Application-Oriented Usage Quality

- The Tools and Materials Approach -

Carola Lilienthal, Heinz Züllighoven
Department of Computer Science
University of Hamburg
Vogt-Kölln-Str. 30
D-22527 Hamburg
+49 40 54 94-2307/2414
{lilienth, zuelligh}@informatik.uni-hamburg.de

One of the main goals of software development is to provide the user with useful and usable software. Many different techniques are employed to ensure that the analysis, design and construction of software leads to a product with quality in usage. However once a system is shipped and installed at the user's site - the gap between the developer's idea of usage quality and the user's needs becomes obvious. From our point of view usage quality can only be achieved by placing a strong emphasis on the application area. The software developer on the one hand needs to understand the user's tasks. The software product on the other should represent the main concepts of the application area. By taking these two guidelines into consideration, software with application[1]-oriented usage quality can be produced.

In this article we present an approach that supports software development with the goal of application-oriented usage quality. Based on object-oriented techniques different concepts such as a leitmotif, design metaphors, documents and prototyping are combined to form a coherent methodology - the Tools and Materials Approach.

[1] We use the term 'application-oriented' to highlight our focus on the application area. 'Application-oriented' should not be misunderstood as oriented towards software as a product.

APPLICATION-ORIENTED USAGE QUALITY

In the early days of software development mainframe-based systems automated individual work routines, such as money withdrawal or money transfer. With the advent of graphic workstations and PCs, a shift from work routines to the support for everyday work occurred. The electronic desktop became the predominant design metaphor and users could find a set of useful objects such as documents and folders on their desktop. Some more or less generic software tools such as word processors and graphical editors also became available to cope with the various tasks. These window-based systems have been one of the major steps towards usage quality in software systems.

This increase in usage quality and faster computer hardware enabled software developers to tackle more complex tasks in different application domains. At the same time it became obvious that merely capturing, changing and deleting data was not enough. Software products had to be turned into specific components that matched concepts and entities of everyday work. Most software development methods in the eighties were therefore directed at designing and implementing application-oriented interactive software.

We have chosen an evolutionary, object-oriented approach, called the Tools and Materials Approach (cf. [1, 2, 3]) that aims to provide

software components with application-oriented usage quality. The Tools and Materials Approach focuses on a close relationship between the tasks and concepts of the application domain and the components of a software system. This relationship allows the users to recognize their specific means and objects of work and enables them to fulfill their tasks by individually organizing work as the situation demands it.

LEITMOTIF AND DESIGN METAPHORS

From our point of view interactive application-oriented software should bring together functionality and appropriate handling and representation. This means that the developer has to cope with a design task in its original sense, by relating form to function. Our guideline for this design task is what we call a leitmotif.

- A *leitmotif* makes the way of looking at software explicit. It helps developers and users to understand and design a software system on a general level.

A leitmotif can be made tangible with the help of a set of design metaphors which solidify the general guidelines in a pictorial way.

- A *design metaphor* describes a component of a software system by means of a phenomenon, or better, an artifact from the users' 'every-day' world. With the help of these metaphors it is possible for software developers and users to relate design and implementation components to familiar implements and terms, so that all parties share a common background and have a basis for communication.

Using metaphors for software design is not a new idea (cf. [5]). There has been long discussion in Scandinavia about using metaphors (cf. [6]). It should be noted, however, that we are offering a comprehensive set of design metaphors integrated within a uniform leitmotif (see below).

Leitmotif and Design Metaphors for the Individual Workplace

As our predominant leitmotif we have chosen the *workplace for expert users* where a certain degree of individual responsibility is called for. It suits the majority of our industrial projects in the area of office work in particular service and financial sectors. Design metaphors which have proven useful for application software in office-like environments are materials, tools, automatons and work environment (see Fig. 1, cf.[2, 4]).

These central metaphors stem from the context of human work. It is a common observation supported by work psychology [7] or epistemology that in many work situations people make intuitive distinctions between those objects which are *worked on*, i.e., materials, and those which are the *means of work*, i.e., tools (see Fig. 1).

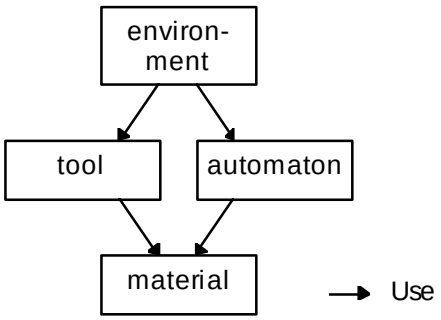

Fig. 1: The metaphors

- A *tool* supports recurring work procedures or activities. It is useful for various tasks and aims. A tool is always handled by its user who decides when to take it up and what to do with it.
- *Materials* are the objects of work which finally become the result or outcome of tasks. They incorporate 'pure' application domain functionality. A material is worked on by tools according to professional needs. A material should be characterized by its potential behavior not its internal structure.

