Engaging Patterns: Challenges And Means Illustrated By An Example

Sabine Niebuhr Kirstin Kohler Christian Graf

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
{kohler, niebuhr, grafc}@iese.fraunhofer.de

Abstract. This paper presents first results of a research project whose goal is to develop a pattern language that enhances business software by motivating and engaging elements. The goal of the pattern language is to turn the soft and vague term of "emotions in user interaction design" into constructive design guidance. The patterns are especially tailored for joy-of-use in business applications. The main contribution of this paper is the description of quality characteristics for this pattern language. They are illustrated by references to existing pattern descriptions and elaborating their deficiencies. This paper shows how these weaknesses were addressed in the pattern language.

ACM Classification Keywords. D.2.1 Requirements/Specifications, D.2.2 Design Tools and Techniques, H.5.2 User Interfaces

1. Introduction

Using patterns (originally introduced in architecture [1, 2]) for developing software is well established [3] and still up-to-date [4]: Why reinvent the wheel if solutions for a problem are already known and approved? Many pattern languages exist for nearly every developing step – e.g., for designing the interaction and the user interface [5-7], or for the software implementation [3]. But for a software developer, applying patterns is not as simple as one might assume.

Let us imagine a software developer who wants to design a user interface. He has found some interaction patterns on the Web and hopes they will help him. Trying to apply these patterns he first has to find an appropriate pattern. This is a big problem to overcome, since matching a specific design problem to the problem descriptions in existing patterns is a question of interpretation. After the software developer is convinced that the pattern he has identified matches his problem, he tries to understand the author's recommendations – how does the author think this problem can be solved? The software developer might not see the correlation between the problem statement and the solution described in the pattern: the problem statement matches his problem, but the solution does not make any sense to him. After interpreting the recommendation our software designer applies the pattern in the way he thinks it would be the author’s intention.

Let’s assume that the software developer has another problem and therefore searches for a fitting pattern a second time: he finds two patterns that are nearly the same – so which one does he have to apply? How do they relate to each other? Does one specialize the other? Do they have different conditions when to apply? The software developer might not be a very patient person, so he stops searching for another pattern and tries his best on his own – without any guidance or implementation advice. What went wrong?

- The pattern descriptions were not concrete enough.
- The developer did not find the right pattern to apply.
- The developer did not understand the pattern idea.

These are just three problems. For us as authors of patterns, this means: Do not repeat existing defects. Identify the developer’s problems with patterns and fix them!

In our project, we try to identify patterns to enhance business software by elements that motivate and engage. In writing down these patterns, several challenges had to be mastered– from setting up a pattern language with all of its attributes and relations up to the internal validation of the patterns and the problem of making it discoverable. When we performed a search in the literature, we mostly found solutions to the syntax problems of a pattern language - how to build up relations and attributes – but no answers to our semantic questions, for example, how we can formulate our patterns in an understandable way. We found the Pattern Language Meta Language (PLML) [8], and we found approaches that name our challenges – e.g., MetaPatterns [9], patterns for writing patterns – but we did not find any real solution for our problems (we will discuss this in chapters 3 and 4).

In this paper, we demonstrate our challenges and how we mastered them with an example pattern. Our contribution consists of defining quality characteristics for pattern languages thar base on our challenges and approaches to master them.

We will first describe our project context to give you an idea of our work: writing engaging patterns. Then we will describe the challenges that came up while writing these patterns, which led us to quality attributes. Since we think it would be easier to understand how we mastered the challenges by reading examples, we introduce an excerption of our pattern language, which is still work in progress. Finally we present our approaches for mastering the described challenges and what we will be doing next.

2. Project Context

The work presented here is part of a three-year research project funded by the German federal government entitled ‘FUN’ (acronym for “**f**un-of-**u**se in Geschäftsa**n**wendungen”)[1]. In the project, three industrial partners and Fraunhofer IESE deal with the topic of “fun-of-use for business applications”. One goal of the project is to develop a pattern library that captures fun-of-use interaction pattern. The

[1] You can find more detailed information about the project at http://www.fun-of-use.de

research work is closely related to the needs of the industrial partners in order to ensure the usefulness of the results for industry.

