Patterns for Understanding Frameworks

Authors

Nuno Flores, Ademar Aguiar
FEUP & INESC Porto, Universidade do Porto
E-mail: nuno.flores@fe.up.pt, ademar.aguiar@fe.up.pt.

Learning and understanding a framework is usually a major obstacle to its effective reuse. Before being able to use a framework successfully, users often need to surpass a steep learning curve by spending a lot of effort understanding its underlying architecture and design principles. This is mainly due to users having to understand not only single isolated classes, but also complex designs of several classes whose instances collaborate for many different purposes, and using many different mechanisms. In addition, frameworks are also full of delocalized plans, and use inheritance and delegation intensively, which makes their design more difficult to grasp at a first glance. How to obtain the necessary information from the framework itself and its accompanying documentation is the main problem with framework understanding. Considering its importance, this paper presents an initial attempt to capture, in the pattern form, a set of proven solutions to recurrent problems of understanding frameworks. The fundamental objective of this work is to help non-experts on being more effective when trying to learn and understand object-oriented frameworks.

Introduction

The introduction of reuse in a software development process implies splitting the traditional software life cycle into two interrelated cycles: one focused on *developing reusable assets*, and another focused on *searching and reusing reusable assets* already developed.

A framework is a reusable design together with an implementation. It consists of a collection of cooperating classes, both abstract and concrete, which embody an abstract design for solutions to problems in an application domain [15][16][17].

In the particular case of framework-based application development, the traditional life cycle can be organized in: a *framework development* life cycle devoted to build frameworks, corresponding to the abstraction phase of software reuse; and an *application development* life cycle (also known as framework usage) devoted to develop applications based on frameworks, corresponding to the selection, specialization, and integration phases of software reuse.

Although the activities of framework development and application development are often separate and assigned to different teams, the knowledge to be shared between them is large, as the design of a framework for a domain requires considerable past experience in designing applications for that domain.

In application development, frameworks act as generative artefacts as they are used as a foundation for several applications of the framework's domain. This contrasts with the traditional way of developing applications, where each application is developed from scratch. The most distinctive difference between the traditional and the framework-based development of applications is the need to map the structure of the problem to be solved onto the structure of the framework, thereby forcing the application to reuse the design of the framework. The positive side of this is that we don't need to design the application from scratch. But, on the other hand, before starting application development, we need to understand the framework design, a task that sometimes can be very difficult and time-consuming, especially if the framework is large or complex, and is not appropriately accompanied with good documentation or training material.

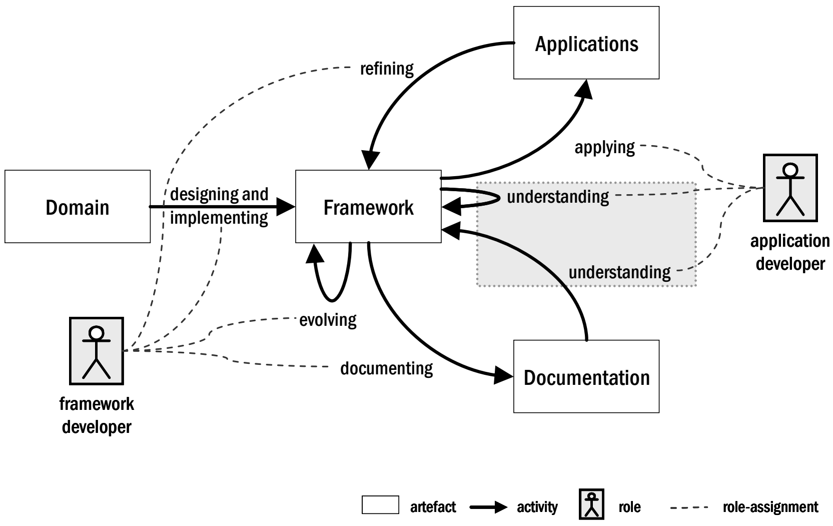

Figure 1 - Activities, artefacts and roles of framework-based application development.

Figure 1 shows a simplified view of framework-based application development that relates the artefacts, activities, and roles most relevant for the topic of the patterns presented in this paper – framework understanding. Because understanding frameworks is of major importance for application developers, in the figure, the activity is assigned exclusively to that role, but in fact it can be also relevant for framework selectors, original framework developers (especially of large frameworks), framework maintainers, and developers of other frameworks, although not with the same degree of importance.

In relation to the other activities depicted, there are pattern languages addressing their own challenges and problems, of which we refer "*Evolving frameworks: A pattern language for developing object-oriented frameworks*" [1], regarding design, implementation and evolution activities, and also "*Patterns for Documenting Frameworks*" [2][3][4], regarding documentation activities.

Therefore, this paper contributes patterns to a pattern language focusing on problems of understanding frameworks (white-box, black-box or gray-box), which must be properly managed before being able to use frameworks effectively.

White-box frameworks rely heavily on inheritance and dynamic binding in order to achieve extensibility. Although white-box reuse is the hardest way to use a framework, it is by far the most powerful.

Black-box frameworks are the easiest to use, because they are structured using object composition and delegation rather than inheritance. On the other hand, black-box frameworks are the most difficult to develop, because they require the definition of the right interfaces and hooks able to anticipate a wide range of application requirements.

Most real-world frameworks combine black-box and white-box characteristics, being thus called *gray-box* frameworks. They allow extensibility both by using inheritance and dynamic binding, as well as by defining interfaces. Gray-box frameworks are designed to avoid the disadvantages of black-box frameworks and white-box frameworks.

Although some of the problems here addressed could also be common to large or complex software systems, frameworks are specifically designed to be easy to reuse, thus adding special needs from the point of view of learning and understanding.

Pattern language

The pattern language comprises a set of interdependent patterns aiming to help users become aware of the problems that they will typically face when starting to learn and understand frameworks. These patterns are targeted for framework users especially novices. The patterns were mined from existing literature, lessons learned, and expertise on using frameworks, based on previous studies and literature reviews of the authors on the topic [5][6].

