An Approach to Evolutionary Design Pattern Engineering

René Reiners, Fraunhofer FIT
Ragnhild Halvorsrud, SINTEF ICT
Aslak Wegner Eide, SINTEF ICT
Daniela Pohl, University of Klagenfurt

The design of interactive systems, especially in distributed research projects, is a challenging process in which many concepts are developed with successful outcomes but also with dissatisfying results. In order to structure and relay knowledge about good or bad approaches, design patterns are a well-known instrument in research and development. Due to the condition that a design pattern must be easy to read, different stakeholders in the system engineering and design process are able to understand the described concepts without the need of specific expert knowledge . In distributed projects, application design knowledge may be scattered and documented in different manners. This means, before we can start formulating patterns, we need to discover and gather the available and partially concealed design knowledge. Since these fragments of knowledge may not always be accurately formulated for being used as design patterns, we seek for a collaborative method for collecting and formulating early findings together with established design knowledge. In this paper we present a concept of an evolutionary process for capturing, formulating, refining and validating design patterns. Our approach aims at involving as many stakeholders as possible in order to shape a pattern language over a project's lifetime in a collaborative process allowing facile participation. We implement our approach in the scope of the EU research project BRIDGE that aims at supporting inter-agency collaboration during emergency response. We close with a discussion of the current state and envisioned next steps in order to foster our considerations.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/ methodology*; K.4.3. [**Organizational Impacts**] Computer-supported collaborative work

General Terms: Human Factors

Additional Key Words and Phrases: Design Patterns, Pattern Languages, Pattern Evolution Process, Collaboration

1. INTRODUCTION

The design of interactive systems, especially in distributed research projects, is an interdisciplinary challenge involving many stakeholders from different disciplines like psychology, design, economy in addition to computer science. Not to forget the most important stakeholder who benefits or suffers from the outcome: *the user* who actually places the demand for certain features of devices or systems. Well-designed applications increase the chance of up-take by users, and can easier lead to economical success [Fornell et al. 2006]. Proven designs are extracted and reincorporated into products or future research and this way provide new ground for ongoing research and improvement. Thus, the important question to ask when designing an interactive system is:

"How do we develop successful applications that are **accepted by users?"**

One approach to answer the above question is to directly involve the users in the design process, as emphasized by Nielsen in his user-centered design philosophy. He describes iterative development cycles consisting of the

The research leading to these results has received funding from the EU 7th Framework Programme (FP7/2007-2013) under GA no.261817, the BRIDGE project (www.bridgeproject.eu) within the Security Programme SEC-2010.4.2-1: Interoperability of data, systems, tools and equipment.

phases "design" "implement" and "analyze" [Nielsen 1993]. Iterative design is a well-suited instrument for system development that focuses on early evaluations and rapidly prototyping design ideas together with users. This approach can be combined with participatory design where designers and engineers of the development process gather experience directly in the domain's daily environment (cf. [Kensing and Blomberg 1998] about issues and concerns). [Dow et al. 2010] foster the approach of parallel prototyping which results in better designs based on more diversity and more task-specific confidence. The refinement of only one prototype in a serial process, however, may hinder the assessment of other alternatives and even narrow the possible design space [Buxton 2007]. This method also supports novices in the field since they can try out several alternatives in parallel and see where the best acceptance lies and follow this direction [Dow et al. 2010].

In parallel to a user-centered and iterative design process, design patterns can be taken into account. They communicate insights into common design problems and reflect solutions that a community of experts has developed over time. The solution proposed by a design pattern should be generic rather than specific, so that it can be implemented in numerous different ways [Tidwell 2005]. Since design patterns are by now available for a broad variety of application domains, such a "pool of knowledge" should be taken into account (if available) during design phases.

However, the proposed ideas for systems and concept design become more and more difficult to apply in case the project size grows. From our experience we could gather with highly distributed international projects with an average of sixteen project partners, not only the collaboration and coordination becomes difficult to handle, but also the knowledge transfer and integration into the whole project scope. Especially in research projects, knowledge transfer and reintegration are crucial parts for synergies and newly combined findings. Similar problems are also stated by other projects and especially addressed, e.g., in the GLOSE[1] project. Here, efforts are undertaken to extract, structure and reintegrate units of knowledge, called *best practices* for the whole project community [Averbakh et al. 2011; Schneider 2009]. Confidentiality is also addressed by providing access rules for parts of the knowledge that may not be accessible to every partner.

For our work, we leave out that last, even more complicated part of confidentiality, but focus on ways to gather and structure knowledge that is collected during the project's lifetime.

We draw our discussion about the problem of knowledge generation, distribution and management on our experiences made in the last five years, especially on the currently running European project BRIDGE[2]. The BRIDGE project provides the characteristics of an international, highly distributed integrated project with different sources of knowledge generation and specializations in many different technical and non-technical levels.

In the following, we give a generalized overview of the structure and interdependencies within a distributed research project and describe the challenges that occur concerning knowledge generation, exchange and management in Section 2.

In Section 3, we discuss our idea of providing a dynamic collaborative pattern evolution process that conceptually addresses the formulated challenges.

The subsequent section describes the *evolutionary design pattern library process* in detail.

In the summarizing section, we provide a roadmap for our next planned steps and discuss still open issues.

2. PROBLEMS FOR KNOWLEDGE MANAGEMENT IN DISTRIBUTED PROJECTS

When starting a new large scale project the recommended management process according to acknowledged strategies like the "Project Management Body of Knowledge" (cf. [Institute 2008]), is to define the project scope and a management plan by taking into account the project's requirements and possible risks. Then, the implementation and change request process starts which ideally does not follow the traditional waterfall model (cf. [Royce 1970])

[1]http://glose-project
[2]http://bridgeproject.eu

with its fixed and non-reversible phases but a more iteratively-driven process, that should at least be based on the spiral model introduced by [Boehm 1986].