The challenges given for the pattern library are motivated by our project context: As part of the project, a call center software has to be redesigned in order to improve users engagement with the software. The software helps agents to solve incoming support calls from people complaining about trouble they have with the product. The work of the agents is kind of frustrating and monotone, which results in a loss of motivation. As a consequence, agents are inefficient, make more mistakes, and take fewer calls.

In the first step of the project, we were looking for existing interaction patterns, that might help us to solve the problems of the call center agents as described above. We found two promising candidates: the status display [5] (listed in Table 1) and the high score list [10]. While searching for patterns and applying them to the software described above, we start doubting that a “software engineer” would have been successful in doing this. We are experienced user interface designers/usability specialists well familiar with the concept of “interaction patterns”. Would a software engineer have found the high score list or status display pattern and would he/she have been able to derive an adequate solution for the software from the description? We turned this impression into a challenge for our project. We investigated effort in extracting “quality requirements” for the pattern library. These quality requirements or challenges will be elaborated in the next section.

3. Quality challenges for pattern languages

We set up a list of characteristics that we believe are required to support software engineers in creating “engaging” user interfaces. To provide valuable support, our pattern collection has to assist the engineer during the following steps:

Step A - Pattern Discovery: The engineer has to find a pattern to the given user interaction problem. The library is intended for software engineers respectively requirements engineers, who design the user interaction as part of the requirements phase. We assume that they follow a task-oriented approach, which means the requirements for the system to be developed are stated as “tasks”. In addition, “non-functional requirements” or business goals are part of the requirements.

Step B - Pattern Application: During this step, the software engineer has to apply the solution given by the pattern, which is often still on a quite abstract level, to a concrete interaction realization.

Looking at these two steps in more detail, we identified a set of four quality requirements for our pattern language. Theses requirements consider quality needs stated by other authors [11, 12], but extend and combine them to address all the problems we investigated. We will explain them by expanding the problems we faced in our project.

3.1 Problem Fit

The pattern language has to guide the user from the problem to the solution; the pattern should be stated in a way that the user can match his problem and project context to the pattern description. This might, on the one hand be a problem of the entire pattern language; the way the pattern are linked or put into hierarchies might not be useful for the engineer. And/or it might be a problem of the individual pattern itself – the pattern description does not give a clue to the real world problem.

The problem in our case was described by the information given in the use case description of the requirements document and the "undesired" behavior of the agent "losing motivation" (which is derived from the business goal "improve agents' job satisfaction". The existing description of the "status display" does not give any idea that it might improve the agents' motivation.

3.2 Understandability

This challenge belongs to steps A and B. The wording and notation of the pattern description has to be understandable for the engineer; otherwise, he will neither be able to identify nor to apply the pattern. What does this mean more concretely?

The reader should interpret the words that describe our pattern in such a way, that he understands the idea behind it and the intention we as authors had in writing this pattern. This means that we have to write unambiguously, so that the reader will not misinterpret the content, and we have to write completely and without contradictions, in order to avoid different interpretations. Here the challenge is: How can we ensure this?

Understandability is also closely related to readability. So another aspect is a syntactical aspect, which supports the readability and understandability of our patterns: the attributes that describe them. Therefore, we searched in literature and found PLML [8], on which many people worked for gaining a uniformed, standardized Pattern Language. This is a very helpful aspect indeed: The reader gets this patterns formulated in the same pattern language, so he knows where to find the context, the problem and the solution. But this approach is not really finished: Many people are still working on this language. However, although definitions for attributes and how to fill these attributes exist, they are not sufficiently defined, leaving out which kind of content can be found in the "context" attribute and which in the "problem" attribute. What would solve this problem?

3.3 Correctness

We want to describe patterns that will motivate or engage users. How can one ensure that the desired effect of a user's engagement or motivation really takes place? Is there any theoretical background that guarantees that the given solution (such as the status display) encourages users to continue their task? Does showing status information really influence the users' motivation? Todd et al. (2004) talk about the "internal validity" of an individual pattern. We define it as the relationship between the description of the problem and the solution: The solution must solve the problem in the given context.

