A Usability Evaluation Pattern Language

Michael Gellner& Peter Forbrig

University of Rostock, Department of Computer Science, Software Engineering Group, Albert-Einstein-Str. 21, 18051 Rostock, Germany

{mgellner, pforbrig}@informatik.uni-rostock.de

Abstract: In most of the cases usability evaluations are done by usability experts. Employing such experts requires a certain size in business. So in a lot of small and middle sized companies developers are forced to learn how to handle usability aspects. This is not much easier than teaching usability engineers how to develop software. The usability evaluation process and its requirements also miss usable advices. As a solution, a lightweight usability evaluation model for software developers is created. This model is described by a pattern language.

Keywords: pattern language, usability engineering, usability testing, usability evaluation, interactive systems

1 Modeling Usability Evaluations and Usability Engineering

We start with an introduction given in a proposal (Gellner 2003 d). One of the most comprehensive views is given by Mayhew (Mayhew 1999). As shown in Figure 1 her lifecycle considers all aspects from the first steps to the successful installation of the software. Fulfilling the requirements of this lifecycle will lead to a complex organizational structure. Further this process model does not work well if several alternations are possible. These requirements are hard to fulfill for a small or middle sized company. A whole staff is

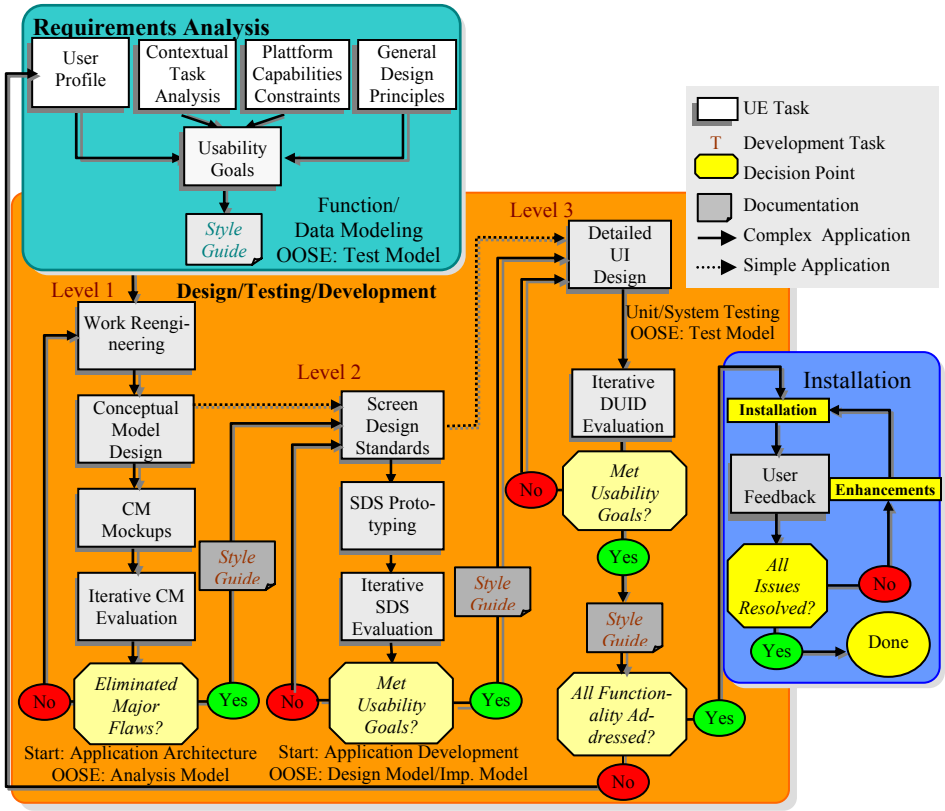


Figure 1: *Usability Engineering Lifecycle (Mayhew 1999)*

1. Know The user
   a. Individual user characteristics
   b. The user's current and desired tasks
   c. Functional analysis
   d. The evolution of the user and the job
2. Competitive analysis
3. Setting usability goals
   a. Financial impact analysis
4. Parallel design
5. Participatory design
6. Coordinated design of the total interface
7. Apply guidelines and heuristic analysis
8. Prototyping
9. Empirical testing
10. Iterative Design

Figure 2: *Usability Engineering Model (Nielsen 1993)*

necessary to manage the various tasks. Mayhew's lifecycle is directed to usability experts or decision makers in a bigger environment that wants to establish a usability department. This approach seems not to be applicable for a small team of developers.