The pattern language outlines a path commonly followed when learning and understanding a framework. As many frameworks can be very difficult to learn and understand, completely or in detail, these patterns aim to expose the tradeoffs involved in the process of understanding a framework, and to provide practical guidelines on how to balance them to find the best learning strategy for each specific person (learner or framework user) and context.

The problems addressed by the patterns are basically raised by the following questions:

- What do I need to understand about the framework to accomplish my task? What kind of knowledge do I need? More concrete or abstract? At code level, design level, documentation level?
- How can I acquire the knowledge I need? Which learning strategy should I adopt? Which one is best for my needs?
- Which kinds of tools can I use to gather, organize, explore and preserve the knowledge I value most?

According to [14], framework reuse can be divided into categories according to the re-user's interests, whether a framework selector, an application developer, a framework maintainer, or a developer of other frameworks. These categories range from selecting, instantiating, flexing, composing, evolving and mining a framework. For the scope of the pattern language presented in this paper, only the most commonly used will be addressed: selecting, instantiating and evolving.

Patterns overview

To describe the patterns, we have adopted Christopher Alexander's pattern form: *Name-Context-Problem-Solution-Consequences* [7]. We've also added a *Rationale* section and a *See also* section, where known uses and further reading directions can be found. Before going to the details of each pattern, we will briefly overview the pattern language with each pattern's intent and a map (Figure 2) showing their relationships.

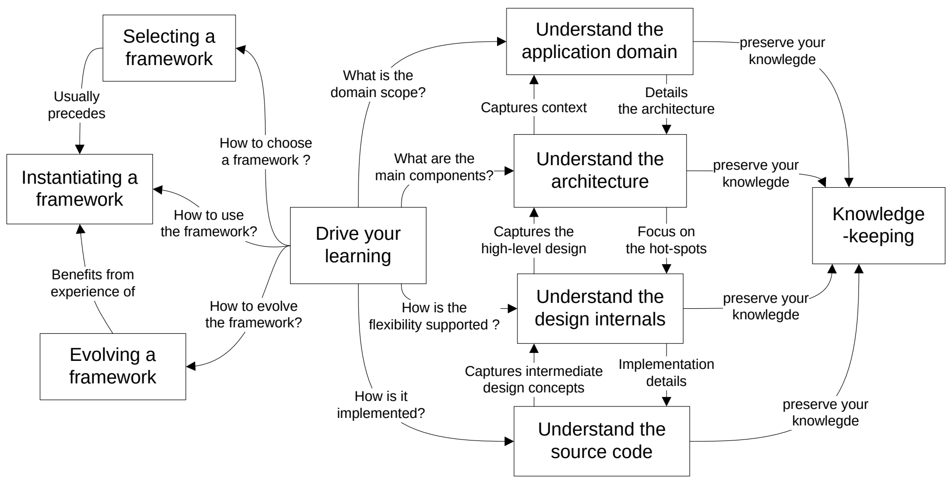
Selecting a framework
Usually precedes
Instantiating a framework
Benefits from experience of
Evolving a framework
How to choose a framework ?
How to use the framework?
How to evolve the framework?
Drive your learning
What is the domain scope?
What are the main components?
How is the flexibility supported ?
How is it implemented?
Understand the application domain
Captures context
Details the architecture
Understand the architecture
Captures the high-level design
Focus on the hot-spots
Understand the design internals
Captures intermediate design concepts
Implementation details
Understand the source code
preserve your knowlegde
preserve your knowlegde
preserve your knowlegde
preserve your knowlegde
Knowledge -keeping

Figure 2 – Framework understanding patterns and their relationships.

Selecting a framework. This pattern allows deciding whether or not to select a framework, after evaluating its appropriateness for an intended application domain.

Instantiating a framework. This pattern shows how to learn about instantiating a framework in order to implement an application.

Evolving a framework. This pattern shows what steps should be taken to learn how to evolve a framework.

Drive your learning. This pattern shows how to plan your learning process throughout the task of understanding a framework.

Knowledge-keeping. This pattern shows how to preserve the acquired knowledge about a framework.

Understand the application domain. This pattern enables the learner to know what is the application domain covered by the framework.

Understand the architecture. This pattern shows the learner how to find architectural knowledge about the framework.

Understand the design internals. This pattern tells the learner how to look for knowledge about the design internals of the framework.

Understand the source code. This pattern helps to understand how to identify where in the source code are the important parts that enable the developer to implement the application.

When referring to a related pattern within this pattern language, its name will appear in SMALLCAPS. Otherwise, if it's an "outside" pattern, it will appear in *SMALLCAPS ITALICISED* together with the respective reference.

Pattern **Selecting a framework**

You are someone (manager, project leader, developer) who is responsible for finding a solution for an application development project in a certain domain. You are about to select a framework that can help you to solve your problem.

Problem Framework selection consists of deciding whether to reuse or not a framework, after evaluating its appropriateness for an intended application in a specific domain.

What do you need to learn about a framework in order to select it effectively?

Forces **Effort**. You don't want to spend too much time learning what you need to know to effectively decide if a framework is selectable.

Certainty/Sureness. You need to be sure that the framework you're about to select covers, not only your application domain, but also all of your specific needs.

Documentation. The existing documentation may not give the necessary insight into the applicability of the framework.

Complexity. The more complex a framework is, the harder it is to understand.

Solution Start by quickly understanding the framework under consideration. Look for a short description of the framework's purpose, the domain covered, and an explanation of its most important features, preferably illustrated with examples.

In order to ascertain if a specific framework covers your domain requirements, you need to UNDERSTAND THE APPLICATION DOMAIN in a clear way, i.e., the domain covered by the framework, and the range of solutions for which the framework was designed and is applicable.

However, knowing the purpose of the framework is not always enough to ensure that all the problems may be met by this framework. It can be important to go deeper to UNDERSTAND THE ARCHITECTURE, UNDERSTAND THE DESIGN INTERNALS, or UNDERSTAND THE SOURCE CODE, until being sure of the framework's appropriateness for the problem at hand.

