (EX). This group used the framework having the DRIVER platform (documentation, patterns and wiki plug-ins) as aid. The purpose was to provide evidence that the proposed collaborative environment also helps experts[c] increase their knowledge of the framework.

4.1.2. *Pre-experiment evaluation*

For an experiment of this kind, it is important to assure that the subjects are similar, that is, their base skills do not pose a significant threat to the validity of the results. Therefore, they were scrutinized based on their academic track, by analyzing their grades on a selected subset of courses, deemed relevant to the outcome of the experiment. An independent samples t-test was conducted to compare the average students' grades (shown in Table 1) between the baseline and the other experimental groups. Table 2 shows that there was no significant difference between the baseline group and the remaining groups.

Table 1. Student grades group statistics.

Group	*N*	Mean	Std. deviation	Std. error mean
BL	6	16.838	0.9261	0.3780
EG1	4	16.928	1.3527	0.6763
EG2	4	16.943	1.4869	0.7435
EX	9	16.730	1.0213	0.3404

Table 2. *Independent Samples Tests.* The first two columns are the Levene's Test for Equality of Variances, showing a significances greater than 0.05 (Sig.) for all cases. The other three columns are the t-test for Equality of Means. Since we can assume equal variances, the 2-tailed values (0.902, 0.802, 0.839), allow us to conclude that there is no statistically significant difference between the two conditions.

	F	Sig.	t	df	Sig. (2-tailed)
(a) Baseline versus Experimental Group 1.					
Eq. Var. Assumed	0.269	0.618	−0.13	8.000	0.902
Eq. Var. Not Assumed			−0.12	4.882	0.912
(b) Baseline versus Experimental Group 2.					
Eq. Var. Assumed	0.607	0.458	0.259	8.000	0.802
Eq. Var. Not Assumed			0.234	4.569	0.825
(c) Baseline versus Experts.					
Eq. Var. Assumed	0.050	0.827	0.208	13.00	0.839
Eq. Var. Not Assumed			0.212	11.62	0.836

[c] *Experts*, in this context, means prior framework users, thus having retained knowledge about how to use the framework.

4.2. *Framework selection*

Choosing a framework to use in the experiment was an issue. One of the main concerns was finding a framework that would suit all the experimental groups. The experiment needed a framework that was unknown to most groups (BL, EG1, EG2) but known to the Experts (EX) group. The choice fell on an in-house, proprietary, white-box framework called OGHMA [12]. It is an object-oriented framework targeted to the development of information systems, on an industrial level, which structural requirements can be best described as "incomplete by design". It relies on the Adaptive Object-Model meta-architectural pattern [45] to allow run-time domain evolution by the user. OGHMA had been used by some of the students during a previous course, enabling both knowledgeable and non-knowledgeable profiles. In future experiments, a previous, more extensive survey must be made to the subjects in order to find a suitable framework that can cope with all the constraints and where training of the Experts group might be required.

4.3. *Environment*

According to [6], regarding ESWSs,

> "*The study setting must be appropriate relative to its goals, the skills required and the activities under study.*"

Considering this requirement, the experiment was conducted on a familiar setting to the students, as to minimize the external environmental factors that might threaten the validity of the results. It took place in laboratory classrooms, usually used by the students to attend classes or develop their course work. To enforce the collaborative work [6], the students were grouped into pairs with colleagues from their own experimental group and placed in two separate rooms for better monitoring. They had limited internet access to minimize distractions (instant messaging, e-mail, etc.) and to control experimental variables. Nevertheless, they had access to the necessary resources to conduct the experiment successfully:

- **Experiment tracker wiki.** This wiki contained instructions on how to start the experiment and a description of the consecutive tasks to be performed, its requirements and goals. It registered task completion times and monitored the experiment progression.
- **Documentation wiki.** To learn about the OGHMA framework, a documentation wiki was provided with enough contents to enable the effective coverage of the requirements presented by each task. This wiki was, in every way, an instantiation of the DRIVER platform, except for the availability of the patterns and the learning cycle supporting plug-ins. During the Treatments phase (see Sec. 4.4.2) each group would be given access to these resources according to their experiment research goals.

- **OGHMA Visual Studio Solution package.** The OGHMA framework was developed in C#, and optimized for use on Microsoft's Visual Studio IDE. Therefore, a Solution package was provided to the students (and referenced by the documentation), hopefully serving as a more quick and easy platform for development.

4.4. *Protocol*

This section describes the experiment protocol and phases (see Fig. 4). After divided into groups, the students were submitted to a pre-experiment questionnaire to establish their initial state. Then, each group undertook a treatment phase to condition their experiment environment accordingly and, after going through a series of tasks, a post-experiment questionnaire was conducted to collect results.

4.4.1. *Pre-questionnaires*

The first phase of the experiment started by handing out a questionnaire to the students. The questionnaires were designed using a Likert scale [31]. This psychometric bipolar scaling method contains a set of Likert items, or statements, which the respondent is asked to evaluate according to any kind of subjective or objective criteria, thus measuring either negative or positive response to the statement. For all the questionnaires in this experiment (both pre- and post-), the Likert items had a five-point format: (1) strongly disagree, (2) somewhat disagree, (3) neither agree nor disagree, (4) somewhat agree, and (5) strongly agree.

Pre-questionnaire A. The pre-experiment questionnaire A was used to ascertain the students background and general profile in order to screen out possible differences amongst the students regarding their basic skills. It also served to confirm their (non-)acquaintance with the OGHMA framework.