Similar to Mayhew's *Usability Engineering Lifecycle* is Nielsen's *Usability Engineering Model*, see Figure 2 (Nielsen 1993). At first view this looks like a list that has to be simply worked off, which is incorect. Nielson's model contains circles and feedback paths, also, but it mentions only the components of usability engineering. The context is assumed to be known. The correct application does not result in a kind of lightweight model. However, it is not so easy to find a suitable model.

Another approach that is often discussed in that context is the *Star Life Cycle* from Hartson and Hix, see Figure 3 (Preece 1994). In contradiction to Nielsen's model the Star Life Cycle offers structural information associated with the components. This model is also not really helpful for developers: It shows only the steps that are known from the development models around the term »Evaluations«. This explains not much about how evaluations could be conducted easily. Hartson and Hix do not consider the Star Life Cycle as a model for usability evaluations or for usability engineering. The focus is on an alternative to the waterfall or spiral model for development purposes since those models even do not mention evaluations or the term »usability«. So the Star Life Cycle is no solution for the problem mentioned above.

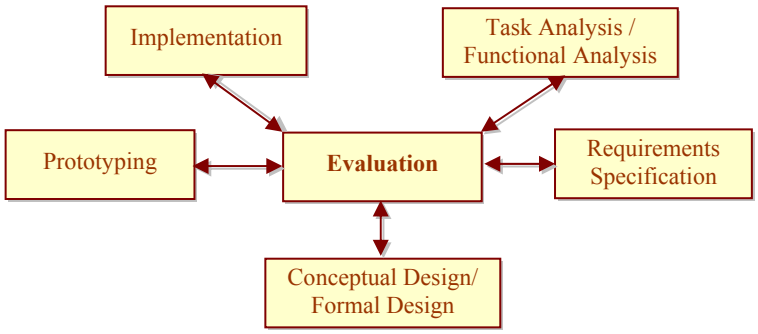


Figure 3: *Star Life Cycle [Pre 94]*

Further approaches for modelling usability engineering, usability evaluations or usability testing are shown by Rubin (Rubin 1994), Constantine and Lockwood (Constantine and Lockwood 1999), Dumas and Redish (Dumas and Redish 1999) and others. The approaches that show usability testing only are not discussed separately since this is only one aspect we want to cover

.

2 Terminology

Before our approach is presented some terms should be explained. The first thing that is interesting is the *weight* of a model or a process. Some processes like *eXtreme Programming* (XP) or Hacking are considered as lightweight models. The counterpart to these are models like the waterfall and the spiral model, as well as the Rational Unified Rrocess (RUP). The difference is the flexibility the process allows for its users (Eckstein 2000). Applying the waterfall model means to go through the first step to the last one only once. Errors are hard to correct since it is allowed to go back only one step if necessary. Practice shows that it is often helpful to go back further. A second criterion for characterizing process models is the quota of work which has to be done. According to these considerations, Mayhew's usability lifecycle and Nielson's usability engineering model are heavy weighted models. Our present model offers a lot of liberties (see section *The Eight Phase Pattern*). It contains structural information but does not hinder going back if necessary. This enables software developers to decide nearly free how to combine steps. Since nearly no work has to be done to fulfill the models requirements we tend to classify it as a lightweight model.

The next topics are the usability terms: *Usability engineering* includes every activity that is related to any

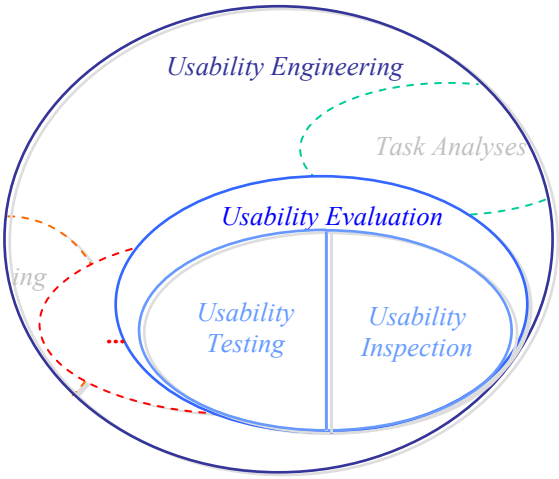


Figure 4: *Layers of Usability Terms*

stage of the product development to improve the usability properties. The term stands for a kind of a superset of work and measures around this issue. For the purpose of evaluating results such models are too broad. Sub-terms are *Usability Evaluation*, *Usability Inspection* and *Usability Testing*.