For a lot of described patterns, the way the patterns are phrased makes this step trivial. For example, the problem of the status display is expressed as “How can the artifact best show the state information to the user”, the solution says “Choose well-designed displays for the information to be shown….”. The topic of our pattern language covers emotional effects (like motivation, engagement, fun) and therefore makes it more important to either empirically prove the evidence between “Problem” and “Solution” of a pattern or relate it to one or more psychological theories.

3.4 Concretization

Assuming he had found the problem, the task of the developer would be to transfer the pattern description, which is quite abstract, to a concrete solution for the call center software. How can one ensure that this concretization still solves the problem? There are often minor differences in design that make a big difference in the desired effect.

Assuming that while detailing out the user interface for our call center someone had the idea of putting in a kind of “ranking” that shows the performance of each agent compared to the others in terms of “time to fix a support call”. At first glance, one can assume that this kind of ranking would lead to competition between the agents and keeps them motivated. And on the abstract level of a pattern, this assumption might be right in terms of Correctness. Unfortunately, this solution destroys the social relationship between agents and enforces the “Galley Slave Model” [13]. As a consequence dissatisfaction and turn-over of agents increase. As stated before, the intention of the user interface redesign was to increase agents’ satisfaction. Another problem with concretization is that a software engineer reading the “status display” as it is might not even have an idea, what range of freedom he has in bringing it to a concrete solution – showing a “progress bar” is not the only way of representing “status”, as we will show in the next section.

The first two quality requirements (Problem Fit and Understandability) address step 1, “find a pattern”, whereas step 2 is related to the quality requirements Understandability, Correctness, and Concretizaton.

4. Engaging Patterns

We would not have been able to concretize these problems if we did not have the idea of writing down patterns to support developers in designing and implementing user interfaces containing motivating elements – elements that help users stay concentrated on their work tasks. For detecting patterns that engage we looked into existing pattern languages as well as into the literature for e-learning and game design. Especially in these disciplines, much time has been spent on developing applications that capture the user, because these applications depend on the user keep on using them voluntarily. We now try to apply this knowledge to business application design.

Some of our engaging patterns can be specialized from the existing usability pattern “Status Display” (see Table 1), established by Jennifer Tidwell in [5]. An overview of the patterns that could be specialized from Tidwell’s “Status Display” is given in Figure 1. Boxes with a grey background contain patterns described in the literature, boxes with a light green or blue background cover patterns specialized by us, boxes with a dark green background are examples for concrete implementations.

.

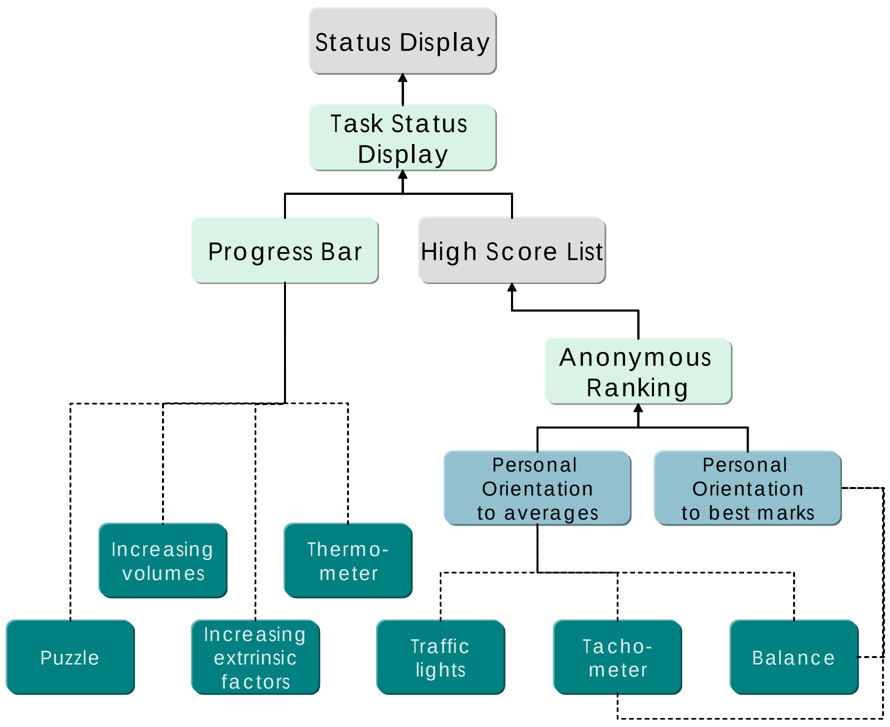

Fig. 1. Hierarchy of Status Display patterns with examples of concrete implementations.

