4 Designing an Ubiquitous Computing Application Development and Evaluation Process Model (UCAN)

In this chapter existing theories are used to develop a process model for developing and evaluating Ubicom applications.

Very early on Weiser laid out a potential research method for ubiquitous computing: “the construction of working prototypes of the necessary infrastructure in sufficient quantity to debug the viability of the systems in everyday use” (Weiser 1993). Weiser might have known this without needing to experience the potential of Ubicomp technology as we see it today. Unless Ubicomp technology is readily available on the market, a thorough user evaluation is almost impossible without working prototypes (see also 3.2.4). Given this fact, the development of the process model will begin with prototypes (see Chapter 2.3) since they also provide a basis for evaluating technology acceptance (see Chapter 3.2). For the purpose of this analysis the process model is based on combining information from the HCI chapter (see Chapter 3.1) and the section on innovation adoption and technology acceptance (see Chapter 3.2). Following these sources, the process model should confirm the following goals:

- The process model assists to identify potentially negative or unneeded functions of an application in an early phase of a development process and allows changes in product development.
- It measures the technology acceptance before entering the market as a final (and thus hardly changeable) product and helps to identify marketing measures for a successful market entry.

For Ubicomp application development it is critical to be able to change and adjust an application during the development process without the risk of losing significant investments in infrastructure (sunk costs). The process model connects various stages of prototypes with adequate evaluation and analysis methods. The process model will be tested using three artifacts from case studies. It serves as a utility theory for this analysis.

This chapter employs theory to develop the process model. First, the challenges related to developing Ubicomp applications are outlined to serve as the groundwork of the model (Chapter 4.1). Then the systematic processes related to evaluating Ubicomp applications will be described (Chapter 4.2), before laying out an initial a-priori process model (Chapter 4.3). Chapter 4.4 describes how the model will be tested with a selection of case studies and why the corresponding case studies were chosen.

4.1 Ubiquitous Computing Application Development

In the previous chapters (see chapter 2.2), several challenges of Ubicomp applications were described: First, creating consumer awareness of applications presents problems. Second, high initial investments in infrastructure makes widespread adoption and quick development of Ubicomp applications almost impossible. Third, the difficulty to develop Ubicomp applications based on current knowledge is limiting the number of current applications available. Section 3.1.1 highlighted additional constraints for humans and their complex
patterns in an Ubicomp world. Taking these factors into account, developing applications in a Ubicomp world differs from developing applications that require less physical presence. In order to shed light on Ubicomp application development, section 4.1.1 describes the importance of non-functional requirements in Ubicomp systems and the differences related to traditional systems. Section 4.1.2 elaborates on the challenges of describing Ubicomp application requirements and why prototypes are helpful in this task. Section 4.1.3 emphasizes the importance of integrating the end user in the development of products or applications. Section 4.1.4 follows with an explanation of how prototypes can provide solutions to integrating the end user. Section 4.1.5 ends by explaining the relationship between system engineering and prototyping.

4.1.1 Determining Initial Requirements

To avoid the pitfalls related to developing Ubicomp applications, determining the necessary functional requirements for developers is needed at an early stage. Davis states three reasons that make obtaining a correct and complete set of requirements difficult (Davis 1982, 5):

- The constraints on humans as information processors and problem solvers.
- The variety and complexity of information requirements.
- The complex patterns of interaction among users and analysts in defining requirements.

The Institute of Electrical and Electronics Engineers (IEEE) defines requirements as “a condition of capability needed by a user to solve a problem or achieve an objective” (standard IEEE 610.12) in (IEEE 1990, 1998). Requirements are usually defined in early stages of system development to specify what should be implemented (Kotonya/Sommerville 1998, 6). It is important to prescribe only those requirements that cover what users expect, are implementable, and can be validated (Bjorner 2006, 368-370). Requirements engineering uses systematic and repeatable techniques to determine, document, and maintain a set of requirements (Eide/Stålhane 2005). Requirements are divided into system requirements which apply to the system as a whole. According to the IEEE document (IEEE 1998), systems requirements were traditionally regarded as a document for communicating user requirements to the technical community. The collection acts as a bridge between the two groups. There are several models that help support a systems and requirements engineering process (Bjorner 2006; Kotonya/Sommerville 1998; Pomberger/Blaschek 1993, 1997).

For many types of information systems it is impossible to distinguish the requirements of the software from broader requirements of the system as a whole. Ubicomp systems range from small embedded systems to large scale infrastructures, making it almost impossible to gain information on detailed software requirements at an early stage. In particular the user-

---

14 The distinction between system, application, and software is again relevant to an overall understanding of this thesis. Here, the range of ubiquitous computing building blocks—more of an eco-system—is described as a system that contains all the relevant parts for shaping the development of the application (see definition in chapter 2.1.1).