Usability evaluation is one component of usability engineering that is realized by usability inspections and usability testing. Further components of usability engineering are *task analyses*, *requirements engineering* and others that are not considered here. Usability Testing means to test artifacts with actual users. The artifacts can be software prototypes but also early paper and pencil mock-ups. A usability inspection is done by comparing artifacts against requirements, checklists or giving them to an expert review. See Figure 4 for an all-embracing view.

The last term that should be explained is »e*Xtreme*«. The term *eXtreme Programming* (XP) is created by Beck in 1998 (Beck 1998). Beck became engaged in the *Payroll Project* at Chrysler, a project entailing substitutions of 15 different payroll systems that were running concurrently at that time. The project suffered classical software engineering problems. XP overwrites with some established rules. The following shown approach is not as radical for usability engineering as Beck's approach was for software engineering. The term extreme is adapted to *extreme evaluations* since our approach wants to enable more agile workflows as well.

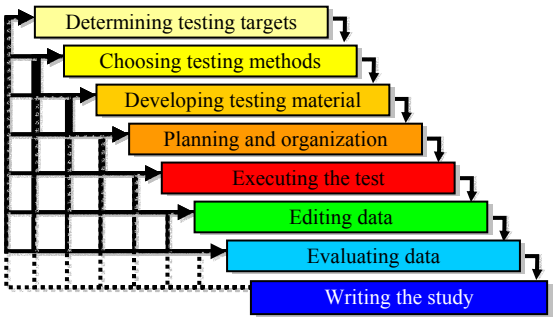


Figure 5: *Eight Phase Pattern*

3 The Eight Phase Pattern

To be understood easily by developers, the approach is designed as a model that is known in that domain. Although the former waterfall model has some disadvantages it is really easy to understand and well known by software developers. For usability evaluations it is no problem to go back only one phase (Gellner 2000). Our eight phased model is shown in Figure 5. Since it is part of a pattern language for extreme usability evaluations it is called *Eight Phase Pattern*. For further information see (Gellner 2003a).

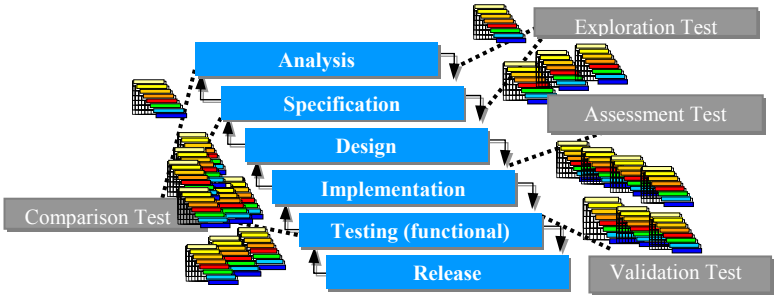


Figure 6: *Integrating the Eight Phase Pattern*

Beneath the information, which sub-tasks have to be fulfilled, also structural information is given. It is possible to go back as far as necessary at every time. Figure 6 shows the relations between the Eight Phase Pattern and the waterfall based development, Figure 7 shows how to integrate the Eight Phase Pattern in spiral development processes. In the same way the integration of tests can take place by other development approaches (incremental model, object oriented model etc.).

In comparision to some other models, the Eight Phase Pattern contains no component that limits the usage in external projects. The scenario of external consultants or usability experts is covered as well as in-house evaluation.