The pattern “Task Status Display” proposes a solution for showing any kind of information concerning the user’s task. The pattern “Progress Bar” as a specialized “Status Display” shows this information in relation to a specific goal. The pattern “High Score List” (this comes out of game design) shows information concerning the work task (for example, performance data) as a specialized status display in relation to other performance data. This data can show performance of other people, statistical values, or values that should be achieved.

A specialized “High Score List” is an “Anonymous Ranking”. Normally, in high score lists, names mark the presented information. This could cause some group effects or discouraging effects, so in some applications, names should not be mentioned. The idea of this pattern can be specialized in a personal ranking – a personal orientation from which the user gets information about his personal performance data related to an average value or related to personal or group-wide best marks.

To give a better idea of how these patterns can be implemented, we display some concrete examples: In the first example, an “Anonymous Ranking” is implemented as a traffic light (see Figure 2a). A “Progress Bar” could be implemented as increasing or decreasing volume, for example as a card stack (see Figure 2b). The picture originates from an application where the user has to fill in an address database. Every time he enters an address, the set of cards in the picture is reduced by one card. Another example is the idea of a puzzle, like the example in Figure 2c), which originates from a computer configuration tool. The puzzle completes a little more every time a user adds one part to a computer. In some companies the employees receive certain incentives – extrinsic motivating values – which can be visualized by a progress bar (see Figure 3).

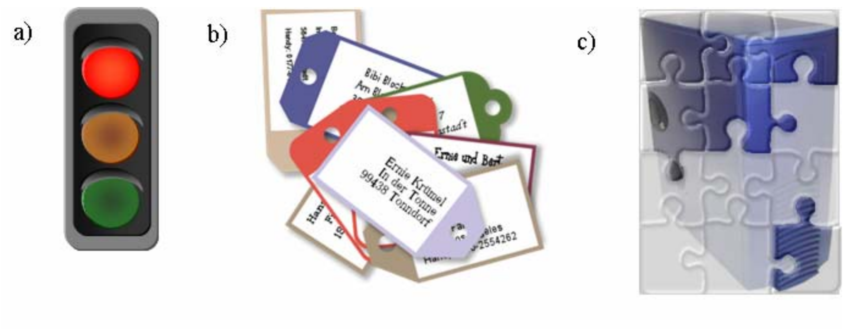

Fig. 2. Different solutions for the “Progress Bar” to display the task status: a) as traffic light, b) as card stack c) as a puzzle.

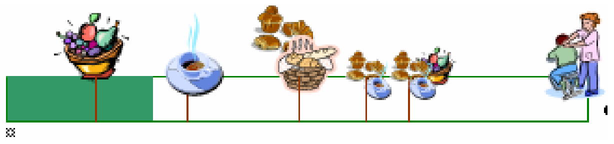

Fig. 3. Progress Bar displaying the status plus the rewards that can be expected when reaching certain degrees of completion.

In the following, you will read more about approaches we found to master the challenges encountered while writing down these patterns.

4.1 Problem Fit

To guide the engineer from his “real world” problem to the pattern solution, our pattern library followed two strategies:

- The hierarchy of patterns (given by the relationship between them) within the pattern language and
- The pattern description of individual patterns.

The hierarchy of patterns guides the engineer from more general patterns to more specific patterns. This helps to "narrow down" the appropriate patterns by matching them to the various context/problem fields of more specific patterns. Figure 1 illusstrates this hierarchy for an excerpt of our pattern language.

