Usability Engineering integrated with Requirements Engineering

B. Paech, K. Kohler
Fraunhofer Institute Experimental Software Engineering
Sauerwiesen 6, 67663 Kaiserslautern, Germany
{ paech,kohler}@iese.fhg.de

Abstract

In this paper we argue that the gap between Software Engineering and Human-Computer Interaction should be closed through the integration of usability engineering and requirements engineering. In particular, we present the elements such an integrated process has to cover.

1. Introduction

Requirements Engineering (*RE*) is the *systematic process of developing requirements through an iterative cooperative process of analyzing a problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained [1].* The essential tasks of RE during software engineering (*SWE*) are the elicitation and negotiation of requirements, their specification and validation as well as the their management over time.

The field of human-computer interaction (*HCI*) is concerned with the joined performance of tasks by humans and computational machines [2]. From the computer science perspective one essential contribution of HCI is the design, evaluation and implementation of interactive computing systems for human use [2]. The process that guides these tasks by specifying, measuring and improving the usability of a product is commonly called *usability engineering [3].*

The requirements process is typically characterized as an *analysis* process, where user needs and constraints must be elicited and analyzed. In contrast, most software engineers view HCI-activities as *design* (or test) activities.

In practice, this often leads to the misunderstanding that HCI-considerations can be brought in after the requirements are elicited and that requirements can be elicited without the consideration of the user interface.

In our view, a fundamental prerequisite for bridging the gap between SWE and HCI is that RE is understood as a *design* activity that includes the design of the user interface. During RE the *support for the user through the software system is designed*. There are many design decisions to be made such as the to-be-activities of the user tasks supported by the software, the system functions which perform parts of these activities, or the interaction between system and user when the system functions are executed.

To substantiate our claim we explain in the following the *requirements design decision types* (*REDT*) we have identified. This list covers typical functional requirements as well as typical HCI-issues such as screen structure.

Based on the REDT it is easy to argue that HCI and RE must be closely integrated to enable informed requirements decision making.

The paper is structured as follows: First we present the REDT, then we discuss the implications for an integrated RE and HCI process.

2.   Requirements Design Decisions

In the following we concentrate on functional requirements and on *user interface- and information-intensive systems* (*UIS* for short).

By an extensive study of the HCI- and SWE-literature we identified 16 requirements decisions to be made for UIS. These are shown in Figure 1 at the end of the paper.

- **(T1)   Decisions   about   the   user   tasks:**
  The decisions determine the user roles and the tasks of these roles to be supported by the system. Business processes determine these tasks.
  Example: A customer of a book-store has the tasks "search book" and "buy book".

- **(D1)   Decisions   about   the   as-is   activities:**
  The user tasks consist of several activities. As-is activities are the steps user currently perform as part of their work without the new system. Decisions must be made what the as-is-activities of a task are (as these are rarely explicit) and whether they are relevant for the system. These decisions shape the understanding of the purpose and the responsibilities of the new system.
  Example: The activities of the "buy book"-task in an conventional book store are "select book", "carry book to the till", "pay book".

- **(D2)   Decisions   about   the   to-be   activities:**
  It has to be decided how the as-is-activities will change as a consequence of the new system. As-is-activities always   have   potential   for   improvement.   New technologies like the internet or handheld can result in

radically new to-be activities. To-be-activities constitute the steps of the user tasks in the future.

> *Example*: The to-be activities for the "buy book" task change in comparison to (D1). For the Web-book-store they are "select book", "provide payment information" and "order book". To complete the buying task the system additionally has to support the "delivery of the book" task of the bookseller.

- **(D3) Decisions about the system responsibilities:** Typically, the system does not support all to-be-activities, but only a subset. These are the system responsibilities. These decisions clarify the key-contribution of the system.

  > *Example*: The booksellers' activity to "inform customer about shipping of the order" is supported by the software, and thus is a system responsibility. In contrast, the to-be-activity "package book" is not supported by software.

- **(D4) Decisions about the domain data relevant for a task:** System responsibilities of UIS manipulate data. Decisions have to be made what domain data is relevant for the system responsibilities.