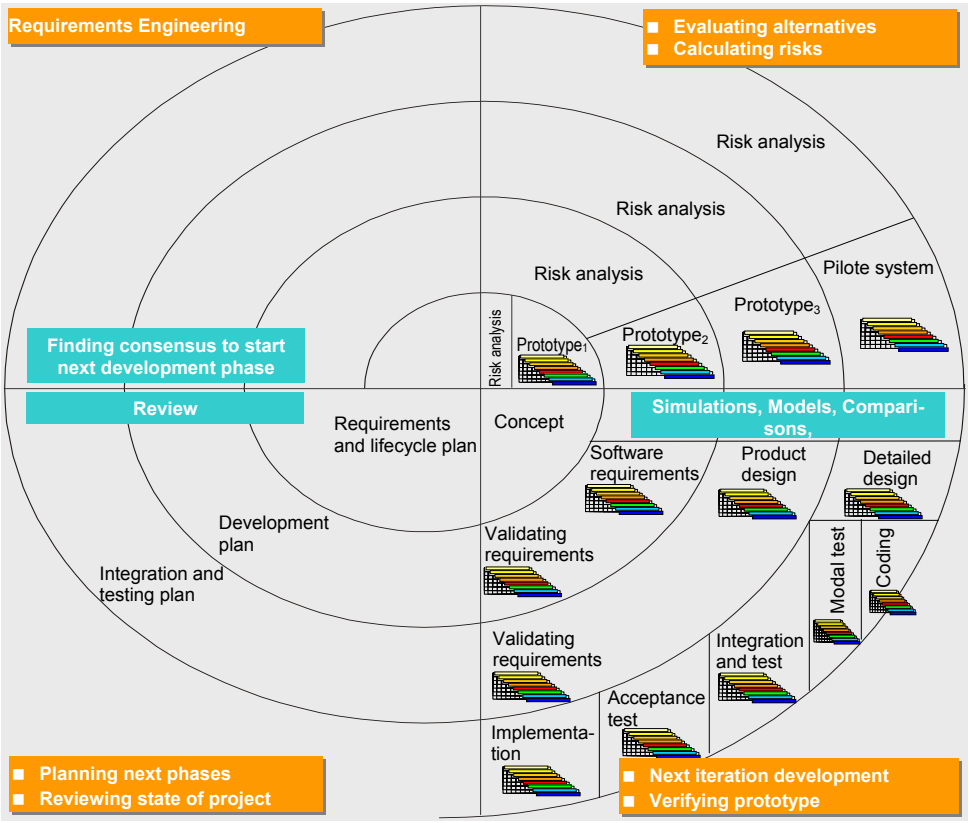


Figure 7: *Integrating the Eight Phase Model in the Spiral Model*

.

4 Applications

The eight Phase Pattern can be seen as a process description and as a development concept (take it and cover every phase with tools). Nielsen's model (see Figure 2) for example contains several points (participatory design, prototyping and others) that are hard to map into software tools, whereas every point in the Eight Phase Pattern can be seen as a component in a workflow management tool (represented in the most primitive case at least as wizards). In comparison to software development we are nearly speechless if tools or methods have to be judged. In software development we can easily assign a tool to a certain phase into the waterfall view (IDE → implementation, UML-Painter → analyzing, specification, CASE Tool → spanning phases). This works even if the waterfall is not the underlying model.

The Eight Phase Pattern enables such communication for usability evaluations (see Figure 8). It is also possible to analyze the costs that were caused by the work for each phase, see Figure 9. This data can be used to compare with other labs to increase efficiency. Further more it allows recognizing areas or phases not covered by tools.

Our intention to find approaches for software support led to a set of tools. Until now there are approaches and solutions for the phases 2 to 7. Tools that support phase embracing evaluations are considered as *Computer Aided Usability Evaluation Tools* (*CAUE*). This term again is created similar to the term *Computer Aided Software Engineering* (*CASE*).[1]

Our most powerful approach *ObSys* combines the methods

- *Event Logging* (using predefined short cuts manually to save observations)
- *Event Recording* (capturing automatically the message queue of an operating system)[2]

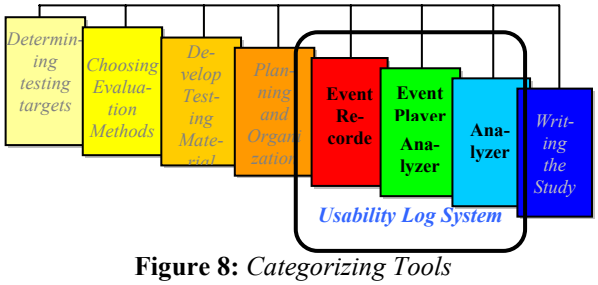


Figure 8: *Categorizing Tools*

[1] In literature there is also mentioned the term *Computer Aided Usability Engineering (CAUsE)*. At the moment we see no basis for such a comprehensive demand.