In distributed research projects, especially integrated projects (IP), specifically targeted research projects (STREP) or Networks of Excellence (NoE) in which cooperation, knowledge exchange and exploiting synergy effects from different expert groups within the project is desired[3], the project management must be very flexible but still follow a budget plan and schedule.

The number of participants quickly spans up to twenty partners within the projects each introducing two to four project associates[4]. Thus, the number of people that need to be managed and coordinated quickly grows to amounts between 30 and 100 people. Arising problems concerning collaboration and knowledge management are described by [Averbakh et al. 2011] who also present approaches to address these challenges.

The usual structure of large kinds of distributed research projects is guided by each partner's competences and roles within the project. Usually, different work packages (up to 15) are specified to handle specific project tasks. One specific work package is responsible for managing the entire project and reporting. The other work packages focus, e.g., on domain analysis, eliciting requirements, implementing specific components, designing application concepts and interfaces, specifying software and system architectures, performing technical integration, testing and validating as well as exploiting of the project results. Figure 1 shows an overview of a possible work package distribution. The individual results and therewith the interdependencies between the work packages is shown as directed arrows. Mutual influence as shown for *WP: Req. - Engineering* and *WP: Domain Analysis* is also possible. *WP: Management* is shown as jagged structure as it is interconnected with all available work packages which is not directly shown in the graphics for clarity reasons. The illustration does not claim for completeness and generally applicable correctness but is based on experience gathered from past and present projects. The work packages we focus on in our approach are drawn with a grey background. The interdependencies that we concentrate on are drawn as solid arrows. Following the relevant arrows, a cycle of influence back from *WP: Validation* to *WP: Domain Analysis* and *WP: Application Design* can be seen. This cycle reflects the iterative nature that our current projects follow according to [Nielsen 1993].

It is important to keep in mind that the results of each work package are available in different formats which is not always understandable or directly reusable as input for another work package. Specific transformations might be necessary. This is especially the case for non-formal outputs like domain analysis reports, application design concepts and validation reports that partially fertilize requirements engineering, implementation and integration efforts. These technically-oriented work packages demand ”hard” specifications.

We need to bridge the huge gap between system engineers that approach the domain with a very technical understanding of "what is technically possible" and the user-centered engineering groups that collect information on ”how people and processes work” and ”what they need”.

In the following, the main problems that we encountered in conjunction with information exchange between the work packages domain analysis, application design, implementation and integration as well as validation are discussed. The focus is laid on knowledge *generation*, *exchange* as well as its *application and feedback* between the work packages. Different information *sources* within the project members and involved stakeholders are also discussed. Figure 2 illustrates the problems that are more deeply discussed in the following.

2.1 Knowledge Generation

Right from the start of a project and usually to more than half of it, domain analysis and requirements continuously takes place. This does not only include the generation of new findings but also updating already existing results or collecting experience with assets produced from other work packages in conjunction with the validation processes.

[3]http://ec.europa.eu/research/fp7/index_en.cfm?pg=cooperation

[4]The CORDIS database provides an overview of the currently running research projects in the EU: http://cordis.europa.eu/projects/home_en.html

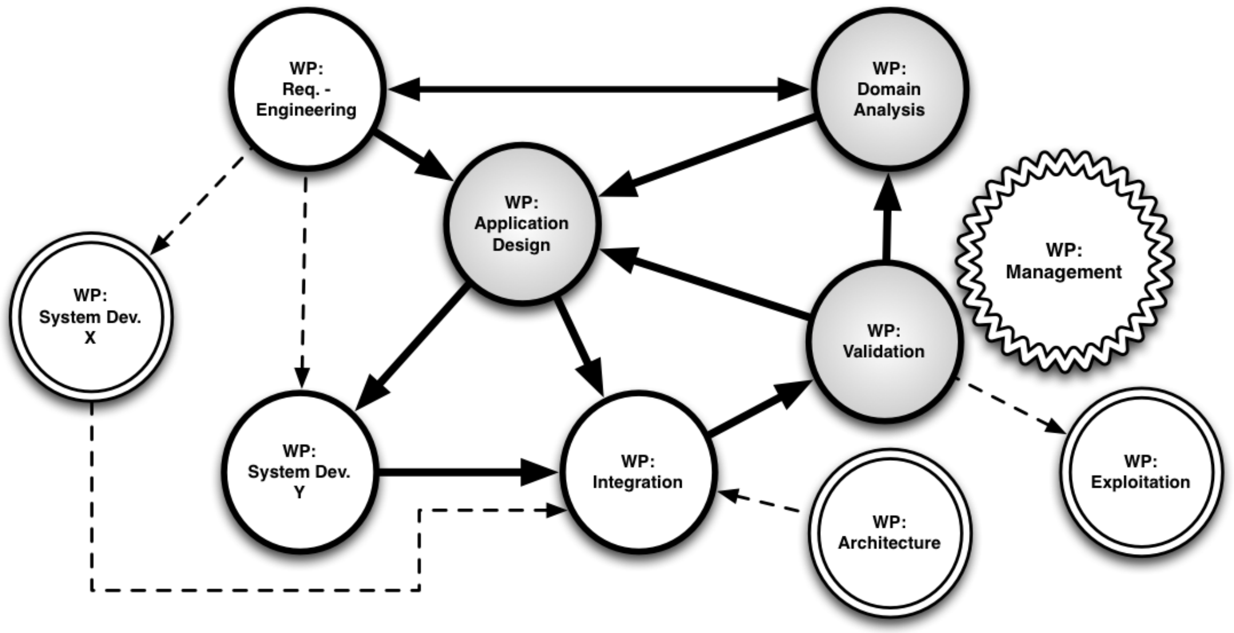

Fig. 1. An example distributed project structure and important influences and interdependencies between the different work packages. The most important ones for our discussion are filled and the connections are drawn in solid strokes. The cyclic dependencies reflect the iterative design, implement and evaluation phases.