Pre-questionnaire B. The pre-experiment questionnaire B extended pre-experiment questionnaire A to include items used to ascertain what knowledge the subjects had about the OGHMA framework, at the start of the experiment.

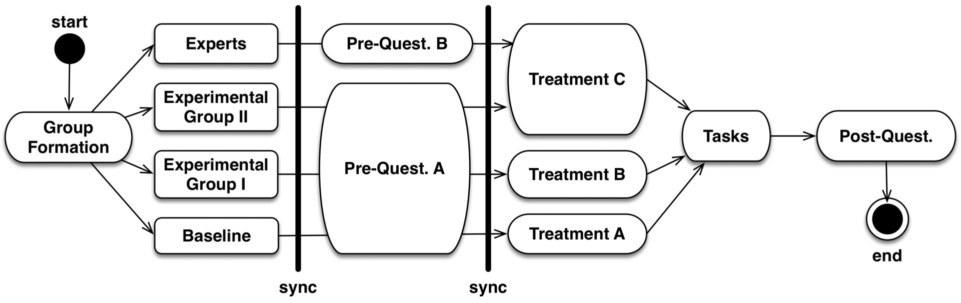


Fig. 4. Experiment protocol and phases.

4.4.2. *Treatments*

After the pre-questionnaires phase, the students were subject to a treatment phase, where each group was introduced to their own experiment environment. This is where the groups actually diverge regarding their experiment research goals. Each treatment is described next.

Treatment A. This treatment introduced the experiment environment as described in Sec. 4.3, but without the patterns or learning cycle supporting plug-ins.

Treatment B. This treatment extended treatment A by allowing the students to have access to the patterns and to go through them before advancing to the next phase.

Treatment C. This treatment extended treatment B by providing the experiment environment with the whole DRIVER platform. The DRIVER knowledge base already had a few learning paths captured in a previous trial run session where a framework expert performed the same experiment protocol. This expert had previous knowledge of the framework, but no acquaintance with the documentation wiki. The students were shown a demonstration video with a quick tutorial on how to use the plug-ins and their purpose, using a different case scenario. This way the students were not biased by any clues on what documentation to look for in order to perform the experiment tasks.

4.4.3. *Tasks*

At this point, all the groups were ready to start executing the tasks that would lead them to use the framework. It was not disclosed how many tasks were there, merely the goal to be effective and as less time-consuming as possible. Tasks were sequential, "unlocking" the next, only upon completion. The tasks were mainly focused on assessing how efficiently the students could incrementally build an information system within four iterations. These can be seen in Fig. 5, although the fourth iteration required no visual diagram aid (and thus not appearing in the figure) but only its textual description. These tasks had already been used in another experience [12] regarding the OGHMA framework, so they were adapted to fit this experiment research goals. The description of each iteration is presented next.

Iteration 1. The first iteration was designed to yield a very simple system. Only a single screen would be needed to view and edit the information. There was no polymorphism, shared aggregations or any type of conditional rules and the whole system could be roughly stored in two database tables. The purpose was merely to make first contact with the framework with no excessive task complexity.

Iteration 2. The second iteration was designed to make the students dig deeper into the framework. There was the need for more screens (due to indexation) and the number of database tables could grow from two to four, but with minimal modification to the existing artifacts.

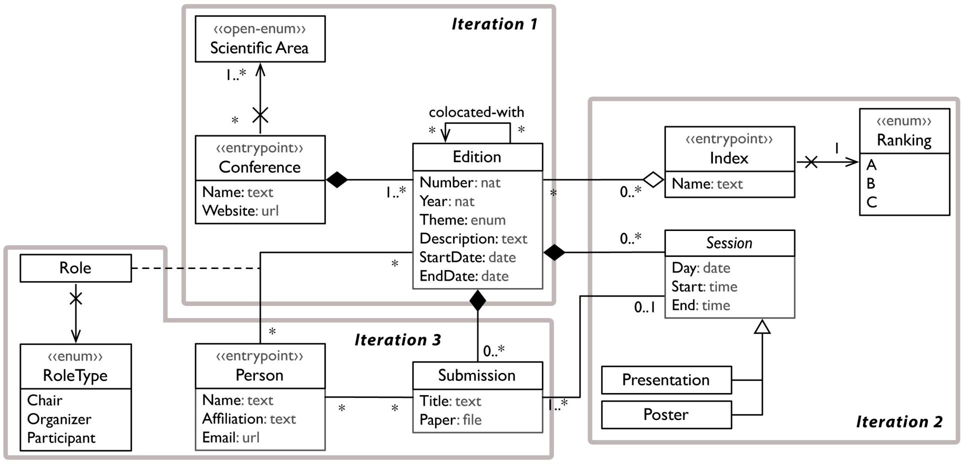


Fig. 5. Iteration-based incremental system development diagram presented to the subjects.

Iteration 3. The third iteration merely increased the complexity of the system, requiring more elaborate user-interaction and conditional rules, involving shared many-to-many relationships and relationship properties. Again the students would have to go deeper into the framework in order to cope with these requirements.

Iteration 4. This final iteration posed a new challenge: framework evolution. The students had no customization points to solve the proposed requirements. Therefore, they would have to go into the framework internal code and extend it to provide this new configuration ability. The goal was to put the students in direct contact with the framework code.

4.4.4. *Post-questionnaire*

At the end of the experiment, a questionnaire was handed out to the students in order to ascertain their framework knowledge intake and to screen potential threats to the validity of the results.