To be more effective, you may consider to DRIVE YOUR LEARNING according to your experience and specific requirements.

Consequences **Cost-effectiveness**. You quickly gain insight into the scope of the framework and its coverage of your specific needs. Going into detail gives you more accurate hints on how the framework is built and addresses your problems.

Narrow knowledge. Yes, it solves your specific problems, but that doesn't give you a whole grasp over what other specific problems it might address. Further investigation might be needed when new contextual-related problems arise.

Rationale When using frameworks, one of the key decisions that need to be made is whether or not the framework fits the application. Since frameworks can be complex, gaining a deep understanding of the framework to make that decision often requires the time consuming process of actually using the framework. Capturing information about the applicable domain of the framework is a way to ease this decision [38]. Limitations and design trade-offs about the framework can help to show what the framework can and cannot be used for. There will always be degree of uncertainty but that can be mitigated by existing documentation or the potential user will often perform experiments to increase their understanding of the framework and to evaluate its appropriateness to the new application.

See also In [18], Andrew Turner and Chao Wang had to evaluate a set of existing AJAX frameworks and select the most suited for their requirements. Their process relied on ascertaining that all the frameworks could cover their specific domain and high-level requirements. They had to dig deeper into the framework internals and even develop some prototypes to test if the framework could address and solve their specific issues.

In [19], Sheick *et al.* proposed and applied a criterion for ascertaining the suitability of a framework to a specific project. It relied on a set of areas to inspect, starting with the intended domain and evolving into detailed issues like the presence of design patterns and lower-level concerns such as error handling and degree of coupling. They then applied their criteria to characterize an existing framework for a transaction processing system implementation called jPOS ISO 8583, to see if it was suitable for selection.

Pattern **Instantiating a framework**

You have been given, or previously selected, a framework to use as a solution for a specific problem. You are now about to instantiate the framework in order to implement the intended functionalities and build your application.

Problem Framework instantiation usually consists on deducing, designing and implementing application-specific extensions to the framework. Despite knowing which extensions the framework requires, it is hard to understand where to "plug" those extensions in the framework.

What do you need to learn about a framework in order to instantiate it quickly?

Forces **Documentation.** Tutorial documentation can help you to walkthrough the initial contact with the framework and to acquire knowledge about the framework's entry points.

Effort. You don't want to spend too much time learning what you need to know to instantiate the framework.

Learner's experience. If you are already familiarized with the framework, you try to find similar areas of flexibility. A novice learner will look for demonstrating examples that might give her a hint of where to start poking the code for *hot-spots*.

Complexity. Complexity may not mean "difficult to use", but surely means "difficult to learn". Issues like indirection, abstraction and obscurity give the framework its power but also hinder its ability to be learnt and understood.

Solution Find the areas of customization of the framework by looking at the existing documentation and resort to instantiation examples to clarify on how to use those areas.

Look at the documentation to find the *CUSTOMIZATION POINTS* [4] of predefined refinement where framework instantiation is supported. In addition, look also into some *GRADED EXAMPLES* [3] that explain how to use the framework to implement more common functionalities. The customization of a framework is usually possible through sub-classing of framework abstract classes and/or composition of concrete classes. Understanding how these classes relate and interoperate is crucial to be able to use them properly.

If you're dealing with a white-box framework, it is important to further UNDERSTAND THE ARCHITECTURE and to UNDERSTAND THE DESIGN INTERNALS. Only then you can start to UNDERSTAND THE SOURCE CODE and effectively start reusing the framework.

To be more effective, you may consider to DRIVE YOUR LEARNING according to your experience and specific requirements.

Consequences **Framework know-how.** You gain knowledge on how to instantiate the framework, progressively increasing your expertise and being able to incrementally build your application.

Blind trust. Using a framework means trusting in code you have never seen. So if the framework is poorly built or has features that it publicizes but are not implemented or don't work well, your solution may suffer with it. It's not rare to see frameworks whose internal code is not available for debugging or modification.

Rationale Framework instantiation into domain-specific application takes place at points of predefined refinement called *hot-spots* [20]. Thus knowing where and how to use these points leads to an effective framework instantiation. Moreover, [17]the best way to start learning a framework is by example [21]. Most frameworks come with a set of examples that you can study, and those that don't are nearly impossible to learn. Examples are concrete, thus easier to understand than the framework as a whole. Frameworks are easier to learn if they have good documentation.

See also In [10], Froelich *et al.* resort to a Hooks-model to describe the framework customization points and use it to instantiate the SEAF (Size Engineering Application Framework). Their approach is similar to this as is relies on documentation describing the customization points (*hooks*) and uses it to know where to instantiate the framework.

Pattern **Evolving a framework**

You are a software engineer who is responsible for the maintenance and evolution of a framework. Your task may be to evolve the framework to support new requirements, refactoring its design, or the correction of errors, while preserving its backward compatibility.

Problem To evolve a framework means understanding where the evolution will take place within the framework and to which extent do you need to go. You need to know what elements to evolve and its impact on the framework as a whole.

What do you need to learn to evolve a framework?

Forces **Documentation.** The documentation is almost always descriptive, which is not good for framework evolvers, because original framework designers can't predict how the framework might be extended in the future through additional flexibility on existing hot spots, or in additional hot spots. A more prescriptive documentation would be better.

Maintenance expertise. It is expected that the framework maintainers are both domain experts and software experts.

Evolution task. Your task may be adding new functionalities or improving existing ones, correcting errors or refactoring the design. Different information needs arise according to the task at hand.

Tools. There might be the need to recover lost design information that is important to the evolution task. Existing reverse engineering tools may prove useful.

Solution Look at the architecture of the framework and understand how it is built and how it meets its purpose. Gain further insight of its components by looking at the design internals and areas of flexibility and treat each variability issue separately.