The information generated during the domain analysis phase needs to be documented together with the generated requirements. These documents are then fed into the appropriate work packages.

As soon as mandatory rules and guidelines within the project domain have an important impact on the system design, it becomes hard to find documented references regarding successful system design. However, the impacts resulting from the rules and guidelines need to be considered right from the start. Here, domain experts are needed to support the system engineering tasks based on their experience.

However, domain analysis, especially if applying user-centered design methods combined with ethnomethodology, as described by [Suchman 1987] or [Hughes et al. 1992], produce a very large amount of quite lengthy documents [Viller and Sommerville 2000]. The documents are especially considered as lengthy by technical work packages who may even refuse to read the documentation and ask for summarization. However, partners with sociology-related backgrounds tend to present the ”large picture”. For them, also the reasons and originating story behind the findings is important to record for a more complete understanding of the domain [Garfinkel 1967].

It is also important to see that every participant in the project already brings in experience or even assets that will be adapted to the project’s scope. This means that there are partners who just wait for the starting shot to implement their concept and (later) match or adapt it to the elicited requirements and do not really intend to wait for the analysis phases to complete. They expect to be able to quickly browse the results from the analysis that are at the same time easy to understand.

2.2 Knowledge Exchange and Update

The work packages concerned with interaction and technology design introduces their concepts addressing the needs and requirements proposed by the domain analysis and requirements engineering process. Also inspired from the analysis, the work packages concerned with software architecture and integration start addressing the needs that fall into their range.

However, still the problem of coordinating the development and exchanging the project knowledge occurs. For instance, the application design work package picks needs from the analysis. When it then comes to a technical implementation, it may happen that not all information gathered from the workshops was incorporated although large parts were already taken into account. Since analysis and development at some point in time happen in parallel, not every update could be communicated well enough. Another fact is that the documentation generated from the analysis roots from user-centered analysis methods that naturally generated many written documents like reports, videos and transcripts. The more technical work package really have a problem of parsing and digesting all this information. At some point, they even must cut down their efforts on this part and incorporate missing features in the next iteration. The problem of communication between expert groups, especially for ethnomethodology in combination with user-centered design and technicians is discussed in more details by [Hughes et al. 1992] and [Sommerville et al. 1993].

Another example concerns the exploitation work package which waits for findings that can be promoted to the public. The mutual influence of the different work packages is not easily given. Lots of efforts have to be put in refinements, repeated discussions and clarifications.

This may results in unawareness of the individual work package outcomes that are available but not really considered in other specialized work packages, although it would be very important to know about. There is a lot of self-responsibility given to each work package to gather the knowledge generated from others and to update and distribute the own ones.

2.3 Information Contribution From Involved Stakeholders

Later in the project, when parts are being implemented and first prototypes are validated, findings also need to be recorded and documented. Again, there are many different kinds of documentations possible. The problem here is to feed the results back into the project and appropriately inform the other project members. Also for new associates entering the project, the information flood significantly increases the later the entry point is.

Other requirements may first come to mind when the system or prototype is currently used. This knowledge and the one coming from experience must also be documented since it is valuable for future designs. We need to directly involve the stakeholders to document their knowledge since participatory design cannot be applied in all situations

Within distributed projects, we need to engineer for many different stakeholders within the considered domain. Every stakeholder has his own requirements for a system and already experience that must be considered during system design. We also aim at capturing that knowledge from all the stakeholders by giving them the chance to contribute their knowledge in easy way right from the start of the project. Domain and application design knowledge may also be scattered across many sources and distributed in many different ways like handbooks, education but also by everyday-learning and refining daily practice. [Denef 2011], for example, shows in his pattern language how firefighters make adopt procedures and tools to their situations. Here, lots of knowledge is undocumented but still essential for application design.

The mixture of observation and active contribution is important for gathering explicit but also *tacit* knowledge which can not directly be expressed since users are unaware of it [Goguen and Linde 1993]. The same holds for unconscious requirements that are implicitly assumed and users do not talk about them [Kano et al. 1984].

3. APPROACH: DESIGN PATTERNS AS COMMUNICATION MEDIUM

In order to tackle the knowledge management problems that are described in Section 2, we aim at providing a knowledge structure for collecting, managing and developing the results that are gathered during the project lifetime. The documentation format may differ between different work packages as described previously and range from findings, guidelines or rules over processes up to successful as well as failed approaches. Thus, we plan