5. Analysis of the Results

This section reports the results of the experiment, regarding both qualitative (questionnaires) and quantitative (time results) metrics. As mentioned in Sec. 4.4, the subjects were given a pre-experiment questionnaire to screen out possible background and basic skills deviations (Sec. 5.2). During the experiment, the task completion time was recorded (Sec. 5.7) and afterwards, a post-experiment questionnaire collected further data (see Table 4), to evaluate possible threats to the validity of the results (Secs. 5.3–5.5) and the amount of framework knowledge intake (Sec. 5.6).

5.1. *Statistical relevance*

To provide statistical relevance in the analysis of the questionnaires items, the results are interpreted as such: let the null hypothesis be denoted as H_0, the alternative hypothesis as H_1, the baseline group as G_b, the experimental group[d] as G_e, and ρ the probability estimator of wrongly rejecting the null hypothesis. Then, the alternative hypothesis are either: (i) $H_1 : G_e \neq G_b$, the experimental group differs from the baseline, (ii) $H_1 : G_e < G_b$, the measure in the experimental group is lower than the baseline, or (iii) $H_1 : G_e > G_b$, the measure in the experimental group is greater than the baseline. The outcomes of the two treatments were compared for every answer using the non-parametric, two-sample, rank-sum Wilcoxon–Mann–Whitney test [21], with $n_1 = 6$ and $n_2 = 4$.[e] The significance level for all tests was set to 5%, so probability values of $\rho \leq 0.05$ are considered *significant*, and $\rho \leq 0.01$ considered *highly significant*.

5.2. *Background*

The goal of pre-questionnaire A was to provide an objective comparison between the technical background of each group. It was composed of the following questions:

I have considerable experience...

- BG1.1 ... using frameworks.
- BG1.2 ... analyzing and specifying information systems.
- BG1.3 ... using object-oriented architecture design and implementation.
- BG1.4 ... with agile development methodologies.
- BG1.5 ... with classical development methodologies.
- BG1.6 ... with formal development methodologies.
- BG1.7 ... with UML class diagrams.
- BG1.8 ... with Visual Studio IDE.
- BG1.9 ... using wikis.
- BG1.10 ... with XML-based languages.
- BG2. I have never used or had contact with the OGHMA framework.

As can be seen in Table 3, an analysis of the results showed there was no significant difference ($\rho > 0.05$ in all cases) between the BL and EG1 and EG2. As such, any subjective difference among the participants regarding their basic skills was rejected.

5.3. *External factors*

Attempting to rule out possible threats to validation, concerning the experiment environment factors, the following questions were posed:

- EF1. I found the whole experience environment intimidating.

[d] The experimental group can be either EG1 or EG2 depending on what group is being analyzed.
[e] Both EG1 and EG2 have four subjects.

Table 3. Summary of the results between of pre-questionnaire A, including the values of the non-parametric significance MWW test.

	EG1		EG2		BL		MWW (EG1 versus BL)			MWW (EG2 versus BL)		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	H_1	W	ρ	H_1	W	ρ
BG1.1	3.25	1.26	3.75	0.96	4.00	0.63	$\neq$	16.5	0.257	$\neq$	19.5	0.610
BG1.2	4.00	0.82	4.00	0.82	3.50	0.55	$\neq$	28.5	0.352	$\neq$	28.5	0.352
BG1.3	4.50	0.58	4.50	0.58	4.83	0.41	$\neq$	18.0	0.476	$\neq$	18.0	0.476
BG1.4	3.50	0.58	3.50	0.58	3.33	0.82	$\neq$	32.0	0.914	$\neq$	32.0	0.914
BG1.5	3.75	0.50	3.25	0.96	3.83	0.75	$\neq$	21.5	0.914	$\neq$	18.0	0.476
BG1.6	3.00	0.82	3.50	1.00	3.00	0.00	$\neq$	22.0	1.000	$\neq$	27.0	0.257
BG1.7	4.25	0.50	4.25	0.96	4.50	0.55	$\neq$	19.0	0.610	$\neq$	20.5	0.762
BG1.8	3.00	1.41	4.00	0.82	3.83	1.60	$\neq$	17.0	0.352	$\neq$	21.0	0.914
BG1.9	4.00	0.00	4.25	0.96	4.33	0.52	$\neq$	18.0	0.476	$\neq$	22.0	1.000
BG1.10	3.25	0.96	3.50	0.58	4.17	0.98	$\neq$	16.0	0.257	$\neq$	17.0	0.352
BG2	4.75	0.50	5.00	0.00	4.83	0.41	$\neq$	21.0	0.914	$\neq$	31.0	0.762

- EF2. I enjoyed programming and developing in the experiment.
- EF3. I would work with my partner again.
- EF4. I kept getting distracted by other colleagues outside my group.

As seen in Table 4, the impact of external factors was not different among the groups as to influence the experiment results.

5.4. *Overall satisfaction*

Questions regarding performance, comfort and feel for the collaborative environment were presented, as subjective results for later comparison with the expected objective results. Those questions were:

- OVS1. Overall, this particular setup was suitable for solving every task presented.

Table 4. Summary of the results of the post-questionnaire (except for framework knowledge questions), including the values of the non-parametric significance MWW test.

