Abstract

In the last five years, several studies and workshops have highlighted the gap between software design approaches in HCI (Human Computer Interaction) and software engineering. Although the fields are complementary, these studies emphasize that they are not well integrated with each other. Several frameworks have been proposed for integrating HCI and usability techniques into the software development lifecycle. This chapter reviews some of the most relevant frameworks. It assesses their strengths and weaknesses as well as how far the objective of integrating HCI methods and principles within different software engineering methods has been reached. Finally, it draws conclusions about research directions towards the development of a generic framework that can: (1) facilitate the integration of usability engineering methods in software
development practices and, (2) foster the cross-pollination of the HCI and software engineering disciplines.

3.1 INTRODUCTION

It is no coincidence that the titles of some chapters in this book evoke the terms “solitudes” and “competition” to characterize the relation between the fields of HCI (Human-Computer Interaction) and SE (Software Engineering) (cf. Jerome and Kazman, chapter 4; Sutcliffe, chapter 5). This uneasy cohabitation dates back to the early days of HCI when human-centered design has been presented as the opposite, and sometimes as a replacement, to the system-driven philosophy generally used in software engineering (Norman and Draper, 1986). Although numerous HCI researchers and practitioners view User Centered Design (UCD) as a process and as a set of specific methodologies to design and develop interactive software applications, HCI is by no means considered a central topic in SE. For example, the SWEBOK, an IEEE initiative for the definition of SE knowledge and practices, defines HCI as a “related discipline”, termed “software ergonomics” (Abran et al., 2004). Usability is considered one of many non functional requirements and quality attributes. No reference is made to specific UCD methods such as those found in the ISO 13407 standard, “Human-centred design processes for interactive systems” (ISO/IEC, 1999).

The reality is that UCD and software engineering techniques do have overlapping objectives of defining which methods to use in the software development process, what kind of artefacts (documents and deliverables) to use, and what quality attributes to prioritize. However, we argue that they have different perspectives on the software development process itself, as depicted in figure 3.1. The SE community focuses on the “system 1” perspective in Figure 3.1: software development is driven by specifications provided for defining the application, including the interface. The user interface has to meet the functional and usability requirements, but the requirements are tied to the system, that corresponds to the application itself. The focus is on the software application and the interface is one of many components that has to meet certain requirements.

In contrast, the proponents of UCD, and more specifically the “quality in use” approach (Bevan, 1999), focus on the “system 2” perspective: The priority is to ensure that each class of users can perform a given set of tasks with the application. The ultimate requirements are tied to what the user can perform with the application. Consequently, the software development process is driven by the need to define and validate these requirements and closely depends on the tasks defined and the users’ capabilities and characteristics.

The two perspectives do not differ solely from their philosophical stance. The can have significant impacts on the how the software development process will be defined and planned, which activities will be conducted, which tools will be used, and the expertise of the team members and its leader. The impacts are particularly important with regards to the requirements management and quality control activities.

While both perspectives are valid, the SE approach is always necessary, since there necessarily is a “system 1” perspective. It either stands alone in the absence of a significant user interface component, or it is embedded in the “system 2” perspective.