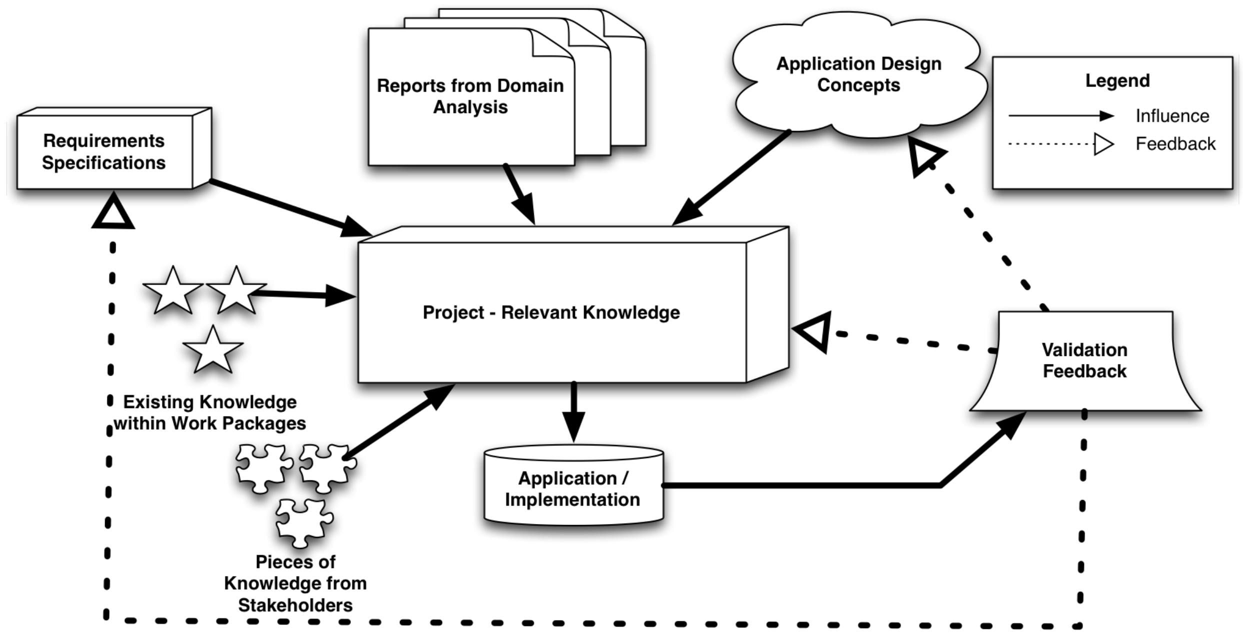
Reports from Domain Analysis
Application Design Concepts
Legend
Influence
Feedback
Requirements Specifications
Project - Relevant Knowledge
Existing Knowledge within Work Packages
Validation Feedback
Application / Implementation
Pieces of Knowledge from Stakeholders

Fig. 2. Information is introduced from different work packages and different partners right from the beginning of the project. Other projects contribute to the global project knowledge over time. Finding from implementations and validations needs to be fed back to the other work packages and to the global repository as well.

to create a common ground structure for formulating the outcomes in a more abstract way. We consider design patterns structured within a pattern language as introduced by [Alexander 1977] and refined by, e.g., [Coplien and Harrison 2004], [Borchers 2001], [Tidwell 2005] or [Schümmer and Lukosch 2007] as a promising approach for an interdisciplinary communication medium.

Our goal is to exactly tailor the pattern language concept to the characteristics of a distributed project. Here, the sharing of results may be seen as an internal process for the partners that should only be made visual in partitions over time. Also, the discussion of the generated knowledge normally stays within the project community until internal and public reports are generated.

Therefore, the generated expert knowledge within a distributed project should be:

—*available* for every project member,

—*understandable* by every project participant and therefore be

—*applicable* to the affected work package tasks

—*extensible* by every stakeholder and

—*dynamic* such that early findings are directly documented and updated in future iterations.

3.1 Pattern Language Generation

Many available pattern languages were formulated over time by smaller sets of expert authors that wanted to share their experience but structure their knowledge. This ”traditional” gathering process spans a longer period of time (that can easily become equal to the project’s lifetime) until others can benefit from the collections. Certain approaches start taking into account more dynamics and publish their patterns for discussion before finalizing them as official assembly, e.g., in printed publications. The discussion and collection usually takes places in dedicated personal *Writers' Workshops* as performed during the PLoP conference programs[5].

As a consequence, knowledge sharing happens quite late in this process and others need to wait to benefit from the accumulated design knowledge. Although the quality and evaluation work spent to compile these collections, readers only have the possibility to consume the knowledge offered. They have to determine for themselves whether the pattern worked out for their design without a direct feedback channel to influence the presented set of patterns.

There is a tendency of discussing pattern collections more collaboratively in a Web 2.0 manner by publishing patterns on portals and providing ways to comment on existing patterns or contribute patterns to the collection. Online collections like the Quince pattern collection [Infragistics 2012], the UI Patterns [Toxboe 2012] or the Amsterdam Pattern Collection [van Welie 2010] provide discussion forums for new pattern ideas or take into account user feedback in pattern rankings. In order to use these features, a registration is needed. These and other collections like [Patternry 2012; PatternTap 2012] try to attract many contributors from all over the world to share or review patterns.

We consider these approaches as useful means to involve more people in the knowledge creation process. However, we see the danger of loosing commitment to the pattern platforms since they are not directly integrated into the project's working environment. Additionally, we still argue that these online collections are valuable but have to struggle with the inclusion of important stakeholders of the current projects. In our view, mostly people who are directly interested in the pattern domain will contribute the most. Another obstacle from our perspective is that possible participants may feel that their contributions will need a certain maturity for submission.

3.2 Knowledge Generation Approaches Embedded Within the Software Engineering Process

The approach followed by [Averbakh et al. 2011] aims at collaboratively defining best practices within software engineering projects. Project participants are encouraged to formulate knowledge as an early concepts which then develops after iterative review to *best practices*. Based on different roles and rights in within a role pyramid, decisions on the state of the current asset are made. However, [Averbakh et al. 2011] do not structure the best practices in a graph structure. The generated documents reside within a collection without relations to each other or hierarchies concerning abstractions or concretizations.

[Grill and Blauhut 2008] describe the consequent application of design patterns as a step in the software system engineering process. The Frequentis[6] pattern application process in systems engineering partially addresses this issue by integrating the application of design patterns as well as their continuous (re)validation and updating. According to the framework, UI design patterns are identified and validated within the field of emergency management through a user-centered design process. They extend the user interface design process beyond the traditional ones by introducing periodic evaluations of the pattern validity. This results in an up-to-date pattern library containing the experiences of different UI designs in safety-critical environments. The library is available for internal purpose only. The process starts with the creation of an early draft of the UI. This step is based on a previous workshop, held to analyze the requirements with the user. During the latest design phase, the design is evaluated with the users based on their expectations. The process therefore results in an iterative manner to have an ongoing improvement of the UI until no more changes are needed and the UI design process freezes.