	EG1		EG2		BL		MWW (EG1 versus BL)			MWW (EG2 versus BL)		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	H_1	W	ρ	H_1	W	ρ
EF1	3.00	1.15	4.50	0.58	3.33	1.37	$\neq$	20.00	0.762	$\neq$	26.00	0.171
EF2	3.00	0.82	3.75	0.50	3.00	0.89	$\neq$	22.00	1.000	$\neq$	27.00	0.257
EF3	4.50	0.58	3.50	0.58	4.50	0.84	$\neq$	21.00	0.914	$\neq$	14.00	0.114
EF4	1.50	0.58	1.00	0.00	1.00	0.00	$\neq$	27.00	0.256	$\neq$	22.00	1.000
OVS1	4.00	0.00	3.50	0.58	2.50	0.55	$>$	24.00	$\underline{0.005}$	$>$	14.50	$\underline{0.048}$
OVS2	2.75	0.96	3.00	0.82	1.50	0.84	$>$	29.50	$\underline{0.043}$	$>$	27.00	$\underline{0.033}$
OVS3	4.50	0.58	4.50	0.58	4.67	0.52	$<$	20.00	0.881	$<$	20.00	0.881
OVS4	1.75	0.50	2.25	0.50	2.50	1.05	$<$	16.50	0.953	$<$	20.00	0.738
DP1.1	2.50	1.00	4.25	0.50	1.67	1.21	$<$	26.00	0.971	$<$	22.50	1.000
DP1.2	1.75	0.50	1.75	0.50	1.50	0.84	$<$	29.50	0.833	$<$	29.50	0.833
DP1.3	3.75	0.96	3.75	1.26	4.17	1.17	$<$	18.50	0.795	$<$	19.00	0.853
DP1.4	5.00	0.00	4.00	1.15	5.00	0.00	$<$	22.00	1.000	$<$	16.00	1.000

- OVS2. I found the documentation available to be sufficient.
- OVS3. I felt the need to have access to more information on how to use the framework.
- OVS4. Despite my experience, the tools available, excluding OGHMA, delayed my work considerably.

As expected, the results showed that there was a significant difference ($\rho < 0.01$) between BL and EG1 and EG2 in questions OVS1 and OVS2. It indicated that the proposed collaborative environment provided a greater sense of accomplishment and satisfaction when compared to just having the documentation.

5.5. *Development process*

Questions DP1.1 to DP1.4, intended to ascertain how hard it was to complete each of the task presented, as to measure the impact of DRIVER in the development process. The template question was: "It was hard to find out how to use the framework to complete iteration X" (where X was replaced by the iteration number).

In none of the desired hypothesis did the scores produce any relevant statistical results, having some items, even, produced unexpected scores.

In an overall analysis, it can be stated that in the case of BL versus EG1, iteration 1 revealed to be an easy task, getting easier in iteration 2, but increasingly more difficult when it came to iteration 3 and 4 (where the non-completion of this iteration, led to the top score of 5). In the case of BL versus EG2, the results are similar to the above stated, with an increased score for iteration 1.

The overall scores of this group of questions were unexpected. This led to a follow-up informal interview with the students to try to understand their reasons for answering such scores. The main conclusion from this interview is that their interpretation of the items led them to answer not so much about the usage of the collaborative environment, but more about the complexity of the OGHMA framework and their subjective analysis of their own performance. As such, no evidence can be assumed from this group of questions, as the answers do not express its intended purpose. Nevertheless, when asked about iteration complexity, they noted that Iteration 1 (because it was the first) and Iteration 4 (no one was able to complete) were the most complex, ordering Iteration 3 as mildly complex and Iteration 2 as the less complex (especially after tackling with Iteration 1).

5.6. *Framework knowledge*

In order to measure the increase in framework knowledge, a set of 17 items was devised and presented to the subjects at the end of the experiment. These questions intended to ascertain how much correct information about the framework the participants had acquired. It was assumed that all groups (except the Experts) had no prior knowledge of the framework whatsoever, as corroborated by item BG2.

5.6.1. *Categories*

According to [3], framework knowledge can be divided into layers, ranging from more abstract to more concrete information. Not only was it relevant to measure the amount of framework knowledge acquired, but also at what depth the subjects went in their learning of the framework. As such, framework knowledge can be divided into the following seven categories, each representing an abstraction layer:

- *Overview* (OV). This category is intended to communicate the purpose of the framework to potential users, in a clear and concise way.
- *Domain* (DM). This category defines the application domain covered by the framework, namely the products that can be developed with the framework, their variability aspects (hotspots), and how the framework can and should be reused.
- *Components* (CP). This category formally defines a black-box view, i.e. the properties and behavior of the products that can be developed.
- *Design* (DN). This category presents the design principles of the framework, and describe its micro-architectures and mechanisms of cooperation between components.
- *Public view of the implementation* (PB). This category represents an external view of the implementation of framework components, its collaborations, roles, interfaces, and classes.
- *Protected view of the implementation* (PT). This category represents the view available for developers of components through extension of classes provided by the framework and its contracts.
- *Private view of the implementation* (PV). This category presents a white-box view over the implementation of the framework, usually in the form of source code.

The items presented in the questionnaire tried to cover all these categories and had a true/false statement-like form. They were then shuffled, so its natural order would not bias the subjects when answering.

5.6.2. *Results*

The relevance of an item-to-item analysis of the scores is not so much important as the total amount of knowledge the subjects acquired. So, the results are shown aggregated and processed in two ways: (i) total knowledge acquired and (ii) total knowledge acquired by category.