Have a good UNDERSTANDing of THE ARCHITECTURE and its rationale, in order to avoid the architectural drift problem [22], commonly consequential of poor framework evolution. UNDERSTANDing THE DESIGN INTERNALS and UNDERSTANDing THE APPLICATION DOMAIN helps at keeping the evolution process in perspective. Look at the *CUSTOMIZATION POINTS* [4] that support the flexibility offered by the framework and plan you evolution tasks.

To be more effective, you may consider to DRIVE YOUR LEARNING according to your experience and specific requirements.

Consequences **Evolution expertise.** You gain enough insight to adequately address your evolution tasks. Be alert to issues regarding delta analysis, architectural drifts, version proliferation and over-featuring [22].

Ignorant surgery. Evolving parts of the framework means understanding its interaction with its other parts. Sometimes, focusing too much on the problem at hand may cause what is called "ignorant surgery" [12]. Inadequate investigation prior to performing a change task limits the understanding of the existing design of a system. The evolver performs a change in a single location in the code that is better understood, but which may lead to unforeseen effects throughout the framework as its dependencies aren't properly identified and taken into account.

Rationale The need to evolve a framework usually arises during any of the following situations: (1) new domain concepts need to be incorporated into the framework, (2) reducing the complexity of the framework through re-design and (3) initial design issues that were neglected need to be addressed [22]. The evolution process usually involves the execution of two tasks: restructure (*refactoring*) and extension. In order to restructure it properly, the developer must be aware of all the repercussions and dependencies of the components or customization areas she intends to extend or alter. Another concern is application compatibility. The framework must remain compatible with earlier developed applications, whereas a faulty evolution process may change the way the framework is supposed to be used, closing otherwise opened customization points. By understanding how the framework is supposed to be used will enable the developer to maintain its interface coherent, without too much effort.

See also In [1], Roberts and Johnson present a pattern language for evolving frameworks where they show that there is need for the understanding of different levels of detail concerning the framework components.

In [39], Cortés *et al.* present a tool to support framework evolution tasks, namely refactoring and extension. They propose to automate certain kinds of refactoring tasks and applying extension rules based on Pree's meta-patterns, which implement variation points as a combination of template and hook methods.

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?

Top-down vs. bottom-up. A top-down approach will start at a higher-level progressing downwards, giving a good overview with little effort but poor details. A bottom-up approach starts at a low-level progressing upwards, giving good detail with little effort, but hindering awareness of the global impact of changes.

Learner's experience. You experience with the framework can affect you learning strategy, when choosing where to start and how to proceed.

Learning style. You may be a more "global", "reflective" learner or you may possess a more "sequential", "active" learning behaviour [13].

Documentation. Depending on the existing documentation artefacts, the learner will have to adapt his learning strategy to quicker and better fill in her knowledge gaps.

Solution Select an entry point to start your understanding. Progress to the understanding level you feel more comfortable with, changing directions whenever needed.

A more experienced learner tends to adopt a more top-down approach, whereas a novice learner will go for a more bottom-up approach [25][26][27]. Remember you can start at any abstraction level. A "global", "reflective" [24] learner will start at a higher level of abstraction and will "top-down" gradually into the framework, because she needs the big picture first. A "sequential", more "active" learner will start at a lower-level, try things out and "bottom-up" into the framework, gathering bits and pieces to form her mental model. Then, change directions, that is, swap strategies, as needed. This is beneficial to reduce cognitive overload and focus on the goal.

Look at the *DOCUMENTATION ROADMAP* [2] and choose the documentation artefacts that may better assist you on your understanding tasks, namely *FRAMEWORK OVERVIEW, GRADED EXAMPLES, CUSTOMIZATION POINTS, DESIGN INTERNALS* and *COOKBOOK & RECIPES* [2][3][4].

Consequences **Methodical approach.** A methodical investigation proves more effective than a chaotic one. By defining a course of action the chances of reaching an answer faster, increase.

Personalized cognitive process. Navigate freely along the abstraction levels until you feel satisfied with the things you've learned. Your mental model will progressively increase throughout task execution.

Rationale On the field of Program Comprehension, many researchers have studied how programmers understand programs through observation and experimentation [28][29]. This research has resulted in the development of several cognitive theories to describe the comprehension process. These range from bottom-up [30][31], top-down [32][33], knowledge-based [34] and systematic [35] converging into an integrated model that frequently switches between all of these [36]. This integrated model would serve a wider range of learners as it would give the learner the option of choosing the most effective learning strategy. All of these cognitive models use existing knowledge together with the code and documentation to create a mental representation of the program.

See also In [40], Schull *et al.* performs a study of about reading techniques while learning about a framework and divides them into two categories: hierarchy-based and example-based. While the former is mostly used by experienced learners, the latter gains the preference of the most novice learners. Nevertheless, one important conclusion of the study is that the learning process should not be strict and allow the learner to freely choose the way she feels more comfortable with, thus potentially achieving the better results faster.

In [37], an exploratory study was performed on how developers investigate source-code in order to perform a change task. One of the major results of that study was that a methodical investigation of the code of a system was more effective than an opportunistic approach. Nevertheless, this theory does not imply that a purely systematic approach to program investigation is the most effective. Successful subjects also exhibited some opportunistic behaviour.

Pattern **Knowledge-keeping**

You want to keep what you have learned while understanding the framework. You want to be able to use that knowledge in the future so that you don't have to do it all again. Also you want it to be fit for other framework users.

Problem Learning how to use a framework means finding, browsing, using and generating understanding knowledge. Reusing the knowledge in future learning tasks is as useful as reusing design and code. Developers go to great lengths to create and maintain rich mental models of code that are rarely permanently recorded. Preserving this knowledge for later use is, therefore, of utter importance.

How to adequately preserve the acquired learning knowledge?

Forces **Existing Documentation.** Adopting existing documentation artefacts as templates to harbour new knowledge depends on its availability, easiness of use and quality of its contents.