  > *Example*: "Book", "order" and "customer" are examples of domain data.

- **(I1) Decisions about the system functions:** System responsibilities are realized by system functions. The decision about the system functions determines the border between user and system.

  > *Example*: The system responsibility "select book" is supported by the system functions "search book" and "shopping bag".

- **(I2) Decision about user-system interaction:** It has to be decided how the user can use the system function to achieve the system responsibilities. This determines the interaction between user and system.

  > *Example*: The following interaction defines the system responsibility "select book":
  > - User calls the "search book" function by specifying search criteria.
  > - System displays a list of books.
  > - User marks one or more books and calls the "shopping bag" function.
  > -   System stores the marked books in the shopping bag.

- **(I3) Decisions about interaction data:** For each system function the input data provided by the user as well as the output data provided by the system has to be defined.

  > *Example*: Interaction data of the "search criteria" example in (I2) is for example "book title", "author name", "ISBN", "key-word".

- **(I4) Decision about the structure of the user interface (UI-structure):** Decisions about the grouping of data and system functions in different *workspaces* have to be made. System functions and data grouped in one workspace will be close together in the graphical user interface (*GUI*). This means that users need less navigation effort in the interface to invoke system functions and view data within the same workspace. By the UI-structure the rough architecture of the user interface is defined. This structure has a big influence on the usability of the system.

  > *Example*: The shopping system has three workspaces a "select book" workspace, the "place order" workspace and the "provide customer data" workspace (see Figure 2 at the end of the paper).

- **(C1) Decision about the application architecture:** The code realizing the system functions is modularised into different components. In the decision about the component architecture existing components and physical constraints as well as quality constraints such as performance have to be taken into account. During requirements only a preliminary decision concerning the architecture is made. This is refined during design and implementation.

  > *Example*: The software follows the model-view-controller paradigm consisting of three subsystems: the core, the GUI and the database.

- **(C2) Decisions about the internal system actions:** Decisions have to be made regarding the internal system actions that realize the system functions. The system actions define the effects of the system function on the data. These decisions also define an order between the system actions as far as is necessary to understand the behaviour of the system function.

  > *Example*: The "place-order" function internally checks whether the customer paid bills of previous orders.

- **(C3) Decisions about internal system data:.** The internal system data refines the interaction data to the granularity of the system actions. The decisions about the internal system data reflect all system actions.

  > *Example*: To check whether the customer paid bills of previous orders, a "payment behaviour" record has to be added to the customer data.

- **(G1) Decisions about navigation and support functions:** It has to be decided how the user can navigate between

different screens during the execution of system functions. This determines the navigation functions. In addition support functions that facilitate the system functions have to be defined. These functions realize parts of system functions that are visible to the user, for example by processing chunks of data given by system functions in a way that can be represented in the user interface. Another example are support functions that make the system more tolerant against user mistakes.

> *Example*: A support function is the function "check address" that checks for the completeness of the customer address before the complete order is submitted. This avoids incomplete order information.

- **(G2) Decision about dialog interaction:** For each interaction the detailed control of the user has to be decided. This determines the dialog. It consists of a sequence of support and navigation functions executions. These decisions also have a strong influence on the usability of the system.

  > *Example:*
  > - User presses "send order" function.
  > - Systems checks for completeness of order information.
  > If e.g. the "shipping address" is missing, the system asks the user to specify the"shipping address". It opens the "customer account" screen containing the address fields.
  > - User types "name", "street" and "city". User selects "country" from a list. User presses "send order" function again.
  > - System shows "Thank you" screen and sends confirmation mail.
  > -

- **(G3) Decisions about detailed UI-data:** For each navigation- and support-function the input data provided by the user as well as the output data provided by the system has to be defined. These decisions determine the UI-data visible in each screen.

  > *Example*: To specify the country of the shipping address a "choice box" lists all European countries.

- **(G4) Decisions about screen-structure:** The separation of workspaces as defined in (I4) into different screens that support the detailed dialog interaction as described in (G2) has to be decided. The screen-structure groups navigation and support functions as well as UI-data. The decisions to separate the workspaces in different screens are influenced by the platform of the system.