[2] At the moment determined to Microsoft Systems; beneath the *ObSyS*-Tool we have a VNC-based solution that works on all platforms to be connected to the internet

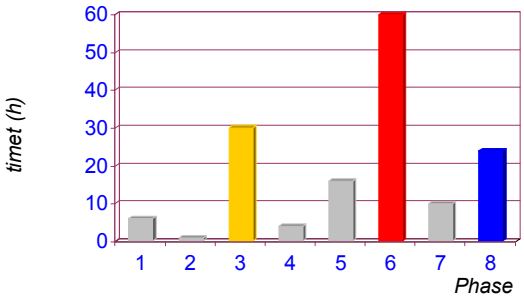


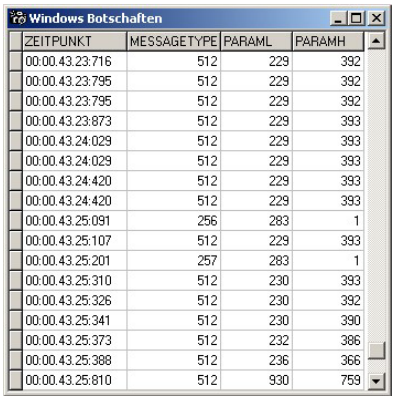
Figure 9: *Rating Performance per Phase*

- *Video Recording*
- *Screen Capturing*

Event recording has especially a high potential to automate product or prototype-based usability evaluations. However, as known to usability experts, it is too late to start with evaluations when there are pre-releases. For that reasons there are more methods we recommend and try to combine. The events are saved in databases and can be processed with SQL to find striking datasets (see Figure 10).

More concisely is our visualization called MouseMap. This multidimensional representation allows watching events graphically. The direction the mouse moves is visualized by color gradients, and clicks are pointed as round dots and the speed is indicated by the thickness of the lines (Gellner and Forbrig 2003, Gellner 2003b).

MouseMaps summarize in an image what as a video sequence takes some time, see Figure 11. On the other hand selecting too many events for a MouseMap is too complex to be analyzed qualitatively.



ZEITPUNKT	MESSAGETYPE	PARAML	PARAMH
00:00.43.23:716	512	229	392
00:00.43.23:795	512	229	392
00:00.43.23:795	512	229	392
00:00.43.23:873	512	229	393
00:00.43.24:029	512	229	393
00:00.43.24:029	512	229	393
00:00.43.24:420	512	229	393
00:00.43.24:420	512	229	393
00:00.43.25:091	256	283	1
00:00.43.25:107	512	229	393
00:00.43.25:201	257	283	1
00:00.43.25:310	512	230	393
00:00.43.25:326	512	230	392
00:00.43.25:341	512	230	390
00:00.43.25:373	512	232	386
00:00.43.25:388	512	236	366
00:00.43.25:810	512	930	759

Figure 10: *List with recorded events*

At the moment, error detection is not automated. An easy way to find error »candidates« is given with the time series of the events. If the time differences are visualized peaks can appear. Assumed that a user acts with a personal workflow, a peak can indicate a tool based problem (but also someone asking something else). In a usability evaluation session all interruptions (reading scenarios, getting explanations etc.) are documented. So it is easy to distinguish between deflections and other sources for peaks.

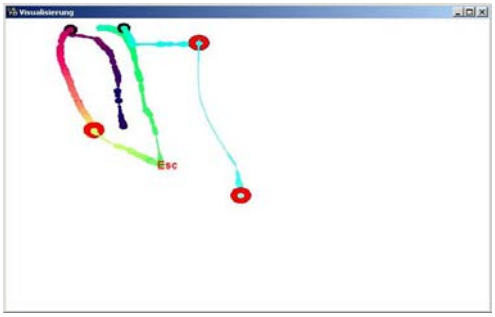


Figure 11: *MouseMap after Observation*

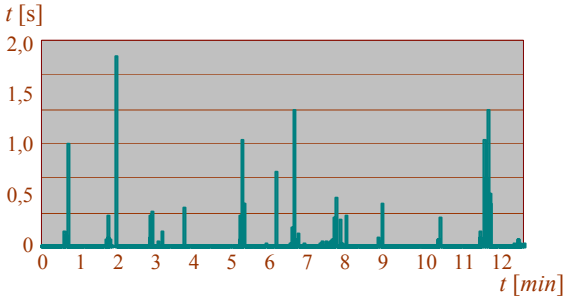


Figure 12: *Differences between events*

The shown scenario in Figure 12 was observed by editing paragraphs in WinWord with built-in problems. Around half of the peaks were caused by reading the scenario items. The other peaks indicate problems. Analyzing six (± 3) exact positions in a 10 minute scenario is much easier than watching the scenario three times on video to find fairly the same positions

.

5 Pattern Language

The work with our tools inspired us in the identification of different patterns in usability evaluation. Besides our first pattern, the eight phase pattern, we identified thirty-two other ones. Two of them are process patterns not related to a single one of our eight phase model. All other patterns can be related to a special phase of our model. They are documented in a way which is described by an EBNF grammar. This will be published in the forthcoming

PhD-thesis of Michael Gellner. Let us have a look at a very short description of the patterns.