The second strategy to improve "problem fit", covers the pattern description of individual patterns. By giving the descriptions of single pattern attributes a more specific semantic ____ (?), patterns can be integrated into a task-oriented requirements approach. This facilitates the "detection" of the appropriate pattern in a natural way. The engineer matches the requirements given by the project to the problem and context section of the pattern descriptions. This means in more detail:

- Individual pattern state the non-functional requirement they contribute to.
- The engineer should be able to mach these non-functional requirements to the business goals that characterize his project.
- The context of a pattern contains fields characterizing the user type, the task, the environment, all the elements that belong to a contextual design. By specifying the context as "completely" as possible, we try to prevent the engineer from applying a pattern that does not fit the "real world" problem.

4.2 Understandability

The first question is: How can we formulate patterns unambiguously? Meszaros and Doble propose to find out who the audience is and to focus on it with wording and notation [9]. This is a helpful approach, but it is not sufficient for solving our problem: We have software developers who (hopefully) will implement our patterns as well as psychologists or graphic designers. By describing several interactions through the use of UML activity diagrams, the software developer gets an exact idea of how to solve the problem, whereas the graphic designer just reads some strange symbols. Thus, for usability aspects we have a broad audience. To ensure that every reader will understand our ideas behind the patterns, we will have to use natural language, which is often ambiguous or badly structured.

The solution of the "Status Display" pattern starts with a sentence in natural language: "*Choose well-designed displays for the information to be shown*". What is meant by "well-designed" and which information should be displayed? This example was just the first sentence of the pattern's solution.

In software engineering, the same problem of a broad audience exists at the beginning of a software project: Requirements for this project have to be defined and written down in a way that guarantees understandability for the software developer as well as for the customer. And this customer might be a dentist or a mechanic, with totally different knowledge and background. Rupp and Götz [14] dealt with this topic in requirements engineering and identified three main problems of natural language used for defining requirements: distortion, generalization, and deletion.

A whole process described as a single event in the textual description leads to distortion and misinterpretation. The problem of generalization can be described as trying to derive a more general description based on your experience while neglecting exceptions. Deletion often occurs when information expected to be well-known by everyone is left out. Therefore, Rupp and Götz propose rules to detect these problems and delete them. One way to keep it simple from the beginning is to use some structured sentences, a pattern for building sentences, which aids readability. Now we propose to use these rules and structured methods that exist for writing down requirements to write down the content in our patterns unambiguously, completely, and without contradictions.

Coming back to the “Status Display” example, we would formulate the solution a little bit more concretely (see “Task Status Display” in Table 3): “Display the task’s state information. […]. Display the information the user needs at a glance.”

Another helpful thing to prevent misinterpretation is to keep the vocabulary constant and simple. Sure, normally it is good style to call the user “user” the first time, “driver” the second time, and else third time something to avoid repeating the words too often. But the reader may ask, whether there are three different users. So why don’t we call our user – if he is a driver – a driver every time we talk about him? It does not sound very nice, but it increases readability. This is why the sentences in our pattern descriptions always look the same: “Display…Display…Display…” instead of “show… paint… draw…display…”

Let us now proceed from the vocabulary aspect to the syntactical aspect that assists readability and understandability of our patterns: The attributes described in PLML [8] should be defined more exactly. They should be differentiated to make clear which content can be found in a specific attribute - especially the attributes “context” and “problem”. For finding a pattern, both attributes have to be read, but the first look should be focused on the problem.

4.3 Correctness

We want to ensure that our patterns are correct, meaning the solution described as part of the pattern solves the problem given in the problem field. We try to achieve this quality characteristic by rationalizing the pattern with psychological theories. Most of these theories describe relationships between triggers and effects. We conducted a literature survey as part of our project, scanning theories that describe triggers for positive emotional reactions like motivation, creativity, and fun. The triggers specified by such theories have to be related to the “solution” part of the patterns. If pattern solutions are design examples for such triggers, they might lead to the desired effect specified in the theory. As a consequence of the effect, the problem stated in the pattern is solved.. Figure 5 illustrates this in an abstract way. Effect and problem are related (indicated by circles but in different colors, because the problem is the “negation” of the effect) and the trigger and solution are associated (indicated by the star),

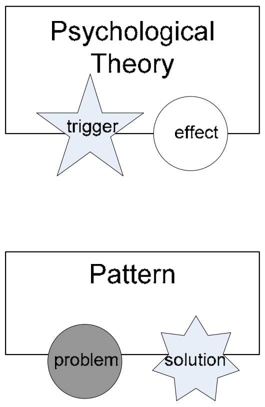

Fig. 4. Relationship between psychological theories and pattern description

To guarantee correctness in the case of the (task) status display, we will consult two different theories that back this approach with psychological reasoning.