Not everything in an office environment is a tool or a material. There are certain cumbersome work routines a user wishes to delegate to a machine. To represent these the metaphor *automaton* (see

Fig. 1) was added. An automaton is started by a user and is active over a long period in the background. Once set and started it produces a predefined result without user interaction.

Tools, materials and automatons need to be presented and accessible to the user. There is always a place where work is done and where tools, materials and automatons can be found. The metaphor *work environment* (see Fig. 1) corresponds to this. Users should have their own work environment with privacy and its own arrangements of things.

On the basis of the metaphors tool, material, automaton and environment we have designed interactive systems for individual workplaces. These workplaces were equipped with single user applications with a good level of usage quality. The next step is to support cooperative work.

Design Metaphors for Cooperative Work

When we observe expert users in cooperative work situations (cf. [12, 13]), we can identify several characteristics of cooperative work in an office environment:

- Cooperation takes place by exchanging materials or placing materials in specific locations. Frequently, materials are grouped into files, folders or other kinds of containers.
- The actual sequence of activities performed with the materials is not fixed. It is possible to describe the routine case, but many actual work situations don't follow this routine.
- The control during subsequent activities within a cooperative work situation of this type is always with the person having access to the material.[2]

These observations led us to the conclusion that cooperative work in most cases can only be supported by a repertoire of resources to use in self-organized work. Complementary to the design metaphors tool, material, automaton and environment we have therefore added the metaphors 'process pattern', 'document folder':

- A *process pattern* is a shared material to reify the process of dealing with a cooperative task for all parties involved. It consists of a list of names of the work steps or sub-tasks that have to be done and of the responsible people or roles. The process pattern is attached to a *document folder* which contains those materials that are the objects of work.
- A process pattern in general represents the standard or routine case of dealing with a task. This routine case has to be established through a common background of cooperation and experience. *Adaptation of process pattern* with a special tool allows changing the routine according to specific needs of the situation.

Process patterns and document folders are exchanged between the individual work environments with the help of a *transport automaton*. Whenever a user has finished his or her work step or sub-task, a check mark is added to the task list of the process pattern. The user can then hand the document folder over to an outbox which passes it on to the transport automaton.

These concepts for cooperative work in an office-like environment not only match the design metaphors of the Tools and Material Approach. The 'workplace leitmotif' is also maintained, at least for many work situations with a high degree of established cooperative processes.

With the 'workplace leitmotif' and the design metaphors a clear decision was made towards application-oriented work support in contrast to work automation. These concepts play a crucial role in achieving usage quality because they help us to focus on the application domain. The key to usage quality however, is the understanding of the user's tasks. For this purpose a set of document types and prototyping will be presented in the following section.

[2] This is true as long as the users are cooperating only by exchanging real paper and folders. 'Workflow systems' which involve a fixed sequence of tasks through which the users are forced to go, are not our aim, when we try to support cooperative work.

DOCUMENT TYPES AND PROTOTYPES

The need for application-oriented document types as the basis for understanding and representing the concepts and tasks of the application domain should be fairly obvious (e.g. [8, 9]). The predominant prerequisite is that the document types be based on the professional language used in the application domain. In most cases they are written in prose. With the our leitmotif and the design metaphors in mind the developers thus are guided to see the development process from an application-oriented perspective.

Document Types for the Individual Workplace

We have successfully used a set of application-oriented document types that are well-known under various names in the literature (cf. [1]).

- *Scenarios* (see Fig. 2) describing the current work situation, the everyday tasks and the objects and means of work. Scenarios are written by developers based on interviews with users and the various other groups involved.
- *Glossaries* defining and reconstructing the terminology of the professional language in the application domain. The entries in a glossary offer the first hints for relevant materials used in the application domain.
- *System visions* (see Fig. 3) anticipating future work situations. They are frequently supported by prototypes. They describe the developers ideas how tools, materials, automatons, process pattern and folders of the future system will be arranged in the work environment and could be handled by the users.
- *Prototypes* are tangible objects for anticipating future situations both from use-related and technical perspectives. They are built out of components that match the design metaphors allowing the users and developers to discuss delimitable parts of the system.

> An advisor fetches a *customer advice file*, looks for the required *product* in the index and opens the *file* at the desired spot. In addition to the customer advice file, there is a *form file* in which *standard forms* (e.g., *contracts* with third parties) are kept, and a specimen file in which completion guides and *code sheets* are deposited.