General process patterns

- *Eight Phase Pattern*: Conducting a usability evaluation means to realize a process. Our process view is explicitly formulated in the *Eight Phase Pattern*.
- *Epoch*: The *Epoch Pattern* brings help to categorize the knowledge background of users. Similar to the historical term an epoch is not only characterized by temporal data (see the definition for *classic*).
- *Evolution*: *Evolution* describes the fact that each development in GUIs could transform the user. As a result users can change their *Epoch*, the state of *Testing Persons* can change.

Determining testing targets

- *Target*: In fact, usability is a conglomerate of different attributes and not a single feature. The *Target Pattern* suggests a set of common subjects and calls on the evaluators (possibly developers) to decide for the needed ones.
- *Effectiveness and Efficiency*: Ergonomics will never lead to exact results. This is characteristic for social sciences and psychology. On the one hand, it is important to accept that total effectiveness is not realisable. On the other hand, under-running a certain degree of efficiency can lead to the total loss of all endeavours.
- *Kind of Test*: Rubin introduced the categories *kinds of tests* (Rubin 1994). Different testing methods are attached to these categories. This is a helpful approach because Rubin associated each category with special features.
- *Criterion*: The *Criterion Patterns* offers a set of criterions that are suited well to parameterise the situation in which an evaluation will be conducted. This information is used in the next phase to determine proper evaluation methods (there are more than 50 evaluation methods known).

Choosing testing methods

- *Requirements Method Indexing (RMI)*: The number of different methods complicates the selection of a suitable one, especially for developers. For that reason *RMI* is introduced, i.e. an algorithm that advices matching methods (based on the information of the *Criterion Pattern*).
- *Testing Person*: A lot of evaluation methods can be used only if there are matching testing persons. The *Testing Person* Pattern proposes an overview about which information is necessary for these decisions.
- *Recording Technology*: Choosing a proper recording technology determines highly what analysis methods can be applied (manually, semi automated, automated, quantitatively etc.). So it is important to decide early what kind of analyses will be done and to select the matching technology.
- *Duration*: Another factor for the selection of a fitting *Recording Technology* is the duration of the signals that should be recorded . The *Duration Pattern* gives an overview about signals that could be recorded and their duration. This pattern is based on information that is given in (Hilbert and Redmiles 1999).

Develop Testing Material

- *Fingerprint*: For some evaluation methods it is helpful to imagine a priori how the data could »look« and what errors are possible. Comparing those »patterns« can lighten the analyses significantly.
- *Template Collection*: The *Template Collection Pattern* gives one idea evaluation materials for developers could be offered.
- *Encyclopaedia*: Building up a small *Encyclopaedia* is another approach offering materials to developers.
- *Maieutics*: The *Maieutics Pattern* shows the principle that underlies software wizards.
- *Wizard*: Another alternative to the *Template Collection Pattern*.

Planning and Organization

- *Set*: Often usability evaluations are similar to movie sets – a lot of factors have to match exactly at one time. The *Set Pattern* discusses the sensible points of such a project.
- *Mind Mapping*: The *Mind Mapping Pattern* shows a method to ship the risks that discussed in the *Set Pattern*.

Executing the evaluation

- *Deleting Testing Persons*: The *Deleting Testing Persons Pattern* shows solutions to avoid unwanted *Epoch* and *Evolution* effects.
- *Atmosphere*: This pattern points to different facets of distortion in tests and offers solutions.
- *Streamline*: Streamline sets up on Spencers suggestions to avoid breaks in inspections.
- *Testing Subject*: *Testing Subject* stresses testing persons are not the testing subjects.