Herzberg's two-factor theory proposes that after having compensated for all the unmotivating factors at the workplace (like uncomfortable workspace, bad relationship with the boss etc.) a person will be in an equilibrium, a neutral state [15]. Beginning in that state, one might try to gain satisfaction through 'motivators' while at work (this is the desired "effect"). Some of these motivators are: performance, being responsible, pay, or promotion (these are the "triggers").

The second supporting theory is the goal setting theory [16]. The central statements of this highly recognized and empirically proven theory are as follows:

– Setting goals that are difficult to achieve leads to higher performance than the setting of easy goals.
– Setting specific goals leads to higher performance than the setting of vague, unspecific or no goals.

Both statements have been supported widely by other researchers and are known to have high external validity, i.e., findings can be transferred to diverse settings, like groups and single persons, different task types, and different cultures [16, 17]. The most important factor in this respect is the complexity of the task. The completion of an easy task can be more successfully supported by goal setting than that of a difficult task. This results from different effects. One is that complex tasks need more efforts and take longer so that the effect of the single effort is not directly visible as performance.

Complementing the goal setting, giving feedback is recognized as an important factor [18]. Feedback transfers information back to the user, so that he knows what he has achieved and how he might possibly adjust his actions. Feedback can motivate because the person notices that earlier set goals have been achieved and this tendency will hopefully last. This results in ongoing or even increase motivation.

Applying either goal setting or feedback might not necessarily result in any performance increase. The maximum effect is reached when combining compulsory goals and related feedback [19].

4.4 Concretization

The challenge of concretization is addressed by two contributions:

- The problem is on a higher level of abstraction than the solution description. This means the solution summarizes design decisions and is therefore closer to the final solution than the given problem. We show a large variety of different concretizations for a given pattern. As one possible concretization for the "progress pattern", we have several very different examples as shown in Figures 2-5. This should open the engineer's thinking to further creative concretizations of the same problem. At the same time, it already provides such a wide range that it might be easy to simply pick one of the solutions
- By working out a variety of different concretizations, we were able to state the commonalities between the variants more clearly. This helped us to make the description of the solution more precise. For the solution part of the "progress pattern" is very precise in listing the user interface elements that have to be defined. It lists elements like "task", "goal", "stating point" etc. All these are variables the engineers has to define through concrete values when developing a concrete user interface solution. The likelihood that a engineer derives a solution from this description, which is not a correct concretization of the "progress bar", is very small.

Table 1. The pattern "status display" as found in [5].

Name	Status Display
Context	The artifact must display state information that is likely to change over time, especially if that state information represents many variables.
Problem	How can the artifact best show the state information to the user?
Forces	– The user wants one place where he knows he can find this state information. – The information about it should be organized well enough so that the user can find what the needs at a glance, and can interpret it appropriately. – It needs to be unobtrusive if the information is not critically important, but... – It does need to be obtrusive if something important happens.
Solution	Choose well-designed displays for the information to be shown. Put them together in a way that emphasizes the important things, deemphasizes the trivial, doesn't hide or obscure anything, and prevents confusing one piece of information with another. Never rearrange it, unless the user does it himself. Call attention to important information with bright color, blinking or motion, sound, or all three -- but use a technique appropriate for the actual importance of the situation to the user
Resulting Context	If there is a large set of homogeneous information, use High-

	density Information Display and the patterns that support it (Hierarchical Set, Tabular Set, Chart or Graph); if you have a value that is binary or is one of a small set of possible values, use Choice from a Small Set. Visually group together discrete items that form a logical group (Small Groups of Related Things), and do this at several levels if you have to. For example, date and time are usually found in the same place. Tiled Working Surfaces often works well with a Status Display, since it hides nothing -- the user does not need to do any window manipulation to see what they need to see. (You might even let the users rearrange the Status Display to suit their needs, using Personal Object Space.) If you don't have the space to describe what each of the displayed variables are (e.g., Background Posture), or if your users are generally experts who don't need to be told (e.g., Sovereign Posture), then use Short Description to tell the users what they are.