  > *Example*: The "select book" workspace is realized by two HTML-Pages. One to search for books ("search book screen") and one to view details of selected books ("book detail screen").

3. Implications for an integrated process

There is no approach so far in the literature which covers all REDT presented in the last section. SWE approaches such as the RUP [1], typically focus on D2-D4, I1,I3 and C1-C3. HCI-approaches focus on task modelling (T1) and user interface concepts (T1, I2-I4, G1-G4), e.g. [5][6][7][8]. With the advent of use cases, I2 and sometimes also G2 are nowadays also designed as part of the RE process, e.g. [9]. In practice, typically D1-D4, I1-I4 and C1-C3 are fixed separately from G1-G4. Often HCI-models such as user interface prototypes are used to stimulate elicitation of requirements, but the decisions wrt. the user interface are not seen as part of RE. One notable exception is [10] which uses the user interface design to drive the requirements specification.

In the following we argue that there are inherent dependencies between these REDTs which imply that HCI and RE activities must be closely intertwined.
The REDTs are aligned on 4 abstraction levels:

- **Task level**: The motivation for users to use a UIS is their work. UIS support the tasks users do as part of their work in a specific role. Decisions about the roles and tasks to be supported by the UIS are made on this level.

- **Domain level**: Looking at the tasks in more detail, reveals the activities users have to perform as part of their work. These activities are influenced by organizational and environmental constraints. At this level, it is determined how the work process changes as a result of the new system. This includes in particular the decision what activities will be supported by the system and which domain data is relevant for these activities.

- **Interaction level:** On this level decisions about the partition of activities between human and computer are made. They define how the user can use the system functions to achieve the system responsibilities. This decision has to be aligned with the decision about the UI-structure, which the user can use to invoke the system functions.

- **System level**: Decisions about the internals of the application core and the graphical user interface *(GUI)* are on the system level. They determine details about the visual and internal representation of the system to be developed.

Each level corresponds to a specific view on the system and its context on a specific level of detail. Furthermore, the decisions on one level depend on the decisions of the

previous levels. Decisions of one level have to be made after all decisions of the previous level have been determined. If decisions of lower levels are made without taking into account the higher level decisions, the system will not support the users adequately in their tasks.

So the first major observation is that the decision about the tasks is an indispensable prerequisite for starting the RE process. As for example advocated in [11], RE approaches often start with goals. However, there is little guidance on how to identify these goals. Task support is the most important goal, since a system will only be accepted by the users, if their tasks are adequately supported. SWE can learn a lot from HCI for the identification of tasks.

The second major observation is that – in contrast to typical RE approaches - the decision about the UI-structure (I4) is an essential ingredient of the interaction level. Use Cases have shifted the focus from system functions (I1) to the interaction between system and user (I3), but without a preliminary UI-structure, it is not possible to make adequate decisions about the interaction (see [7]and [10] for forceful arguments why this is necessary). It is an interesting observation that such an integrating structure is also part of the application core and the GUI, namely the architecture (C3) and the screen structure (G4). At all levels there are decisions concerning behaviour chunks like activities, functions or actions as well as decisions concerning data. Interaction and dialog put these chunks into a sequence. UI-structure, architecture and screen-structure group data and behaviour chunks together.

A third major observation is that the design of the application core and the GUI are heavily interdependent. The details of the system functions depend on the way these functions are presented to the user. Navigation and support functions must be designed to ease the control of the user on the execution of the system functions.

Thus, altogether these dependencies imply that RE and HCI must be intertwined and thus RE and HCI experts must collaborate closely.

4. Conclusion

We have presented the fundamental decisions to be made during RE and argued that they need to include the usability engineering decisions.

The REDT and their dependencies have been identified from conceptual considerations as well as our experience. We have validated them by looking at different RE and HCI approaches. We checked whether these decisions are covered by these approaches and whether we miss issues covered in the approaches.