Within the design phase, design experts have the possibility to access a pattern library, containing important patterns identified from previous projects. The experts can browse the pattern library and the results are analyzed. As a result of the analysis design patterns are either created if no existing pattern fits; updated if the existing pattern can be further generalized; or reused if the pattern fits the expectation of the experts. The process is only

[5]see http://hillside.net for more information
[6]http://www.frequentis.com

applied during *one* phase of the project and only validated patterns are taken into account for re-evaluation. An iterative documentation process over time as we foresee it (cf. Section 4) is not part of the process.

3.3 A New Approach to Pattern Evolution

We want to tackle the problem of a closed author group in conjunction with delayed design distribution. We also see a lack of early integration of pattern ideas in the process. Thus, we fear that the proposal may get neglected over time. In pattern collections assembled as books, which are also very prominent at the moment, long and well-evaluated experience is gathered and structured before handed to the public.

3.3.1 *Flexible Pattern Language Structure .* The traditional *"pattern guru"* approach ensures a high quality of the generated patterns but could result in a limited and inflexible pattern language structures as discussed in preliminary work where this problem is referred to as "lacking extensibility and openness of knowledge" [Reiners 2011]. Until publishing the patterns, a long time passes in which the design recommendations are not shared with other development groups. Updating the patterns is also rather difficult since the interested community has to wait for updated editions of the print which is not necessarily in planning. This problem was considered as one essential shortcoming in the current approaches in expert workshops that we arranged in initiate our work in our interaction design work packages.

3.3.2 *Integration of Existing Patterns.* In our opinion, the traditional approach is hard to connect to other pattern collections. We want to introduce mechanisms to insert and adapt already proven design context and *reuse* them in other pattern collections. Adaptation may be needed since the original pattern was eventually suited to special needs of a specific domain. This circumstance has to be regarded when formulating the pattern. The insertion may need to adapt some naming conventions and formulations and transform them into the pattern language's terminology.

3.3.3 *Inclusion of Anti-Patterns.* The insertion of anti-patterns as described in [Reiners et al. 2011] also plays an important role since surprisingly failing designs also represent knowledge about flaws that should not be repeated and which were not obvious at design time. Our approach allows knowledge sharing and development during the project as soon as possible. Continuously updating and revalidation steps are already described in the Frequentis UI design library as a mandatory step. Surprisingly failing design concepts should also be documented as anti-patterns.

3.3.4 *Stakeholder Involvement.* Instead of leaving the experience management to a dedicated group of designers (and potential design pattern authors), we want to evolve *all stakeholders* from the beginning of a project in the knowledge creation, adaptation and management process. In order to allow for quickly collecting and initially adapting experience, we lower the threshold of inserting a contribution as a pattern idea very much such that everything following the conventions of a pattern formulation (see below) is firstly integrated into the pattern library.

3.3.5 *Iterative Knowledge Integration.* We plan to include all findings *as early* as possible in the pattern collection. Time, discussion and evaluation needs to show whether the idea is a successful pattern, a surprisingly failing approach leading to an anti-pattern or if the early formulation turns out as not suited to the pattern library and eventually needs reformulation.

The parallel design strategy is reflected by inserting different patterns for problems. Like in other pattern collections, the designer is asked to choose among the suggestions or combine them.

The participatory design approach finds its way into our process by involving a *large group* of potential pattern authors. In our view, every stakeholder of a project has experience and ideas that need to be formulated and collected in the pattern repository. The low threshold for participation intends to involve all stakeholders of the project and to keep the pattern library vivid and dynamic. During the project's lifetime, new experiences, findings, ideas and evaluations will occur that need to be fed back to the design knowledge continuously.

At this stage we do not yet speak of design patterns but early findings and experiences. Over time, we need to ensure the validity and suitability of the formulations by refinements and discussions.

Our proposed process for establishing a dynamic pattern library with a low-participation threshold is described in the following section.

4. THE DESIGN PATTERN EVOLUTION PROCESS

Our process aims at supporting the contribution of new ideas, provision of feedback, concept validation and refinement of design patterns. Thus, we introduce a pattern evolution process as depicted in Figure 3:

Each newly submitted pattern undergoes a first semi-automated quality check process meaning that the system first ensures that all required pattern fields are actually filled. The submission is then forwarded to a internal group of pattern reviewers who decide whether the pattern can be published in the library or if certain formulations need to be changed again by the author. With this quality check, we want to avoid flooding the pattern library with incomplete contributions. In case that a pattern is published it is published in the library and therewith open for discussion.

From now on, every registered user of the system can provide feedback to the pattern, its formulation and support or refute the pattern statement by providing more references in favor of the pattern or against it. Providing qualitative feedback reflecting experiences when applying the proposed pattern encouraged in order to learn about the pattern in practice and to derive later ratings. It is also possible to make suggestions to reformulate parts of the pattern, split it into more patterns or unify it with already existing ones.

This way, the pattern state changes over time. From being born, it may develop to more mature states like under consideration, pattern candidate, applied pattern and approved pattern. We defined these states to reflect the usage and support of the pattern.

Currently, we have not yet defined a measure for the state of a pattern's maturity since the process was just introduced in our project. We need to monitor the user behavior and the kind of feedback we get on the pattern concepts and derive rules for transitions of pattern maturity states. One indicator for a pattern's maturity may be the number of applications of a pattern as used by Grill et al. [Grill and Blauhut 2008].