Intrinsic knowledge. Much relevant information is kept in the minds of experts that have used the framework. This knowledge decays with time and never becomes useful to others but the expert himself. Sharing this knowledge is important, but might be expensive to experts as it causes interruption and can be time-consuming.

Tools. Documentation generation tools, using recovery and extraction techniques, might be used to generate several specific kinds of views and formats over the information about the framework.

Motivation. Producing documentation can be tiresome and boring. The long-term cost-benefit is often overlooked, thus affecting the motivation to spent time and resources producing documentation.

Solution Use documentation methodologies and tools to produce documentation artefacts and store them in an open, shared, collaborative environment where the information can be accessed and evolved through time.

Choose the documentation artefacts that most adequately can register the knowledge you've acquired, namely *FRAMEWORK OVERVIEW, GRADED EXAMPLES, CUSTOMIZATION POINTS, DESIGN INTERNALS* and *COOKBOOK & RECIPES* [2][3][4].

Consequences **Shared knowledge base.** The learning knowledge is shared through the community of learners, from experts to novices, being all able to use and improve it according to their needs.

Collaborative effort. By opening the knowledge to the community, its quality improves from the constant revising by a heterogeneous group of learners, grasping all the benefits this can bring.

Rationale Good quality documentation is crucial for the effective reuse of object-oriented frameworks. Without a clear, complete and precise documentation describing how to use the framework, how it is designed, and how it works, the framework will be particularly hard to understand and nearly impossible to use by software engineers not initially involved in its design.

Documenting a framework is not trivial. Producing framework documentation needs to address several issues ranging from contents consistency to contents organization. Using framework documentation also poses a problem where issues like understandability, searchability, and effectiveness need to be adequately addressed [5].

Adopting known documentation artefacts [2][3][4] specific to our learning task to store our understanding knowledge helps to lessen the burden of recording our findings. If that knowledge is then shared with a community of other fellow users, that burden can be even less because the responsibility of keeping the information up-to-date is also shared by the other contributors.

The "community" factor also contributes to the refining and quality increase of the documentation as factors like diversity, independence, decentralization and aggregation [46] will mitigate quality issues like accommodating different audiences, having different views over the information or even the lack of standards.

See also In [5], a minimalist approach to framework documentation is proposed. It presents an extensible documentation infrastructure based on the WikiWikiWeb concept and XML technology. It provides several document templates and a simple cooperative web-based environment to produce and use minimalist framework documentation. The proposed approach covers the overall documentation process, from the creation and integration of contents till the publishing and presentation. It encompasses a documentation model, a process and a set of supporting tools.

Pattern **Understand the application domain**

You have a framework you want to use, but you don't know its general purpose or if it covers your application domain.

Problem You need to be sure that the framework answers your functional and domain requirements. Not only the general purpose of the framework must be clear but also its reach and the assurance that it covers, if not all, the required problem domain areas and constraints of the application to develop.

How do you learn what is the purpose of the framework and the domain scope it covers?

Forces **Learner's domain knowledge.** The easiness of finding where the domain concepts are present, and which areas relate to those domains, strongly depend on the learner's knowledge about the application domain. Metaphor and technical jargon may be useful to track down and identify hints on component names that might relate to domain concepts.

Expert domain knowledge availability. If such an expert is available for consult, it should nurture and speed up domain knowledge acquisition and promote a domain-driven analysis of the framework.

Documentation. The documentation should give ideas on how the domain is mapped onto the framework. It could contain a brief description of the framework and its main purpose and concepts.

Solution Identify the general purpose of the framework and its application domain by browsing the existing documentation and capture the main domain concepts, how they relate and how the framework addresses them.

A FRAMEWORK OVERVIEW [2] is a good way to do so and *GRADED EXAMPLES* [3] provide detail on how the main features can be implemented.

Find the framework top components and their metaphor (names and designations) and UNDERSTAND THE ARCHITECTURE of how they are related to cover the domain concepts.

Preserve all the information gathered, adopting a KNOWLEDGE-KEEPING strategy.

Consequences **Broadness.** Viewing the framework at this level enables the learner to know the general purpose of the framework and its overall domain applicability.

Shallowness. Without going into more detail it is sometimes difficult, if not impossible, to ascertain if a certain functionality or technology is covered by the framework. As such, one needs to dig deeper and try to UNDERSTAND THE DESIGN INTERNALS in order to understand how some pieces fit in together, because the system requirements need detailed specifications of certain functionalities.

Rationale When you know nothing about a framework, usually you try to see what the framework is for. You look for the title, a paragraph, maybe the name of the components. These elements are usually on the documentation that accompanies the framework, whether is a specific document, website or other kind. When trying to find out its purpose, you look for keywords or something that will shed some light about the domain concepts of the framework. It is about graphics? It is about networks? It is general-purpose? What are the concepts it encompasses and how? Only after you've acquired this information you start looking for other details.

See also In [19], the process of determining a framework's suitability to a problem domain starts with the domain analysis activity. This activity has several non-contiguous steps to reach a domain model, where existing documentation (when this documentation is not available for the framework itself, they resort to documentation belonging to exiting application developed using that framework) is reviewed and domain experts are consulted. Also existing standards for the domain are studied. The result of the activity is a domain analysis model containing the requirements of the domain, the domain concepts and the relationships between concepts.

Pattern Understand the architecture

You are using a framework and you want to know if its architecture is compatible with your application needs. You want to understand how the framework elements are structured and how they relate.

Problem Using or evolving a framework impacts the framework as a whole. The awareness of the full implications of any change to the framework requires a sound notion of the framework's architecture and how its elements, which map the domain concepts, relate with one another. You need to understand its architecture.

How do you learn about the framework's architecture, its components and internal relationships?

Forces **Framework maturity.** A matured framework is likely to be better structured, being easier to identify its main architectural elements.

Documentation. If there is existing documentation that explains the overall architecture, it can be a great understanding aid.

Tools. These can complement the lack of overview documentation, by reverse-engineering the architectural information.

