Abstract

User interfaces (UI) pervade our daily lives. To do office tasks, to pilot an airliner, to write programs, UIs are the tangible vectors that enable users to interact with software systems. The development of UIs involves multiple roles. Designers and ergonomists are in charge of the design and evaluation of UIs from a strict human factor viewpoint. They use concepts and theories established by the Human-Computer Interaction (HCI) community. Software engineers develop, validate, maintain UIs using software engineering techniques. UI engineering is an interdisciplinary field that cross-cuts these two roles and their underlying domains, HCI and software engineering. In the 80’s Draper and Norman motivated the UI engineering field as follows:

"The discipline of software engineering can be extended in a natural way to deal with the issues raised in a systematic approach to the design of human-machine interfaces. To a larger extent all that is needed is to take the problem of engineering the user interface as seriously as any other part of software engineering and to apply to it the same kind of techniques, appropriately adapted".

The IFIP Working Group on UI engineering proposes a more technical definition of UI engineering:

"UI engineering addresses all aspects related to methods, processes, tools, technologies, and empirical studies involved in the invention, design and construction of interactive systems [...] with a particular focus on principled, methodological engineering approaches".

These definitions agree on one point: UIs are complex pieces of software that require specific development approaches. UI engineering involves the software engineering and HCI communities that should work together, but as explained by Palanque:

"Innovation and creativity are the main research drivers of the HCI community, which is currently investing a vast amount of resource in the design and evaluation of 'new' user interfaces and interaction techniques, leaving the correct functioning of these interfaces at the discretion of the helpless developers."

This problem of interactions between these two communities is also argued by HCI researchers such as Beaudouin-Lafon:

"HCI researchers have created a variety of novel [user] interaction techniques and shown their effectiveness in the lab, such 'point designs' are insufficient. Software developers need models, methods, and tools that allow them to transfer these techniques to commercial applications."

A gap thus exists between how the HCI community envisions a UI and how software engineers are currently able to implement it. In this "habilitation à diriger des recherches" (HDR), I overview a decade of research work in the fields of UI engineering and software engineering. I defend the following thesis as a global objective: UI engineering research has to provide software engineers with theories and techniques at a correct level of abstraction and assessed empirically to help them in coding, testing, documenting usable modern UIs. I develop this thesis through two research axes.

In the first axis I detail new UI engineering concepts. These concepts follow the same leitmotiv: engineering UIs is a specific problem that requires specific abstractions. These abstractions thus go beyond the classical object-oriented abstractions provided by the current programming languages to provide abstractions that focus on UI concerns.

In the second axis I focus on domain-specific languages (DSL). DSLs are specific user interfaces that stand between domain experts and their engineering problems. As for any UI, engineering DSLs must make use of HCI concepts at different levels. First, DSL editors must be usable and their editors must be supplemented with adapted interactive features. Second, developing DSL is a complex engineering job that involves various tasks, such as developing, testing, or maintaining editors. Language designers must be helped during these tasks to provide domain experts with usable DSL editors.