When answering a true or false statement, using a five-point format Likert scale, the scores not only show the answer (strongly disagree (1) as false and strongly agree (5) as true) but also the confidence level of the respondent. The closer the answer gets to the boundaries of the scale, the more certain the subject is of the answer (being neither agree nor disagree (3) not knowing the answer). The scores were then processed and converted into distances from the correct answer, e.g. a score of 2 for a true statement (5) converts into a distance of 3 ($|5 - 2| = 3$), whereas for a false

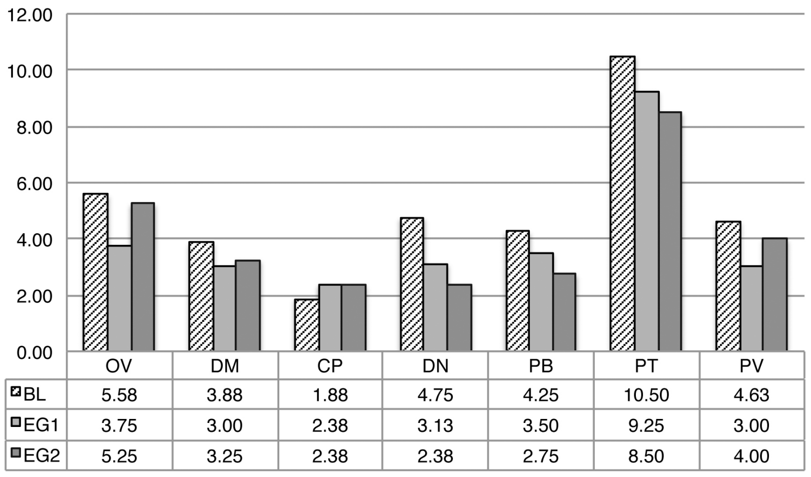
Table 5. Framework knowledge group statistics.

Group	N	Mean	Std. deviation	Std. error mean
BL	17	2.087	0.5598	0.1358
EG1	17	1.647	0.5234	0.1269
EG2	17	1.676	0.5431	0.1317

Table 6. Independent Samples Tests for Framework Knowledge. The first two columns are the Levene's Test for Equality of Variances, showing a significance greater than 0.05 (Sig). The other three columns are the t-test for Equality of Means. Since we can assume equal variances, the 2-tailed values of 0.024 and 0.047 allow us to conclude that there is a **statistically significant difference** between the two conditions, in both tables.

	F	Sig.	t	df	Sig. (2-tailed)
(a) Baseline versus Experimental Group 1					
Eq. Var. Assumed	0.215	0.646	2.364	32.00	0.024
Eq. Var. Not Assumed			2.364	31.86	0.024
(b) Baseline versus Experimental Group 2					
Eq. Var. Assumed	0.041	0.841	2.071	32.00	0.047
Eq. Var. Not Assumed			2.071	31.98	0.047

statement it converts into a distance of 1 ($|1-2|=1$) and so forth. Items the subjects did not know the answer (3) would always contribute the same distance (2). Finally, an average of the scores for each item was computed and, as done for the initial students' average pre-experiment evaluation, an independent samples t-test was conducted to compare the averages of the items between the Baseline and Experimental Groups 1 and 2. These results can be seen in Tables 5 and 6. A comparison for the framework knowledge distances for each category can be seen in Fig. 6.



	OV	DM	CP	DN	PB	PT	PV
▨ BL	5.58	3.88	1.88	4.75	4.25	10.50	4.63
░ EG1	3.75	3.00	2.38	3.13	3.50	9.25	3.00
▓ EG2	5.25	3.25	2.38	2.38	2.75	8.50	4.00

Fig. 6. Framework knowledge distances results.

The results provide evidence that the collaborative environment contributes to an increase on framework KA, thus supporting the hypothesis that it helps novices on learning about a framework.

5.7. *Performance and effectiveness*

During the experiment, the duration of each group took to complete each iteration was recorded. At the end, these results were processed and corrected, considering the effectiveness of the deliverables, so that certain quality-related time deviations (e.g. failure to comply to requirements, code standards, implementation variations, etc.) could be minimized and the reliability of the results increased.

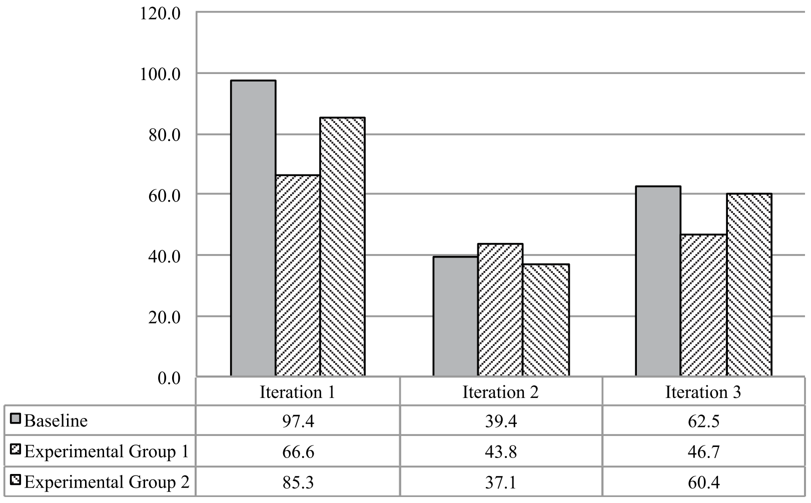
All deliverables were inspected for quality and effectiveness and graded, rendering a time penalty accordingly. The deliverables were graded with the following scale: grade A (no time penalty), for deliverables that covered all the requirements and presented the expected implementations; grade A- (5 min time penalty) for deliverables that slightly deviated from the expected implementation, nevertheless covered all the requirements; grade B, (10 min time penalty) for deliverables that somewhat deviated from the intended implementation, although still covering the requirements; grade B-, (15 min time penalty) for deliverables that failed to cover one or more requirements, although the coding of a possible solution was present. The grading of the deliverables can be seen in Table 7.