Solution Look into the documentation or any existing reverse-engineered design information and search for instances of architectural patterns [11]. Usually present in a more mature framework, these can indicate its main architectural style.

Browse through the *DESIGN INTERNALS* [4] to identify the main architectural concepts and its relationships. If you need more detail, look for architectural primitives [8]. These can give an incremental view of the overall architecture by identifying interfacing ports between framework components and later, by aggregation, lead to defining a known architectural pattern or structure.

Preserve all the information gathered, adopting a KNOWLEDGE-KEEPING strategy.

Consequences **High-level awareness.** There is an awareness of all the framework internal components and how they relate. You can piece together all the framework's parts to see if it fits your application needs.

Shallowness. Despite being comprehensive, there is no grasp of how the components that relate to each other, interoperate, or how they function internally. You need to further UNDERSTAND THE DESIGN INTERNALS, to be able to know more about their behaviour.

Rationale A framework is an architectural abstraction. An architectural abstraction identifies and names composition of elements with a certain structure and functionality. This facilitates communication about designs. A framework provides a set of abstractions that are useful when discussing and describing a domain [41]. When a white-box framework is used, it is necessary to understand the concepts and architectural style of the framework in order to develop applications that conform to the framework. Many errors can be avoided and the application can be constructed more efficiently if the framework user understands its strategies and styles. In a matured framework, during design stage, a suitable architectural style was adopted and usually these are known domain-specific architectural patterns.

See also In [42], Shaw and Garlan, first introduce the notion of software architectural styles as a family of systems in terms of a pattern or structural organization. More specifically, it determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined.

In [11], Buschmann *et al.* presents a pattern catalogue of architectural styles based on the work of Shaw and Garlan and introduces a software design classification system consisting of architectural patterns, design patterns and idioms, covering different perspectives and different abstraction levels.

In [8], Zdun and Avgeriou propose to remedy the problem of modelling architectural patterns through identifying and representing a number of "architectural primitives" that can act as the participants in the solution that patterns convey. According to the authors, these "primitives" are the fundamental modelling elements in representing a pattern and also they are the smallest units that make sense at the architectural level of abstraction (e.g., specialized components, connectors, ports, interfaces). Their approach relies on the assumption that architectural patterns contain a number of architectural primitives that are recurring participants in several other patterns.

Pattern Understand the design internals

You want to use the framework to solve a specific problem. You need to know how the framework can be used to implement a specific solution to that problem.

Problem To effectively use a framework, its flexibility and reuse points reside mostly at an intermediate design level. Due to its complexity, it is not clear where those points are and how they are used to implement the solution. Therefore, understanding the design internals is essential to find those flexibility points.

How do you understand the design internals of a framework?

Forces **Design complexity.** A framework design is, by nature, complex. Most of its complexity can be found at this level of design.

Inheritance vs. composition. Design variations for the same problem may prove to be a hinder because they obfuscate the identification of existing solutions as they seem dissimilar.

Documentation. Depending on the existing documentation, one may find bits and pieces of information about how design solutions were implemented to solve specific domain problems.

Tools. Reverse engineering and software visualization tools that aid in identifying known design structures and patterns may save time and give a different view over the whole or part of the framework's design.

Solution Go through the *DESIGN INTERNALS* [4] documentation, or browse the existing classes, and identify the concept classes and their interactions.

Look for instances of known design patterns. Design patterns [9] are often used as building blocks for frameworks, because they introduce the flexibility it needs. The more mature a framework is, the more design patterns will it encompass. Design patterns aggregate these "hot spots" or *CUSTOMIZATION POINTS* [4]: areas of flexibility we can "hook" [10] into and take full advantage of the framework's reusability.

Preserve all the information gathered, adopting a KNOWLEDGE-KEEPING strategy.

Consequences **Framework internal mechanisms.** You gain knowledge about how the framework provides the flexibility for adapting its semi-implementation to develop an application. You acquired most of the information to adapt the framework to your needs.

Still at a Design level. Understanding is kept at a design level. Adapting means implementing, and implementing means coding. You need therefore to UNDERSTAND THE SOURCE CODE.

Rationale Frameworks are designed and implemented to fully exploit the use of dynamically bound methods. Template and hook methods [43][9] are two kinds of methods extensively used in the implementation of frameworks, to provide it with its flexibility and adaptability.

Generally, template methods are used to implement the frozen spots of a framework, and hook methods are used to implement the hot spots. The frozen spots are aspects that are invariant along several applications in a domain, possibly representing abstract behavior, generic flow of control, or common object relationships. The hot spots of a framework are aspects of a domain that vary among applications and thus must be kept flexible and customizable.

The difficulty of good framework design resides exactly on the identification of the appropriate hot spots that provide the best level of flexibility required by framework users. More hot spots offers more flexibility, but results in a framework more difficult to design and use, so somewhere in between resides a balanced design.

In [43] are identified several ways of composing template and hook classes, and presented under the form of a set of patterns, globally called *meta-patterns*. Meta-patterns categorize and describe the essential constructs of a framework, on a meta-level. Design patterns provide proven solutions to recurrent design problems and are extremely useful to design object-oriented frameworks. The motivation for using meta-patterns is to provide a means to categorize and describe design patterns on a meta-level, and to support framework construction. Therefore, design patterns become the building blocks of frameworks.

Design patterns can be used as inspiration when looking for flexible hot-spots within a framework. A framework that contains design patterns can be understood in term of these; therefore when adapting a framework, users can perceive the specific adaptation steps (sub-classing or configuring framework classes) as adaptations of small wholes – the involved design patterns – instead of making new atoms (classes). Users see their adaptations in a perspective larger than that of a single class [41].

See also In [44], Bruch et al. propose the use of data mining techniques to extract reuse patterns from existing framework instantiations. Based on these patterns, suggestions about other relevant parts of the framework are presented to novice users in a context-dependent manner.