Table 2. The "Status Display" pattern explicated for one task.

Name	Task Status Display
Context	The user wants to fulfill a task. The artifact must display state information that is likely to change over time, especially if that state information represents many variables.
Problem	The user needs an orientation on how far he has come with his task.
Forces	– The user wants to see the task's state information. – The state information should display information the user needs at a glance. – The state information should be appropriately interpretable. – If the information is not critically, the state information should be too unobtrusive. – If the information is critically, the state information should be obtrusive. – Information is critically, if something important happens.
Solution	– Display the task's state information. – Always display the information in the same place. – Display information the user needs at a glance. – Display the state information in an appropriately interpretable way. – If the information is not critical, display the state information unobtrusively. – If the information is critical, display the state information obtrusively.
Rational	Herzberg's two-factor theory (Herzberg 1959) ; Goal setting theory (Schmidt & Kleinberg 1999)

Resulting Context	The user gets orientation on how far he has come with his task. The user is able to estimate his task status.

Table 3. The „Progress Bar“ pattern [20].

Name	Progress Bar
Context	The user is working on a **task**. The user knows the task’s **goal**. An employee has to achieve different goals at **work**. The work has **one or more defined goals**. The work can be **dreary** or **long lasting**. An employee has to fulfill different **tasks** at work. The task has one ore more defined goals. The task can be dreary or long lasting.
Problem	The user loses sight of the goal. The user needs to be **reminded what the goal is** about.
Forces	See forces from the pattern “Status Display”. Additionally: – The displayed information should contain the goal. – The displayed information should contain the distance to the goal. – The displayed information should contain the scale of the movement into a direction. – The displayed information should contain the starting point. – The displayed information should contain the distance to the starting point. – The information should contain if the user draws nearer to the goal.
Solution	See the solution from pattern “Status Display”. Additional: – Display the **task**. – Display **goal**. – Display the **starting point**. – Display the **distance from the starting point**. – Display the **distance to the goal**. – Display the scale of the movement into a direction (**step width**). – Display the **direction** of the movement (if the user draws nearer to the goal)
Resulting Context	The user won’t lose track of the goal. The user can see if he draws nearer to this goal. The user can see how far he is away from the goal. The user can see how far he is away from the starting point. The user is able to estimate his work progress from this data. The user is able to estimate the remaining time.

5. Next Steps

After having identified promising approaches from other disciplines that have proven to engage users, we will produce empirical results that show how well these ideas were transformed into effective means for motivating in the particular context – into high quality patterns that work.

With each specific implementation of an idea, we will undergo a thorough validation process. The process will consist of two phases: First, we are going to check in a laboratory setting if the result of the particular implementation of a pattern satisfies the “intended outcome” section of the pattern description. If the result is as intended the pattern can be viewed as valid (for this context). Second, the pattern will be tested in a field study with a group of real users. These users will be from the target audience of the enhanced application and will be trained to work with a basic version of the application. Thus we want to avoid effects of curiosity or learning effects that might distort or spoil the result of the analysis. In the field study, we want to learn if the application can transfer its motivational nature to the target audience. It will show whether the realizations of patterns are understood and up to what level of abstraction (as some patterns are very basic - e.g. the status pattern - others are more high-level).

With the results from the first evaluation, we are planning to try out other patterns originating from the games context or e-learning context. We expect that not all ideas from those specific contexts will be beneficial in the target domain. As a result, a pattern language with multiple relations like “contributes to”, “is supported by” or “is suspended by” will evolve for the domain of information services.

Having learned about patterns in one domain it will be challenging to look for possible transfer into other domains in the same way as interaction patterns [7, 21] can be found in different domains like the Web [22, 23] or mobile devices [24]. That question will be a topic of future research.

One practical aspect of our research – current and upcoming – is the process integration of the present and future patterns into the daily work of software engineers. We strive for a beneficial, yet easy, handling of patterns in the context of use. To support developers, we have started the development of a plug-in for the Eclipse Framework (www.eclipse.org). As an open source platform with a thriving community, it is highly suitable for an effort such as deploying and actively developing a pattern library. Let developers and users of software be engaged by patterns that engage!