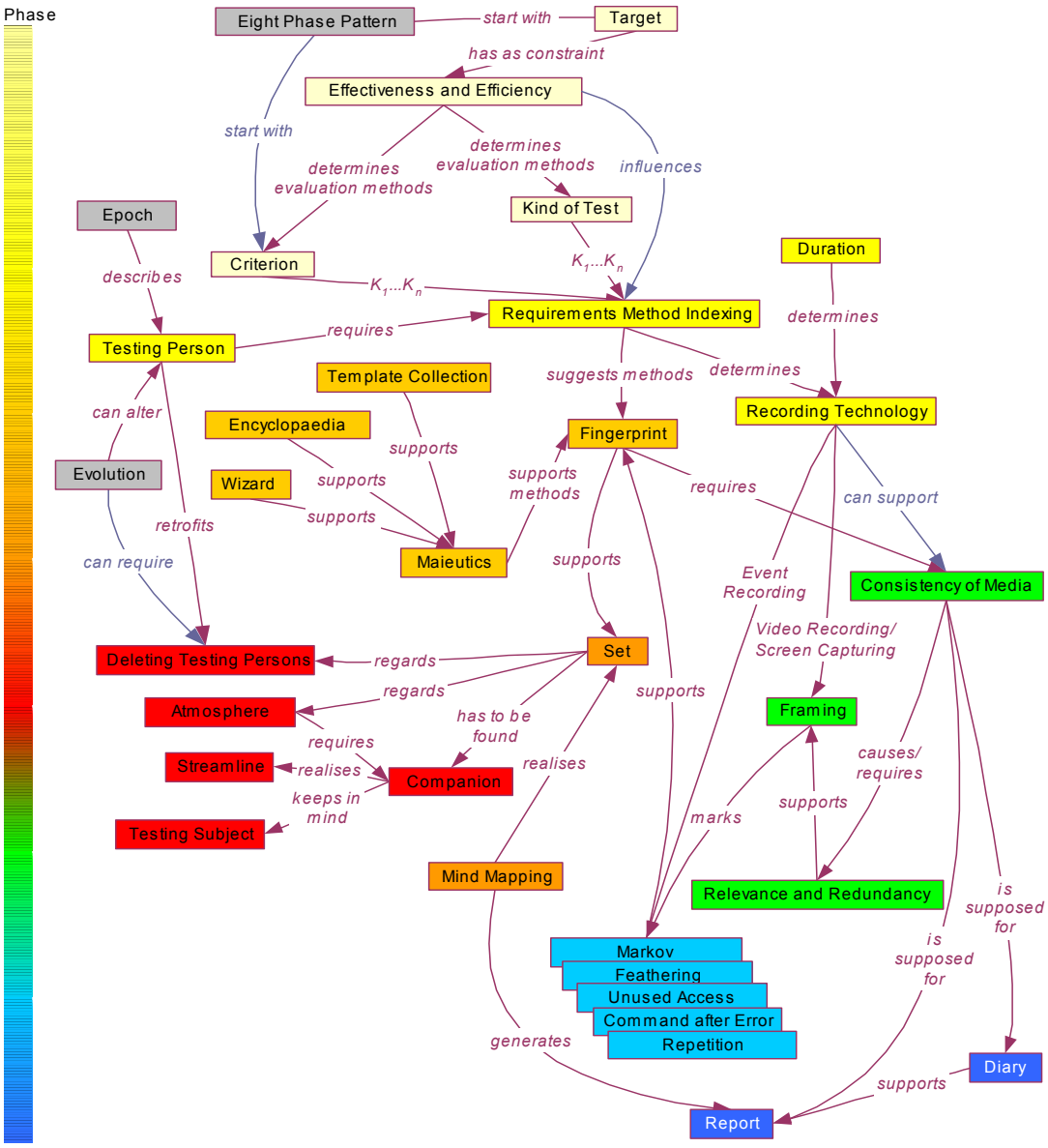


Figure 13: *Usability evaluation patterns and their relations*

- *Companion*: In this pattern, advice on how to behave duringtests with testing persons are given.

Editing data

- *Consistency of Media*: For a high degree of automation and efficiency the *Consistency of Media Pattern* is highly important.
- *Framing*: In this pattern analyses, ways of using and reusing data are described.
- *Relevance and Redundancy*: *Relevance and Redundancy* explains the ratio of unneeded parts.

Evaluating data

Hint: These patterns are recognition patterns. These patterns are different to the other *repeatable solutions to problems in similar contexts* (due to Alexander, GoF etc.).

- *Markov*: Markov Chains are an interesting approach to investigate actions that are performed about different applications and windows.

- *Feathering*: Our first sessions showed something like body language by using input devices. If this idea can be verified a powerful instrument for model independent automation of usability evaluations was given.
- *Unused Access*: Opening a GUI element without using it, indicates a problem.
- *Command after Error*: The command an user executes after an error can tell what the user wanted originally. Error and next trial are important components to interpret a testing persons mental model.
- *Repetition*: Another important error indicator are commands the user executes more than one time. This information can become even more expressive if special items are used (e.g. the ESC key).

Writing the study

- *Diary*: *Diary* shows a self reporting protocol mechanism.
- *Report*: This pattern shows how information that is managed with a *Mind Mapping* tool can be used to generate reports.