As an overall analysis, the results (see Fig. 7) indicate that there were better time performances from Experimental Groups 1 and 2, in comparison to the Baseline group. The longer time EG2 took when compared to EG1 can be explained by the overhead taken using a new tool (the learning cycle plug-ins). Iterations 1 and 3 corroborate this.

Regarding Iteration 2, there is a more even set of results, where, oddly EG1 took a slightly bit more than both BL and EG2. It is believed that the short finishing time they took in Iteration 1 may have rendered them overconfident for tackling with this iteration, thus affecting their time performance. On the other hand, despite slim, EG2 performed better than BL. As for Iteration 4, only a couple of groups finished within the expected time frame for the experiment, rendering their results useless for

Table 7. Deliverables grades.

	Iterations			
	1	2	3	4
Baseline Pair 1	B	B	A−	–
Baseline Pair 2	B	B	B−	–
Baseline Pair 3	B	A−	B	–
Experimental Group 1 Pair 1	A−	A	A−	–
Experimental Group 1 Pair 2	B	B	A	–
Experimental Group 2 Pair 1	B	B	B	–
Experimental Group 2 Pair 2	A−	B	A−	–



	Iteration 1	Iteration 2	Iteration 3
▢ Baseline	97.4	39.4	62.5
▨ Experimental Group 1	66.6	43.8	46.7
▧ Experimental Group 2	85.3	37.1	60.4

Fig. 7. Iteration completion time results (average per group). Units in minutes.

analysis. This is explained by the increased complexity of the iteration, combined with the already long experiment duration and tiredness of the groups.

It is interesting to observe that when complexity and extensiveness increase, the results are better. This can be seen in Iteration 1, when there is a first contact with the framework and then again in Iteration 3. The complexity gap between Iterations 1 and 2 is not so great, so the performance is somewhat similar. But when one turns up the heat, (from Iteration 2 to Iteration 3) the tools step in to aid on performing better.

5.8. *Experts group analysis*

The purpose of the Experts group was to study the impact the collaborative approach might have on subjects with prior knowledge of the framework, although not actively using it. For this experiment, the subjects had been in contact with the framework around six months prior to the experiment and during a period of three months. During this time, they engaged in instantiation and evolution tasks. The inactivity period allowed the framework knowledge acquired to decay, therefore, regaining contact with the framework, after a while, would generate different cognitive needs than the novice users that had never used the framework.

The results proved inconclusive as to where the collaborative approach helps experts on improving framework learning. Despite the expected better time performance when compared with the other groups, the KA results did not indicate that the approach helped the Experts group in their (re-)learning of the framework.

Despite these results, other indicators where captured, namely the usage of the collaborative approach to capture their own knowledge of the framework. Although not disclosed to them, they quickly discovered that some (if not all) the subjects of the Experts group were sharing the same knowledge base of learning paths, so they spent some time adding their own and rating others.

It is believed that without the burden of the overhead of learning a new framework, the subjects spent some time using the DRIVER platform to capture their learning paths and improve the knowledge-base with their own expertise. In a later follow-up session, they provided lots of useful feedback on how to improve the collaborative environment.

6. Threats to Validation

The outcome of validation is to gather enough scientific evidence to provide a sound interpretation of the results. Validation threats are issues and scenarios that may distort that evidence and thus incorrectly support (or discard) expected results. Each validation threat should be expected and addressed *a priori* in order to yield unbiased results or, at least, minimized *a posteriori* with effective counter-measures.

The following expected validation threats were discarded through both pre- and post-experiment questionnaires:

- *Misunderstanding of the given tasks.* Because the tasks relied on textual specifications and UML diagrams, is was necessary to ensure that the participants correctly interpret them (item BG1.7).
- *Insufficient skills to execute the tasks.* The tasks required participants to have the necessary skill to build and evolve information systems, namely knowing how to work with the given programming language, IDE and database engine (items BG1.2, BG1.3, BG1.8 and BG1.10).
- *Overhead due to lack of experience with the new tools.* When presented with new tools, these will, unavoidably, introduce overhead into the development process. This overhead was expected and compensated when interpreting the observed results. Nevertheless, the evaluation of the participants subjective feel over this overhead had to be discarded so that the tools would not pose as a relevant threat beyond expectations (item OVS4).
- *Experiment-related factors.* Knowingly being part of an experiment, changes the mood and may be an inhibitor of normal development. The performance may be conditioned by the feel of being observed and judged (item EF1).
- *Team factors.* Despite the forming of pairs being handed to the participants as to alleviate the possibility of having conflicting partners, it was necessary to make sure that the final grouping was not a threat to validity (item EF3).
- *Lack of motivation.* Due to the length of the tasks (experiment went beyond 3 h), and the fact that there was no compensation to individuals participating in the experiment, the lack of motivation could hinder the outcome (item EF2).
- *Inter-group competition.* In an open-space setting and knowing all groups are undertaking the same experiment, the ability to have feedback on how others are

performing may influence groups differently. Different people react differently to pressure. Therefore, there was the need to discard this threat by ascertaining if this pressure was an issue (item EF4).