Fig. 2: A scenario

> An advisor opens a *customer advice file* with a double click, selects all *products* from a visible table of contents, and then selects the desired *variant* from a second table - thus displaying the corresponding sales help facility.

Fig. 3: A system vision

The crucial point in using these documents is the transition from the current work situation to the future system. While documents that describe current work situations (e.g. scenarios, glossary entries) can usually be discussed and evaluated by the application domain experts without major difficulties, discussing and evaluating the design of the future system is different. People tend to find it hard to anticipate future ways of coping with tasks or handling a system. For this reason and not surprisingly, prototypes play a central role in our approach (cf. [10, 11]). It is however still important that the emerging vision of the future system is documented beyond the actual prototypes. A prototype does not show the intended use context, explain design decisions or outline the anticipated handling of tasks. Therefore system visions describing the different aspects of the future system are provided.

Document Types for Cooperative Work

When we analyze cooperative work it becomes necessary to extend the former repertoire of document types. Scenarios and glossaries describe the individual activities and tasks within cooperation, but the overall perspective is missing. To bridge this gap, we use cooperation pictures, based on pictograms, to represent cooperative work on different abstraction levels. These pictures portray the participating workplaces or roles and the cooperation between them by using every-day pictograms or graphics linked by arrows. Used with sequence numbers actual work steps of a cooperative tasks can be

shown (s. Fig. 4). If several related tasks are mapped into one picture and numbers are omitted, the general structure of cooperation between different parties can be identified and nodes of high communication and cooperation 'traffic' become apparent.

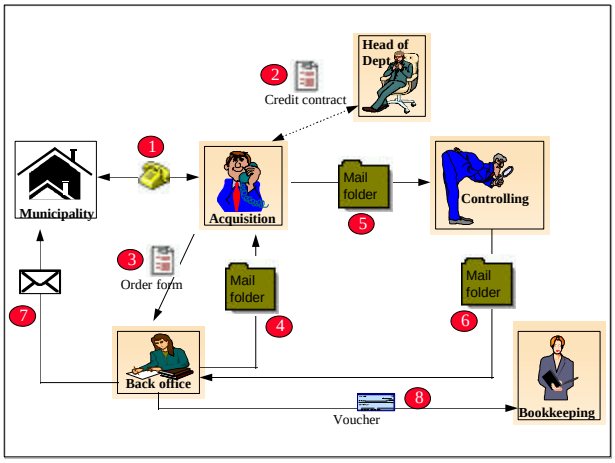

Fig. 4: Cooperation Picture for 'loan approval'

We have used cooperation pictures in workshops, firstly to understand cooperation in an organization and, secondly to discuss the possibilities of reorganizing cooperative work.

GETTING THE TOOLS AND MATERIALS APPROACH TO WORK

Leitmotif, design metaphors, document types and prototyping are different tools to support software developers in understanding the users' tasks and the concepts of the application domain. To initiate and maintain the learning and communication process between developers and expert users, we are using an evolutionary concept of fast design and feedback cycles (see Fig. 5). With these evolutionary cycles we explicitly turn away from the traditional life cycle strategies.

The evolutionary cycles is based on documents combined with prototypes (s. Fig. 5). Instead of following the pre-defined work steps of a waterfall model with its sequence of milestone documents, we have identified three complementary and highly intertwined activities that developers carry out during the software development process: analyzing, designing and evaluating. By *analyzing* the users' tasks the developers gain an initial understanding of the concepts of the application area. This understanding is recorded in documents and prototypes (based on object-oriented classes and frameworks) within the activity of *designing*. To make sure that the understanding of these tasks and concepts is shared by the domain experts the third activity of *evaluating* the documents and prototypes is performed.

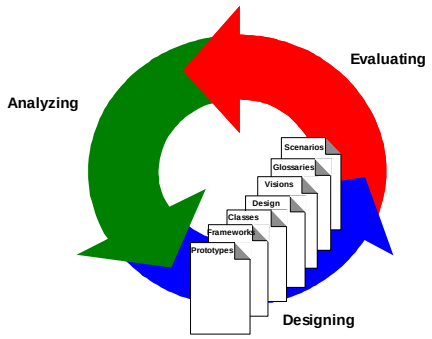

Fig. 5: Evolutionary process with feedback cycles

For example, developers interview users at their workplaces (analyzing) and prepare scenarios (analyzing/designing) which are discussed with the interviewees and other users (evaluating). For the development of the new system on the basis of these scenarios, system visions are written (analyzing/designing) by developers and respective prototypes are realized (designing). These prototypes are afterwards evaluated in workshops or 'hands-on sessions' by the users.