Acknowledgements

This work is supported by the German Federal Ministry of Education and Research (BMBF) within the project FUN (Grant: 01 IS E06 A). Some of the patterns presented were conceptually designed with our project partner a3systems GmbH (www.a3systems.com), who also contributed the implementation in a real-world business application.

References

[1] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language : Towns, Buildings, Construction*: Oxford University Press, 1977.

[2] C. Alexander, *The Timeless Way of Building*: Oxford University Press, 1979.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Boston: Addison Wesley, 1994.

[4] E. Gamma, "Design patterns: ten years later " in *Software pioneers: contributions to software engineering* Springer-Verlag New York, Inc., 2002 pp. 688-700

[5] J. Tidwell, "COMMON GROUND: A Pattern Language for Human-Computer Interface Design," vol. 2006, 1999.

[6] M. van Welie, "The Amsterdam Collection of Patterns in User Interface Design," vol. 2006, 1999.

[7] J. Borchers, *A Pattern Approach to Interaction Design*. Chichester, England: John Wiley & Sons, Ltd, 2001.

[8] S. Fincher, "CHI 2003 Workshop Report - Perspective on HCI Patterns: Concepts and tools (introducing PLML)," *Interfaces*, vol. 56, pp. 27-28., 2003.

[9] G. Meszaros and J. Doble, "Metapatterns: A pattern language for pattern writing," presented at The 3rd Pattern Languages of Programming conference, Monticello, Illinois, 1996.

[10] S. Björk and J. Holopainen, *Patterns in Game Design*: Charles River Media, 2004.

[11] E. Todd, E. Kemp, and C. Phillips, "What makes a good user interface pattern language?," presented at Proceedings of the fifth conference on Australasian user interface - Volume 28, Dunedin, New Zealand, 2004.

[12] W. Cunningham, "Tips for writing Pattern Languages," 1994.

[13] N. Kjellerup, "The Galley Slave Model," in *Call Centre Know How Essays: Productivity, Measurements & Benchmarks*, vol. 2006, N. Kjellerup, Ed. Ashgrove, Queensland, Australia: Resource International Pty Ltd, 2005.

[14] C. Rupp and R. Goetz, "Linguistic Methods of Requirements-Engineering (NLP)," presented at Proceedings of the European Software Process Improvement Conference (EuroSPI), Denmark, 2000.

[15] F. Herzberg, *The motivation of work*: John Wiley & Sons, 1959.

[16] K.-H. Schmidt and U. Kleinbeck, "Funktionsgrundlagen der Leistungswirkungen von Zielen bei der Arbeit," in *Emotion, Motivation und Leistung*, M. Jerusalem and R. Pekrun, Eds. Göttingen: Hogrefe, 1999, pp. 291-304.

[17] G. P. Latham and T. W. Lee, "Goal setting," in *Generalizing from laboratory to field settings*, E. A. Locke, Ed. Lexington, MA: Lexington Books, 1986, pp. 101-117.

[18] K.-H. Schmidt, *Motivation, Handlungskontrolle und Leistung in einer Doppelaufgabensituation*. Düsseldorf: VDI-Verlag, 1987.

[19] E. A. Locke and G. P. Latham, *A theory of goal setting and task performance*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[20] S. Niebuhr, C. Graf, and D. Kerkow, "Pattern für Fun-of-Use im Kontext einer Service-Center-Anwendung," Fraunhofer-IESE, Kaiserslautern, Germany, IESE-Report 154.06/D, 2006.
[21] M. van Welie and H. Trætteberg, "Interaction Patterns in User Interfaces," presented at 7th. Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA, 2000.
[22] M. van Welie, "Patterns in Interaction Design - Web Design Patterns," vol. 2006, 2006.
[23] I. Graham, *A Pattern Language for Web Usability*: Addison-Wesley Longman Publishing Co., Inc., 2003.
[24] M. van Welie, "Patterns in Interaction Design - MobileUI Design patterns," vol. 2006, 2006.