Of course, further application to industry-scale projects is necessary to evaluate them wrt. completeness and necessity. In [12] we sketch a specification method covering all the REDT. We believe that in practice there is

rarely time to specify all of them explicitly, but we are convinced that depending on the project context, different subsets of the decisions for different subsystem parts should be specified explicitly.

In our view an agreement about the decision types and their dependencies is an important prerequisite for the development of joint RE and HCI curriculae and joint RE and HCI processes and tools. The curriculae should cover all the REDT such that RE and HCI experts are aware of the decisions to be made by the other experts and their dependencies. Processes and tools should in particular support the intertwining of the decisions e.g. with giving detailed guidance to use decisions from the higher-levels to come up with the lower-levels decisions.

5. Acknowledgement

6. References

[1] Loucopoulos, P., Karakostas, V., *System requirements engineering*, McGraw-Hill, 1995

[2] Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., and Verplank, W., ACM SIGCHI Curricula for Human-Computer Interaction, 1996, htttp://www.acm.org/sigchi

[3] Good ,M, Spine, T. M., Whiteside, J. , George, P., User-derived impact analysis as a tool for usability engineering, Conference proceedings on Human factors in computing systems, April 1986

[4] Kruchten, P. B., *The Rational Unified Process: An Introduction*, Addison-Wesley, 2000

[5] Diaper, D., *Task analysis for human-computer interaction.* Ellies Horwood, 1989

[6] Hackos, J.T., Redish, J.C., *User and Task Analysis for Interface Design*, John Wiley & Sons, 1998

[7] Beyer, H., Holtzblatt, K., *Contextual Design: Defining Customer Centered Systems*, Morgan Kaufmann Publishers,1998

[8] Constantine, L., Lockwood, L., *Software For Use*, Addison Wesley, 1999

[9] Armour, F., Miller, G., *Advanced Use Case Modeling*, Addison-Wesley, 2000

[10] Lauesen, S., Harning, S., "Virtual Windows: Linking User Tasks, Data Models and Interface Design*", IEEE Software*, pp. 67-75, July/August 2001

[11] Cockburn, A.,*Writing Effective Use Cases*, Addison Wesley 2001

[12] Paech, B., Kohler, K., Task driven requirements in object-oriented development, in Leite, J., Doorn, J.,(eds.) *Perspectives on Requirements Engineering*, Kluwer Academic Publishers, to appear
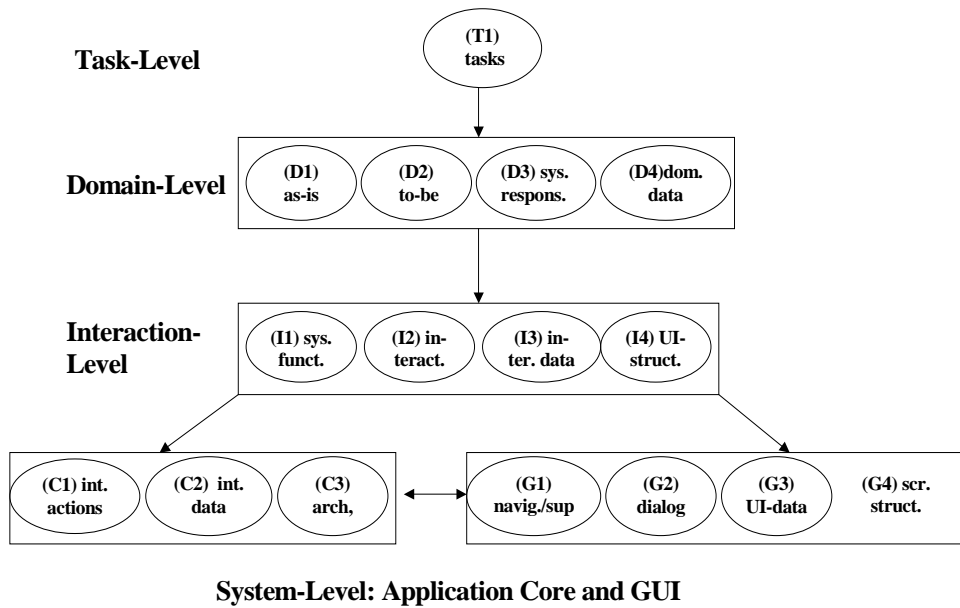
Task-Level

```
(T1)
tasks
```