It is important to note that during the feedback cycles any problems that occur should direct us to the appropriate documents that need modification (see Fig. 5). There is no predefined sequence of working with documents; in principle, all are available at any point. If some misunderstanding surfaces while evaluating a prototype the developers may decide that they need to write or rewrite more scenarios, glossaries and system visions. Then they should be free to stop working on prototypes and start a new analyzing-designing-evaluating cycle with scenarios.

CONCLUSION

In this article we have suggested that usage quality is a major goal in software development. To enable the design of useful and usable software, a leitmotif, design metaphors and document types have been introduced. These concepts play such an important role in achieving usage quality because they help us to focus on the application domain.

With the 'workplace leitmotif' a clear decision was made towards application-oriented work support in contrast to work automation. The design metaphors *tool*, *material*, *automaton*, *environment*, *process pattern*, *document folder* and *transport automaton* solidify the leitmotif and equip us with implements to furnish electronic workplaces.

By employing a leitmotif and design metaphors, application-oriented usage quality can be maintained. The prerequisite for this usage quality however, is the understanding of the users' tasks. For this purpose scenarios, glossaries, system visions, prototyping and cooperation pictures have been presented.

The combination of a leitmotif with design metaphors, document types, object-oriented construction techniques and an evolutionary development process based on prototyping provides us with an excellent basis for software development.

ACKNOWLEDGMENTS

This paper summarizes the essential ideas of [3, 4, 12, 13]. We would like to thank Guido Gryczan, Anita Krabbel, Ingrid Wetzel and Martina Wulf for their advice and help.

REFERENCES

[1] Bürkle, U., Gryczan, G., Züllighoven, H. (1995) *Object Oriented System Development in a Banking Project: Methodology, Experience, and Conclusions*. Proceedings of HCI, Vol. 10, pp. 293-336.

[2] Riehle, D., Züllighoven, H. (1995) A Pattern *Language for Tool Construction and Integration Based on the Tools & Material Metaphor*. In Coplien, J. O., Schmidt, D. C. Pattern Languages of Program Design. Addison-Wesley, Reading, pp. 9-42.

[3] Lilienthal, C., Züllighoven, H. (1996) *Techniques and Tools for Continuous User Participation*. In: Blomberg, J., Kensing, F., Dykstra-Erickson, E. (Eds.): PDC'96 Proceedings of the Participatory Design Conference, Cambridge, Massachusetts, November 1996, pp. 153-159.

[4] Bäumer, D., Gryczan, G., Knoll, R., Züllighoven, H. (1996) *Large Scale Object-oriented Software Development in a Banking Environment*. In: Cointe, P. (Ed.) ECOOP '96 - Object-Oriented Programming, Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 73-90.

[5] Carroll, J. M., Mack, R. L., Kellogg, W. A. (1988) *Interface Metaphors and User Interface Design.* In Helander, M. (Ed.) Handbook of Human-Computer Interaction, pp. 283–307.

[6] Ehn, P. (1988) *Work–oriented Design of Computer Artifacts*. Almquist and Wiksell International, Stockholm.

[7] Volpert, W. (1991) *Work Design for Human Development.* In Floyd, C., Züllighoven, H., Budde, R., Keil-Slawik, R. (Eds.) Software Development and Reality Construction, Springer-Verlag, pp. 336-348.

[8] Carroll, J. M., Rosson, M. B. (1990) *Human Computer Interaction Scenarios as Design Representation.* Proceedings of the Hawaii International Conference on System Sciences, Los Alamitos *CA* IEEE Computer Society Press, pp. 555-561.

[9] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1992) *Object-oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley, Reading.

[10] Lichter, H., Schneider-Hufschmidt, M., Züllighoven, H. (1994) *Prototyping in Industrial*

Software Projects - Bridging the Gap Between Theory and Practice. In IEEE Transactions on Software Engineering 20, 11, pp. 825-832.

[11] Budde, R., Kautz, K., Kuhlenkamp, K., Züllighoven, H. (1992) *Prototyping*. Springer-Verlag, Berlin.

[12] Krabbel, A., Wetzel, I., Ratuski, S. (1996) *Participation of Heterogeneous User Groups: Providing an Integrated Hospital Information System*, In: Blomberg, J., Kensing, F., Dykstra-Erickson, E. (Eds.): PDC'96 Proceedings of the Participatory Design Conference, Cambridge, Massachusetts, November 1996, pp. 241-250.

[13] Wulf, M., Gryczan, G., Züllighoven, H. (1996) *Process Patterns - Supporting Cooperative Work in the Tools & Materials Approach*, In: Dahlbom, Bo, et al. (Eds.) Information systems Research seminar In Scandinavia: Proceedings of IRIS 19, Gothenburg: Studies in Informatics, Report 8, pp. 445-460.