A threat not completely discarded was assertion of task completion. There was no deterministic mechanism (e.g. automated tests) to verify if each iteration was completed. It has to be done by manual inspection and testing. This forced a post-investigation of the deliverables to compensate for the effectiveness deviations that might had occurred. In future experiments, this deterministic mechanism should exist to assure the effectiveness of the deliverables.

The power of this study could also be improved by (i) increasing the number of participants, and (ii) switching the participants roles, where individuals in the experimental groups would undergo the baseline process and vice-versa.

7. Related Work

Regarding framework understanding, most solution proposals found in the literature converge to produce and enhance existing documentation, such as design patterns [17], pattern languages [25], cookbooks [29], hooks [16], exemplars [18] and minimalist documentation [2]. 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 [38], 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 organization; lack of design choice rationale and shallowness in understanding the framework internals. DRIVER not only relies on examples, but it also allows for the learner to choose which examples are more suited to his/hers learning needs. Not only community-driven learning copes with divergent search for knowledge, but also the best practices imbued into the platform *drive* the learner into a more personalized learning experience.

In [35], 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 repeating the same task over time. Yet, a more proficient developer has to engage on high level framework knowledge learning. DRIVER relies on a community-mediated selection of the best learning paths, as a way to mitigate the time needed to acquire knowledge. Thus, it replaces repetition by the same person with consensus by a community of heterogeneous developers.

In [27], 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. DRIVER uses a pattern language for understanding a framework as a set of best practices for coping with framework learning. This language, not only tackles mapping problems, but it addresses all four fundamental problems of framework reuse, additionally regarding cognitive aspects of the learner and knowledge keeping issues.

Several studies have been led by Hou *et al.* [22–24], regarding framework usage and understanding. They unveiled issues regarding design (tight-coupling, delocalized concerns, excessive special cases), documentation (doc-driven understanding fares better than 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 showed that novice learners tend to favor functional aspects over non-functional and that spending some time (upfront) 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. The authors took most of these requirements into consideration when developing DRIVER, building upon the existing research, and adding extra collaboration features into the learning knowledge management layer.

8. Conclusions and Future Work

This paper presented DRIVER, a collaborative environment that supports the framework learning process. Not only it relies on an easy, shared, lightweight, editable platform (wiki), that provides documentation artifacts about the framework, but promotes KA 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 plug-ins 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 within a controlled experimental environment to validate the usefulness of the DRIVER environment.

MSc students were divided into two major groups: one of novices and another of Experts. The novices were then divided into three groups (Baseline, Experimental Group 1 and Experimental Group 2) to which their background and basic skills were screened through a pre-experiment questionnaire, guaranteeing no statistical deviation.

All three groups went through the development of the same technical tasks using a previously unknown framework, all using different development settings to enable a comparison between having, or not having, the proposed collaborative approach. The Experts group already knew the framework, but served to assess the usefulness of the collaborative approach. A post-questionnaire and their time track were used to assess the outcome of the experiment.

The final results support the hypothesis that the collaborative approach helps novices to more effectively learn about a framework. Although more evident between the Baseline and Experimental Group 1, than Experimental Group 2, both experimental groups fared better at both time spent and knowledge intake, when compared to the Baseline group. No evidence was collected when it comes to experts, but it is believed that, it allows experts to easily capture and share their learning knowledge about the framework.

Some threats to this validation were identified and further discarded by analyzing the results in pre- and post-experiment questionnaires and due to the nature of the experimental setting. Not all original hypothesis were supported though, and some borderline threats which emerged after the experiment can help refine further studies.

As future work, enhancements to the DRIVER platform are currently under way, focusing on improving recommendation heuristics, increase learner's profile awareness and convergence to the semantic web. These enhancements are a first step into the integration of the DRIVER tool into a software knowledge management platform, currently being devised and developed by the authors [8, 7], while incorporating other concerns on communicating software knowledge on a community level [41].

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.

References

1. Esse wiki, http://softeng.fe.up.pt/esseWiki. Accessed 31 March 2016.
2. A. Aguiar, Framework Documentation — A Minimalist Approach, Ph.D. thesis, FEUP, September 2003.