In [45], Fairbanks et al. present a pattern language based on the notion of design fragment. A design fragment is a pattern that encodes a conventional solution to how a programmer interacts with a framework to accomplish a certain goal. It provides the programmer with a "smart flashlight" to help him/her understand the framework, illuminating only those parts of the framework he/she needs to understand for the task at hand. Design fragments give programmers immediate benefit through tool-based conformance and long-term benefit through expression of design intent.

Pattern **Understand the source code**

You want to code your solution using the framework.

Problem To actually use a framework you have to code. Therefore, understanding the source code is mandatory. But a framework is not a common piece of code: it has no clear entry point and there isn't a "main" method from where to start understanding the flow of control. Its "hot-spots" are scattered across the code and the way to use them may not be straightforward.

What to look for to understand the source-code and plug-in your solution?

Forces **Hollywood principle.** Your code will have to be inserted at a specific location that the framework will eventually call and execute. It might not be straightforward how and where that calling will take place.

Language familiarity. If you are not familiarized with the programming language in which the framework is built, you're going to take more time to understand the code.

Task-orientation. To be cost-effective, learners tend to focus on the task at hand, and to find the quickest way to solve their immediate problem.

Code Annotations. Code annotations and inline documentation can give helpful insight on a certain code fragment was implemented or served a purpose.

Documentation. Usually, the framework comes with examples on how to quick start or how to quickly address initial problems. These can be extremely helpful as they show how to begin and force you to try to understand how the system works.

Solution Browse the documentation for examples on how to address the task at hand. Identify the code main entry points for your specific task.

Usually, the framework comes with a *COOKBOOK & RECIPES* [3] on how to solve common problems the framework addresses. These show you how to begin coding and enable you to understand how the overall system works.

If no documentation is present, try to look for beacons and idioms that might hint to the entry point(s) of the framework (Control classes and "main" methods) and track down the flow of control. Idioms are coding patterns that are used to solve recurrent problems (You use a loop to iterate over an array, etc.). Beacons are fragments of code that may resemble algorithm techniques or coding strategies to known problems. Classifying and chunking code into these concepts might prove useful to increase code granularity search.

Identify your insertion points as you go along and preserve all the information gathered, by adopting a KNOWLEDGE-KEEPING strategy.

Consequences **Missing the whole picture.** At such a low-level, the learner has local expertise but might overlook more global side-effects of code insertion or modification. A broader notion of what is happening might be necessary, so you might need to UNDERSTAND THE DESIGN INTERNALS to gain further awareness.

Rationale Prior to performing a software modification task, developers must inevitably investigate the code of the target system in order to find and understand the code related to the change. With frameworks, the theory is the same. If we assume that the way a developer investigates a program influences the success of the modification task, then ensuring that developers in charge of modifying software systems investigate code of the system effectively can yield important benefits such as decreasing the cost of performing software changes and increasing the quality of the change.

Developers should follow a general plan when investigating a program, should perform focused searches in the context of this plan, and should keep some form of record of their findings.

Documentation here is crucial. Not only should there be some sort of guide for browsing the code, but also examples on how to address the most common problems. Going through these examples would be a valuable assisted first "dive" into the framework code and would help emerge the control-flow mechanism of the framework and the way it is supposed to be used.

See also In [25], Sillito *et al.* performed a study where they observe developers trying to understand a system in order to perform a change task. They harvested and identified 44 different kinds of questions developer ask during that process and divided them into four categories based on the characteristics of the source code graph capturing the information needed for answering a given question: those aimed at finding the initial focus points, those aimed at building on those points, those aimed at understanding the sub-graph, and those over such sub-graphs.

In [37], Robillard *et al.* conducted another study where observed successful and unsuccessful developers while performing a software evolution task and came up with a theory of program investigation effectiveness in the form of a series of observations and associated hypotheses. Overall, they found that successful developers exhibited a highly methodical approach to program investigation, where they identified the high-level structures and planned the changes to be made, without forcefully spending more time than a more opportunistic approach.

References

[1] Roberts, D. and Johnson, R. E. (1997). Evolving frameworks: A pattern language for developing object-oriented frameworks. In Pattern Languages of Program Design 3. Addison Wesley.

[2] Aguiar, A., and David, G. (2005). Patterns for Documenting Frameworks – Part I. In Proceedings of VikingPLoP'2005, Helsinki, Finland.

[3] Aguiar, A., and David, G. (2006). Patterns for Documenting Frameworks – Part II. In Proceedings of EuroPLoP'2006, Irsee, Germany.

[4] Aguiar, A., and David, G. (2006). Patterns for Documenting Frameworks – Part III. In Proceedings of PLoP'2007, Portland, Oregon, USA.

[5] Aguiar, A. (2003). A minimalist approach to framework documentation. PhD thesis, Faculdade de Engenharia da Universidade do Porto.

[6] Flores, N. (2006). From Program Comprehension to Framework Understanding: a roadmap. Available at http://www.fe.up.pt/~nflores

[7] Alexander, C., Ishikawa, S., and Silverstein, M. (1977). A Pattern Language. Oxford University Press.

[8] Zdun, U. and Avgeriou, P., (2005) "Modeling Architectural Patterns Using Architectural Primitives", OOPSLA

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

[10] Froehlich, G., Hoover, H., Lui, L. and Sorenson, P. (1997) "Hooking into Object-Oriented Application Frameworks", Proceedings of the 19th International Conference on Software Engineering, pp.491- 501.

[11] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M..(1996) "Pattern oriented software architecture - a system of patterns". John Wiley and Sons.

[12] Robillard, M., Coelho, W., and Murphy, G., (2004) "How Effective Developers Investigate Source Code: An Exploratory Study", IEEE Transactions on Software Engineering, vol..30, no. 12, December 2004

[13] Felder, R.., and Spurlin, J. (2005) "Applications, Reliability, and Validity of the Index of Learning Styles". International Journal of Engineering Education, v. 21, n. 1, pp. 103-112.