Figure 13 shows a graph that gives further information about how the patterns are related. Due to (Borchers 2001) we call this as a pattern language. These patterns continue the idea of patterns for interaction design to patterns for usability evaluations. In this way, the whole collection is a *usability evaluation pattern language*.

6 Further Work

For finding appropriate tools and patterns the Eight Phase Pattern was a helpful and effective approach. Our investigations resulted in a pattern language for usability evaluation. Further tools can be deduced and have to be realized. The most important step is the implementation of the observed error patterns for automating the detection. As a result, the evaluator would get video snippets with relevant scenes and MouseMaps around the error situation. Based on that material the evaluator must decide what steps have to be taken next. A high amount of the time consuming finding of the relevant positions of videos were eliminated. With the shown approaches even software developers would be able to conduct usability tests and to evaluate the results. Other focuses are on the first phases. In the next month a tool for selecting well suited methods based on the algorithm in (Gellner 2002; Gellner 2003a) will be completed.

With the MouseMap visualization we found something like a body language (another kind of pattern) users show by working with mice and keyboards. We assume that there are many aspects beyond simple causalities like Fitt's Law or the Steering Rule that can be investigated in greater detail.

References

Beck, K. (1999), Extreme programming explained: embrace change. Addison-Wesley.

Borchers, J., (2001) A Pattern Approach to Interaction Design. John Wiley & Sons, Chichester, England.

Constantine, L. L. and Lockwood L. A. D (1999), Software for use, a practical guide to the models and methods of usage-centered design. Addison Wesley Longman, Inc., Reading, Massachusetts.

Dumas, J. S. and Redish, J. C. (1999), *A Practical Guide to Usability Testing*. Revised Edition. Intellect Books Limited, Exeter, England.

Eckstein, J. (2002), XP – eXtreme Programming: Ein leichtgewichtiger Software-Entwicklungsprozess. In: basicpro, Vol. 35, 3, pp. 6-11.

Gellner, M. (2000), Modellierung des Usability Testing Prozesses im Hinblick auf den Entwurf eines Computer Aided Usability Engineering (CAUE) Systems. In: *Rostocker Informatik-Berichte*, Vol. 24, pp. 5-21. Rostock, 2000.

Gellner, M. (2002), A Pattern Based Procedure for an Automated Finding of the Right Testing Methods in Usability Evaluations. In: Forbrig, P., Limbourg, Q., Urban, B. und Vanderdonckt, J., Bricks & Blocks: *Towards Effective User Interface Patterns and Components*, Proceedings of the 9th International Workshop Design, Rostock, 2002, pp. 423-427.

Gellner, M. (2003a), Automated Determination of Patterns for Usability Evaluations. In: Hruby, P. und Sørensen, K. E. [Ed.]: *Proceedings of the VikingPLoP Conference 2002*, Micorsoft Business Solutions, ApS, 2003, pp. 65-80.

Gellner, M. (2003b), Mousemaps – ein Ansatz für eine Technik zur Visualisierung der Nutzung von Software und zur Automation der Entdeckung von Bedienungsfehlern. (submitted and accepted at *Mensch & Computer 2003*)

Gellner, M.; Forbrig, P. (2003c), *ObSys – a Tool for Visualizing Usability Evaluation Patterns with Mousemaps*. *HCI International 2003, Heraklion, Greece*

Gellner, M.; Forbrig, P. (2003d), *Extreme Evaluations – Lightweight Evaluations for Soft-ware Developers, Workshop on SE and HCI, INTERACT 2003, Zürich, Switzerland*

Hilbert, D.M. und Redmiles D.F., (1999) Extracting Usability information from user interface events. Technical Report UCI-ICS-99-40, Department of Information and Computer Science, University of California, Irvine.

Mayhew, D. J. (1999), The Usability Engineering Lifecycle. Morgan Kaufmann Publishers, Inc., Kalifornien, San Francisco.

Nielsen, J. (1993), *Usability Engineering*. AP Proffessional, New Jersey.

Preece, J., (1994), Human-Computer-Interaction. Addison-Wesley, Harlow,.

Rubin, J., (1994) Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests, John Wiley & Sons, Inc..

Spencer, R., (2000) The Streamlined Cognitive Walkthrough Method, Working Around Social Constraints Encountered in a Software Development Company. In: Turner, T., Szwillus, G., Czerwinski, M. und Paternò, F. [Ed.], *CHI 2000 – The Future is here*, Conference Proceedings, The Hague, Netherlands, 2000, pp. 353-359.