3. A. Aguiar and G. David, Patterns for documenting frameworks – Part I, in *VikinPLoP'2005*, 2005.
4. A. Aguiar and G. David, Patterns for effectively documenting frameworks, in *Transactions on Pattern Languages of Programming II*, eds. J. Noble and R. Johnson (Springer-Verlag, Berlin, Heidelberg, 2011), pp. 79–124.
5. G. Butler, A reuse case perspective on documenting frameworks, in *Proc. Fifth Asia Pacific Software Engineering Conf.*, 1998.
6. J. C. Carver, L. Jaccheri, S. Morasca and F. Shull, A checklist for integrating student empirical studies with research and teaching goals, *Empir. Softw. Eng.* **15**(1) (2010) 35–59.
7. F. Correia, Documenting Software using Adaptive Software Artifacts, Ph.D. thesis, FEUP, December 2014.
8. F. Correia, H. Ferreira, N. Flores and A. Aguiar, Incremental knowledge acquisition in software development using a weakly-typed wiki, in *5th Int. Symp. Wikis*, 2009.
9. L. P. Deutsch, Design reuse and frameworks in the smalltalk-80 system, *Software Reusability: Vol. 2, Applications and Experience*, 1989, pp. 57–71.
10. M. Fayad and D. Schmidt, Object-oriented application frameworks, *Commun. ACM* **40** (10) (1997) 32–38.
11. R. Felder and J. Spurlin, Applications, reliability, and validity of the index of learning styles, *Int. J. Eng. Educ.* **21**(1) (2005) 103–112.
12. H. Ferreira, Adaptive Object-Modelling: Patterns, Tools and Applications, Ph.D. thesis, University of Porto, Faculty of Engineering, 2011.
13. N. Flores and A. Aguiar, Patterns for framework understanding, in *15th Pattern Languages of Programming Conf. (PLoP'08)*, 2008.
14. N. Flores and A. Aguiar, Understanding frameworks collaboratively: Tool requirements, *Int. J. Adv. Softw.* **3** (2010) 114–135.
15. N. Flores and A. Aguiar, Driver — A platform for collaborative framework understanding. in *30th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2015.
16. G. Froehlich, H. Hoover, L. Lui and P. Sorenson, Hooking into object-oriented application frameworks, in *Proc. 19th Int. Conf. Softw. Eng.*, 1997, pp. 491–501.
17. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns — Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995).
18. D. Gangopadhyay and S. Mitra, Understanding frameworks by exploration of exemplars, in *Proc. CASE-95*, 1995, pp. 90–99.
19. M. Goulao and F. Brito Abreu, Modeling the experimental software engineering process, in *Proc. 6th Int. Conf. Quality of Information and Communications Technology*, 2007, pp. 77–90.
20. T. Gruber, Collective knowledge systems: Where the social web meets the semantic web, *J. Web Semant.* (2007).
21. M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods* (Wiley-Interscience, 1999).
22. D. Hou, Investigating the effects of framework design knowledge in example-based framework learning, in *IEEE Int. Conf. Software Maintenance*, 2008, pp. 37–46.
23. D. Hou and L. Li, Obstacles in using frameworks and apis: An exploratory study of programmers' newsgroups discussions, in *IEEE 19th Int. Conf. Program Comprehension*, 2011.
24. D. Hou, K. Wong and H. J. Hoover, What can programmer questions tell us about frameworks? in *Proc. 13th Int. Workshop on Program Comprehension*, 2005, pp. 87–96.
25. R. Johnson, Documenting frameworks using patterns, in *Proc. OOPSLA'92*, 1992, pp. 63–76.

26. R. E. Johnson and B. Foote, Designing reusable classes, *J. Object-Oriented Program.* **1**(2) (1988) 22–35.
27. D. Kirk, M. Roper and M. Wood, Identifying and addressing problems in framework reuse, in *Proc. 13th Int. Workshop on Program Comprehension*, 2005, pp. 77–86.
28. B. Kitchenham, H. Al-Khilidar, M. Ali Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang and L. Zhu, Evaluating guidelines for reporting empirical software engineering studies, *Empir. Softw. Eng.* **13**(1) (2008) 97–121.
29. G. E. Krasner and S. T. Pope, A cookbook for using the model-view-controller user interface paradigm in smalltalk-80, *J. Object-Oriented Program.* **1**(3) (1988) 26–49.
30. T. D. LaToza, G. Venolia and R. DeLine, Maintaining mental models: A study of developer working habits, in *Proc. Int. Conf. Software Engineering*, 2003.
31. R. Likert, A technique for the measurement of attitudes, *Arch. Psychol.* **22**(140) (1932) 1–55.
32. Y. I. Liou, Collaborative knowledge acquisition, *Expert Syst. Appl.* **5**(1–2) (1992) 1–13.
33. Y. I. Liou, Knowledge acquisition: Issues, techniques and methodology, in *SIGMIS Database 23*, Vol. 1, 1992, pp. 59–64.
34. S. Mittal and C. L. Dym, Knowledge acquisition from multiple experts, *Al Mag.* **6**(2) (1985) 32–36.
35. M. Morisio, D. Romano and I. Stamelos, Quality, productivity, and learning in framework-based development: An exploratory case study, *IEEE Trans. Softw. Eng.* **28**(9) (2002) 876–888.
36. K. Nakakoji, K. Ohira and Y. Yamamoto, Computational support for collective creativity, *Knowl. Based Syst. J. Elsevier Sci.* **13**(7–8) (2000) 451–458.
37. K. Nakakoji, Y. Yamamoto and Y. Ye, Supporting software development as knowledge community evolution, in *Proc. CSCW Workshop on Suporting the Social Side of Large Scale Software Development*, 2006.
38. F. Schull, F. Lanubile and V. Basil, Investigating reading techniques for object-oriented framework learning, *IEEE Trans. Software Eng.* **26**(11) (2000) 1101–1118.
39. F. Shull, J. Singer and D. I. K. Sjøberg, *Guide to Advanced Empirical Software Engineering* (Springer-Verlag, 2007).
40. J. Surowiecki, *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations* (Anchor Publishing, 2004).
41. C. Treude and M.-A. Storey, Effective communication of software development knowledge through community portals, in *Proc. 8th Joint-Meeting of the ESEC/FSE*, 2011, pp. 91–101.
42. G. Uddin and M. P. Robillard, How api documentation fails, *IEEE Softw.* **32**(4) (2015) 68–75.
43. W. Wang and M. W. Godfrey, Detecting api usage obstacles: A study of ios and android developer questions, in *10th IEEE Working Conf. Mining Software Repositories*, 2013, pp. 61–64.
44. D. A. Waterman, *A Guide to Expert Systems* (Addison-Wesley, 1986).
45. J. Yoder, F. Balaguer and R. Johnson, Adaptive object models for implementing business rules, *Urbana* (2001).
46. M. V. Zelkowitz and D. R. Wallace, Experimental models for validating technology, *IEEE Comput.* **31**(5) (1998) 23–31.