During a pattern's lifetime and depending on the feedback and validation steps, the pattern (idea) will, over time, turn into a validated pattern, i.e. a design guideline or an advice to avoid certain design alternative. This notion of an anti-pattern is also considered as useful since surprisingly failing concepts also need to be documented in order to avoid similar design flaws in the future, and instead motivate the exploration of new designs. The third alternative is that the initial pattern sketch does not develop to a pattern. it may be the case that an idea was not abstract enough to become a pattern and later reformulations could not improve its quality. Reformulating or merging ideas could result into a resubmission of a pattern idea into the system. Thus, failed formulations do not necessarily lead to the rejection of eventually good ideas.

4.1 Pattern Structure Extensions: State and Origin

Our pattern structure follows the one described by Alexander and others (cf. [Alexander 1977], [Borchers 2001] and [Schümmer and Lukosch 2007]) but extend it by new fields needed to reflect the liveliness and bottom-up approach of the patterns in the design pattern library: A pattern's **origin** and **state**.

The pattern state is used to track the development of the pattern over time. The pattern evolution process includes the following initial pattern states:

—**just created** for patterns that were recently submitted and needs review and evaluation
—**under consideration** means that the pattern looks promising but needs further evaluation
—**pattern candidate** states that the pattern is close to being approved
—**approved** finalizes the pattern review process and settles the pattern as a design pattern. However, this does not exclude the possibility to open the pattern for discussion in the future again due to new findings.

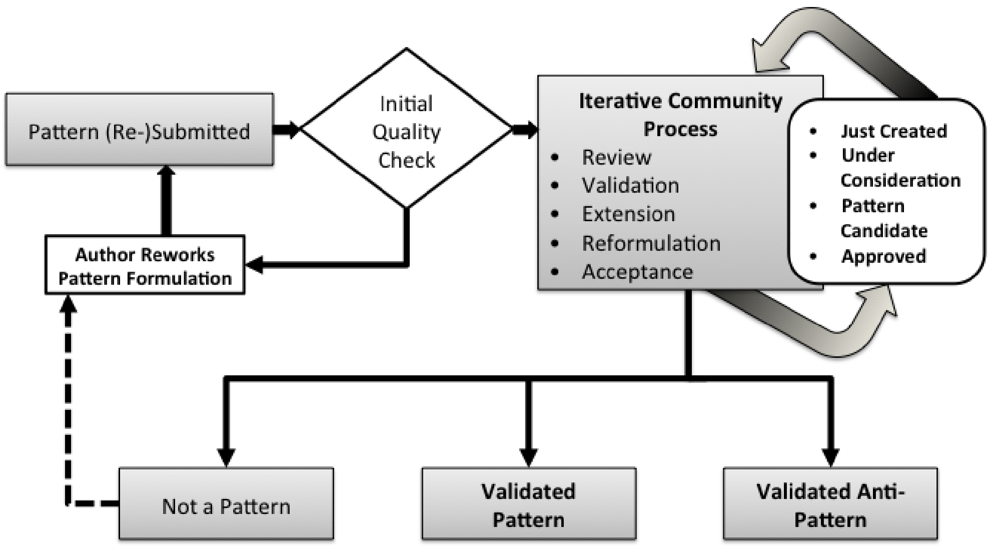

Fig. 3. The pattern evolution process: After a pattern candidate is submitted, it undergoes several maturity transitions and resulting in (anti-)patterns or rejection, with the opportunity to be reformulated.

An already existing pattern, i.e., a pattern originating from an external source, can be approved or rejected in the project domain. If associated with uncertainty it can be tagged as being *under consideration*. Adapted patterns and newly integrated ones must initially be put in the state under consideration (default value) and be validated as a pattern or rejected over time. During a pattern's lifecycle, the different states can be changed due to new consideration or discussion. It still needs to be clarified if a pattern can be fixed disallowing any future changes. Regardless of originating from the project or external pattern, all patterns must be justified in a separate text entry field. Existing patterns must be referenced by their origin (articles, reports, web page or similar).

A pattern idea that fails during validation might be marked as *anti-pattern* which play an important role as a reminder to avoid future implementations of non-trivial (surprising) design flaws.

The patterns in the design pattern library (DPL) can originate both from the project itself and from external sources. We distinguish between three different categories:

—**Derived from project**: The pattern derives directly from the work within the project. Patterns in this category will automatically be assigned the state "under consideration". It needs to be reviewed, eventually refined and validated.

—**Adapted to project**: The pattern originates from external sources, but has been adapted to the the project's context .

—**Project-external**: The pattern exists in other pattern collections (e.g. a standard UI pattern) an is implemented in the current project's products and services.

4.2 Dynamic Structures

We have ideas, guidelines and rules as well as technical assets coming from different contributors and specialists in the work packages. Bringing them all together in one pattern language repository is a challenge.

Therefore, we introduce dynamic hierarchies for the pattern language as illustrated in Figure 4. The only demand for the hierarchy is that it must be concretized towards the lower levels. From our experience so far, it is probable that the top level includes rules and constraints of the domain that must be followed by application designs. Another level below in the hierarchy might deal with official and unofficial guidelines, respectively, that exist in the domain.