Domain-Level

```
(D1)      (D2)      (D3) sys.   (D4)dom.
as-is     to-be     respons.    data
```

Interaction-Level

```
(I1) sys.   (I2) in-   (I3) in-    (I4) UI-
funct.      teract.    ter. data   struct.
```

```
(C1) int.   (C2) int.   (C3)          (G1)        (G2)     (G3)      (G4) scr.
actions     data        arch,         navig./sup  dialog   UI-data   struct.
```

System-Level: Application Core and GUI

Figure 1: Requirements Design Decision Types

Select Books

Search books
Purpose: Selection of books
Data:
- search criteria
- list of books with title and author
Function:
- search
- move to shopping bag

Book details
Purpose: Detailed info about book
Data: abstract, picture of cover,
ISBN Nr., year, review, order
conditions, availability
Function:
- move to shopping bag

Place Order

Order
Purpose: Definition of order
conditions
Data: Payment method, address
Function:
- submit order

Shopping Bag
Purpose: overview about selected
books
Data: shopping bag, total sum
Function:
- delete item from list
- move to memo

Memo
Purpose: Keep list of interesting
books
Data: memo list
Function:
- delete item from memo list

Customer account
Purpose: View and change
information about customer
Data: status of order, Email,
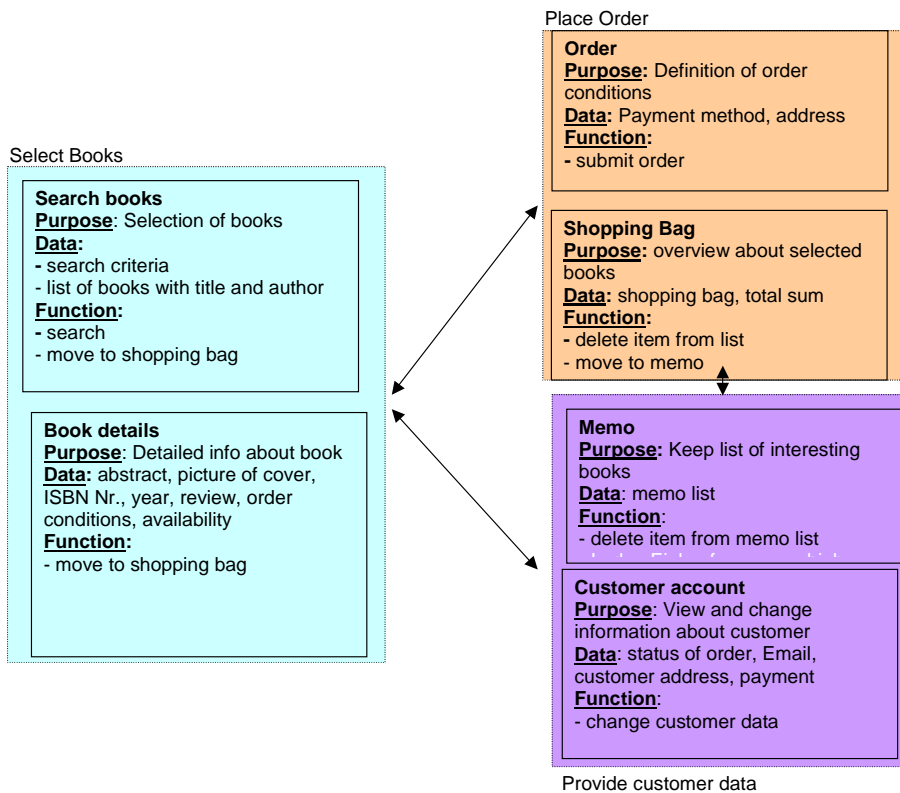customer address, payment
Function:
- change customer data

Provide customer data

Figure 2: Workspace example