[14] Butler, G. (1997). A reuse case perspective on documenting frameworks. http://www.cs.concordia.ca/faculty/gregb.

[15] Johnson, R. E. and Foote, B. (1988). "Designing reusable classes". Journal of Object-Oriented Programming, 1(2):22–35.

[16] Deutsch, L. P. (1989). "Design reuse and frameworks in the smalltalk-80 system". In Software reusability: vol. 2, applications and experience, pages 57–71. ACM Press.

[17] Fayad, M. E. and Schmidt, D. C. (1997b). "Object-oriented application frameworks." Communications of the ACM, 40(10):32–38.

[18] Turner, A. and Wang, C. (2007). "AJAX: Selecting the Framework that Fits". Dr. Dobb's Journal, May 01, 2007. http://www.ddj.com/web-development/199203087

[19] Ahamed, S. I., Pezewski, A., and Pezewski, A. (2004). "Towards Framework Selection Criteria and Suitability for an Application Framework." In Proceedings of the international Conference on information Technology: Coding and Computing (Itcc'04).

[20] Pree, W". "Design Patterns for Object-Oriented Software Development" (1995). Addison-Wesley / ACN Press 1995

[21] Fayad, M. E., Schmidt D. C., Johnson, R. E. "Building Application Frameworks" (1999). John Wiley & Sons, Inc. 1999

[22] Codenie W., Hondt K., Steyaert P., Vercammen A. "From Custom Applications to Domain-Specific Frameworks." (1997) Communications of the ACM, 40(10): 71-77. 1997

[23] Butler G., Xu L. "Cascaded Refactoring for Framework Evolution" (2001). Proceedings of 2001 Symposium on Software Reusability. ACM Press. P. 51-57. 2001

[24] Felder, R.M. and Silverman, L.K. "Learning and Teaching Styles in Engineering Education" (1988), *Engr. Education, 78*(7), 674-681, 1988

[25] Sillito, J., Murphy, G. C., and De Volder, K. (2006). "Questions programmers ask during software evolution tasks". In *Proceedings of the 14th ACM SIGSOFT international Symposium on Foundations of Software Engineering* (Portland, Oregon, USA, November 05 - 11, 2006)

[26] Kirk, D., Roper, M., and Wood, M. (2005). "Identifying and Addressing Problems in Framework Reuse". In *Proceedings of the 13th international Workshop on Program Comprehension* (May 15 - 16, 2005).

[27] Shull, F., Lanubile, F., and Basili, V. R., "Investigating Reading Techniques for Framework Learning" (1998), Technical Report CS-TR-3896, UMCP Dept. of Computer Science, 1998

[28] Storey, M-A, "Theories, Methods and Tools in Program Comprehension: Past, Present and Future." (2005) Proceedings of the 13th IEEE International Workshop on Program Comprehension (IWPC). St. Louis, MO, pp. 181-191, IEEE Computer Society Press 2005.

[29] Storey, M-A, Fracchia, F. and Muller, H.. "Cognitive design elements to support the construction of a mental model during software visualization". (1997) Proceedings of the 5th International Workshop on Program Comprehension (IWPC'97), Dearborn, Michigan, pp. 17-28, May, 1997

[30] Pennington, N., "Stimulus structures and mental representations in expert comprehension of computer programs" (1987), Cognitive Psychology, pp. 295-341, vol 19, 1987.

[31] Shneiderman, B. and Mayer, R., "Syntactic/semantic interactions in programmer behavior: A model and experimental results" (1979). International Journal of Computer and Information Science, pp. 219-238, 8(3), 1979

[32] Brooks, R., "Towards a theory of the comprehension of computer programs" (1983), International Journal of Man-Machine Studies, pp. 543-554, vol.18, 1983.

[33] Soloway, E. and Erlich, K. "Empirical studies of programming knowledge" (1984), IEEE Transactions on Software Engineering, pp. 595-609, SE-10(5), September 1984.

[34] Letovsky, s. "Cognitive processes in program comprehension" (1986), Empirical Studies of Programmers, pp.58-79, 1986

[35] Littman, D.C., Pinto, J., Letovsky, S. and Soloway, E., "Mental models and software maintenance" (1986), Empirical Studies of Programmers, pp. 80-98, 1986

[36] Maryhauser and Vans. "Program comprehension during software maintenance" (1995), IEEE Computer pp. 44-55, August 1995.

[37] Robillard *et al.* "How Effective Developers Investigate Source Code: An Exploratory Study" (2004). IEEE Transactions on Software Engineering, Vol. 30, Nº12, December 2004.

[38] Froehlich, G., Hoover, H.J., Liu, L., Sorenson, P.G. "Choosing an Object-Oriented Domain Framework" (2000), ACM Computing Surveys, vol.32(1), 2000.

[39] Cortés, M., Fontoura, M., Lucena, C. "Framework Evolution Tool" (2006). Journal of Object Technology, vol.5, no.8, November-December 2006, pp.101-124.

[40] Schull, F., Lanubile, F., Basil, V. "Investigating Reading Techniques for Object-Oriented Framework Learning", IEEE TSE, vol.26, nº.11, 2000

[41] Jacobson, E.E., Nowack, P. "Frameworks and Patterns: Architectural Abstractions" (1999). In *Building Application Frameworks*, Chapter 2, pp.29-54, John Wiley & Sons, 1999

[42] Shaw, M., Garlan, D., "Software Architecture – Perspectives on an Emerging Discipline", Prentice Hall, 1996

[43] Pree, W. (1995). Design Patterns for Object-Oriented Software Development. Addison-Wesley / ACM Press.

[44] Bruch, M., Schäfer, T. and Mezini, M., "FrUiT: IDE Support for Framework Understanding" (2006) OOPSLA Eclipse Techonology Exchange, 2006

[45] Fairbanks, G., Garlan D. and Scherlis, W. "Design Fragments Make Using Frameworks Easier" (2006), OOPSLA 2006.

[46] Surowiecki J. "The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations" (2004), Little Brown 2004