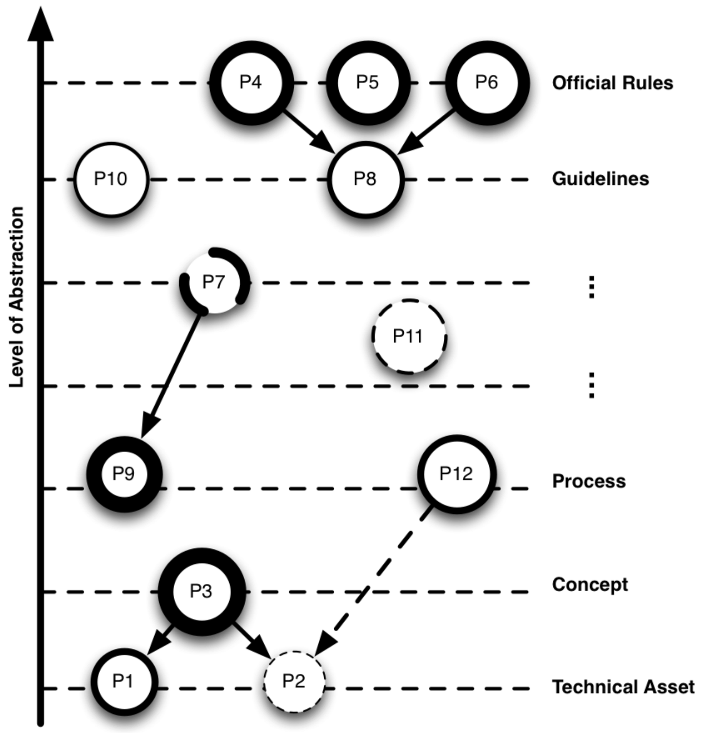

Fig. 4. Possible hierarchies and patterns in different states that are proposed during the evolution of the pattern library.

The next levels could consist of patterns about processes or behavioral descriptions that have their justification in rules and guidelines. On a deeper, closer to the bottom level, application design concepts and finally, technical assets could reside that represent technical solutions formulated in an independent way.

We do not prescribe these hierarchies but want the participants to think about where in the (maybe still premature) structure the introduced pattern could reside or whether an already positioned pattern should be moved to another hierarchy level. For these processes, it is possible for contributors to make proposals for new hierarchy levels.

The same influence is given for pattern relations. The associations between patterns could be obvious in case that there is a subsequent formulation of a pattern from a higher hierarchy or the pattern is an abstraction from one or several patterns from a more detailed level.

Over time, associations can be added or changed like the pattern state (cf. Section 4.1), the associations also need to be confirmed by the community. A change or deletion of associations, respectively, undergoes the same proposal process as newly added associations. The propositions of associations are illustrated as dashed association arrows whereas accepted connections are shown as solid associations. Proposed patterns and their state are visualized according to the thickness or the enclosing circle. Some connections are not yet made to show that it is thoroughly possible that the pattern are not or only partially interconnected. The dynamic structure is intended to reflect the acquired project knowledge over time which is gathered from domain analysis, prototyping and validation (cf. Section 2). Knowledge is generated for different levels within the hierarchy and therefore the structure has to grow iteratively.

4.3 Roles and Access Levels

Since we want to involve a large group of pattern authors we need to keep the access to the design pattern library easy. Therefore, any visitor can browse the design patterns, as well as contribute/suggest new patterns by contacting the library administrators. Registered users can directly submit patterns and comment on existing patterns and the library structure. After submission, a user automatically becomes the pattern responsible and therewith responsible for maintenance (e.g. upgrade) of the pattern in question. Note that the pattern responsible is not necessarily the originator or the inventor of the pattern.

In the following we describe the roles that we introduce in our approach:

Project externals are visitors and readers of the DPL who are allowed to *browse* and *view* the available patterns. In order to *provide feedback* or *suggest new pattern ideas*, the DPL team needs to be contacted.

The user can also register to become a **pattern author** to directly posses these rights. Knowing the users for feedback and suggestions is important to keep track of the changes and to be able to follow on question regarding authorship of the contributions.

The pattern author automatically takes on the role as **pattern responsible**, who cares for maintaining the pattern inside the library. Whenever feedback or change requests are subject of discussion, the pattern responsible is always included in it. This mechanism is applied to make sure that the original intention of the pattern authors is kept.

The DPL **editors** are responsible for the general quality assurance of submitted and updated patterns. Whenever a new pattern is submitted or updates are formulated they represent the initial quality gate (cf. Figure 3) that makes sure that the submission is complete and in a well-understandable state.

The **validator** role is assigned to members are responsible for validating design patterns in custom implementations, find further evidence in literature or available solutions to support or opt against the pattern. Finding evidence in favor of or against a pattern is a stronger mechanism than providing feedback. Here, references need to be delivered.

Usually, a system design following an iterative design approach is strongly founded on demonstration activities of tangible results with a subsequent evaluation and validation process. It is assumed that products associated with a GUI are demonstrated and evaluated during the course of the project lifetime. Users that take on the role as DPL validators are responsible for the validation of design patterns that are under consideration. They do so by incorporating one or several design patterns into an asset, i.e., hardware, software or process.

5. CONCLUSION

In this paper, we have introduced an evolutionary pattern language development process for documenting, sharing, refining and reusing design knowledge for system design which is based on current work-in-progress. Our process extends the usage and creation of design patterns by allowing every stakeholder in a project to read, comment and contribute to the pattern language as early as possible in the project lifecycle. Patterns can be derived directly from the project work or adapted from existing pattern repositories. Everybody involved in the project is a potential pattern author and reviewer. This changes the derivation approach of a pattern language from the expert's point-of-view to a mixed approached in which experts and novices can benefit from each other. Ideas and concepts are brought in as pattern candidates. A community-based iterative process helps developing a pattern candidate to a validated pattern or anti-pattern.

Our test bed for this approach is the BRIDGE project that has the goal to develop technology that supports the inter-ageny collaboration in the emergency response domain. The abstract design knowledge and general guidelines for system design in this domain are continuously captured and updated in an evolutionary pattern library, namely the BRIDGE Design Pattern Library[7]. An early prototype with a first set of 26 derived patterns has

[7]http://pattern-library.sec-bridge.eu

gone online[8]. The concepts described in this paper will continuously be integrated into the platform throughout the year. Pattern collection and formulation workshops together with the project partners will take place during the next weeks. After this initial starting aid, we hope that we will be able to present results about the acceptance and usage of the platform in early 2013. Currently, much effort is spent on better visualizations of the pattern library and the integration of collaboration processes and rules. We plan to integrate a new visualization metaphor by the end of 2012 together with a more elaborated process model for pattern evolution.

Since we follow a collaborative approach, we potentially have different shares of contributions to a pattern regarding authorship. One idea to address this issue is by adapting the CollabReview tool developed by [Prause 2011] that allows for determining the contribution shares to wiki articles or source code files. We also plan to use this tool support to address pattern author's reputations; the CollabReview system dignifies contributions based on quantity and quality. The latter is also determined by feedback from others. We see that this feature can also be connected to the feedback method that we intent to integrate.

New requirements will certainly come up and will be fed into a further design iteration of the evolutionary pattern approach and the technical implementation as design pattern library. We consider the formulations in this paper as constitutional steps for our work in progress. An additional aim of our work is to transfer the findings from our research work to any kind of distributed project that allows for such an open collaboration approach.

REFERENCES

ALEXANDER, C. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York, New York, USA.

AVERBAKH, A., KNAUSS, E., AND LISKIN, O. 2011. An experience base with rights management for global software engineering. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies.* i-KNOW '11. ACM, New York, NY, USA, 10:1—-10:8.

BOEHM, B. 1986. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes 11,* 4, 14–24.

BORCHERS, J. 2001. *A Pattern Approach to Interaction Design* 1st Ed. John Wiley & Sons.

BUXTON, B. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design.* Morgan Kaufmann.

COPLIEN, J. O. AND HARRISON, N. B. 2004. *Organizational Patterns of Agile Software Development* illustrate Ed. Prentice Hall International.

DENEF, S. 2011. A pattern language of firefighting frontline practice to inform the design of ubiquitous computing. Ph.D. thesis, Delft University of Technology.

DOW, S. P., GLASSCO, A., KASS, J., SCHWARZ, M., SCHWARTZ, D. L., AND KLEMMER, S. R. 2010. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Transactions on Computer-Human Interaction 17,* 4, 1–24.

FORNELL, C., S., M., MORGESON III, F. V., AND KRISHNAN, M. S. 2006. Customer satisfaction and stock prices: High returns, low risk. Journal of Marketing. *Journal of Marketing 70,* 1, 3–14.

GARFINKEL, H. 1967. *Studies in Ethnomethodology.* Prentice Hall Inc., Englewood Cliffs, New Jersey, US.

GOGUEN, J. A. AND LINDE, C. 1993. Techniques for requirements elicitation. *Requirements Engineering*, 152–164.

GRILL, T. AND BLAUHUT, M. 2008. Design Patterns Applied in a User Interface Design (UID) Process for Safety Critical Environments (SCEs). In *HCI and Usability for Education and Work*, A. Holzinger, Ed. Lecture Notes in Computer Science Series, vol. 5298. Springer Berlin / Heidelberg, 459–474.

HUGHES, J. A., RANDALL, D., AND SHAPIRO, D. 1992. From Ethnographic Record to System Design: Some Experiences from the Field. *Computer Supported Cooperative Work 1*, 123–141.

INFRAGISTICS. 2012. The Quince Pattern Collection.

INSTITUTE, P. 2008. *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)* 4 Ed. Project Management Institute, Newtown Square, Pennsylvania, USA.

KANO, N., SERAKU, N., TAKAHASHI, F., AND ICHI TSUJI, S. 1984. Attractive Quality and Must-Be Quality. *Journal of the Japanese Society for Quality Control 14,* 2, 147–156.

KENSING, F. AND BLOMBERG, J. 1998. Participatory Design: Issues and Concerns. *Computer Supported Cooperative Work (CSCW) 7,* 3, 167–185.

NIELSEN, J. 1993. Iterative user-interface design. *Computer 26,* 11, 32–41.

PATTERNRY. 2012. Patternry.

[8]Situation in September 2012

PATTERNTAP. 2012. Pattern Tap.

PRAUSE, C. R. 2011. Reputation-based self-management of software process artifact quality in consortium research projects. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.* ESEC/FSE '11. ACM, New York, NY, USA, 380–383.

REINERS, R. 2011. An Extended Pattern Language Approach for UbiComp Application Design. In *Lecture Notes in Informatics Informatiktage 2011*. GI - Gesellschaft der Informatik, Bonn, Germany, 4.

REINERS, R., ASTROVA, I., AND ZIMMERMANN, A. 2011. Introducing new Pattern Language Concepts and an Extended Pattern Structure for Ubiquitous Computing Application Design Support. In *PATTERNS 2011, Third International Conferences on Pervasive Patterns and Applications*, F. Laux, R. Reiners, and A. Zimmermann, Eds. Number c. XPS - Xpert Publishing Services, Rome, 61–66.

ROYCE, W. W. 1970. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, 1–9.

SCHNEIDER, K. 2009. *Experience and Knowledge Management in Software Engineering* 1st Ed. Springer Berlin Heidelberg, Berlin, Germany.

SCHÜMMER, T. AND LUKOSCH, S. 2007. *Patterns for Computer-Mediated Interaction*. John Wiley & Sons, Chistester, West Sussex, England.

SOMMERVILLE, I., RODDEN, T., SAWYER, P., AND BENTLEY, R. 1993. Sociologists can be surprisingly useful in interactive systems design. In *Proceedings of the conference on People and computers VII*. HCI'92. Cambridge University Press, New York, NY, USA, 342–354.

SUCHMAN, L. 1987. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University Press, New York, New York, USA.

TIDWELL, J. 2005. *Designing Interfaces* 1 Ed. O'Reilly Media.

TOXBOE, A. 2012. UI Patterns - User Interface Design Pattern Library.

VAN WELIE, M. 2010. The Amsterdam Pattern Collection.

VILLER, S. AND SOMMERVILLE, I. 2000. Ethnographically informed analysis for software engineers. *International Journal of Human-Computer Studies 53,* 1, 169–196.