Addressing Interactive Computing Systems' concerns in Software Engineering degrees

José Creissac Campos[0000−0001−9163−580X] and António Nestor Ribeiro[0000−0002−0681−8909]

Department of Informatics, University of Minho & HASLab/INESC TEC
Campus de Gualtar
4710-057 Braga, Portugal
{jose.campos,anr}@di.uminho.pt

Abstract. This paper arises from experience by the authors in teaching software engineering courses. It discusses the need for adequate coverage of Human-Computer Interaction topics in these courses and the challenges faced when addressing them. Three courses, at both *licentiate* and master's levels, are used as triggers for the discussion.
The paper argues that the lack of relevant Human-Computer Interaction concepts creates challenges when teaching and learning requirements analysis, design, and implementation of software systems. The approaches adopted to address these challenges are described.

Keywords: Human-Computer Interaction · Software Engineering · Education.

1 Introduction

As pointed out by John et al. [4], Software Engineering and Human-Centred Development (HCD) [3] processes have different concerns. Typically, the former will be more concerned with the quality of systems from a technical perspective, while the latter will be more concerned with the human aspects of such quality. Computer Science and Software Engineering degrees should provide students with a balanced perspective of both types of concerns. This is not always the case, and sometimes the degrees can be biased towards one area or the other.

This paper arises from experience by the authors in teaching software engineering courses on an integrated master's degree[1] on Informatics Engineering at the University of Minho. It discusses the need for adequate coverage of Human-Computer Interaction (CHI) topics, and the challenges faced when addressing them in the particular context of the degree in question.

The paper focuses, in particular, on three courses. One at the end of the *licentiate* level (Software Systems development – Semester 5). Two others at the master's level (Applications Architectures and Interactive Systems – both on

[1] A five years degree awarding a *licentiate* at the end of 3 years and a master's on completion.

Semester 8). Together these courses cover from model-based software development to Web-based applications development.

Human-Computer Interaction topics appear only at the master's level, in the Interactive Systems course, and this creates a number of challenges that are discussed herein. The paper outlines the approaches adopted to address these challenges and describes recent changes to the degree.

The paper is structured as follows: Section 2 provides context on the Informatics Engineering degree and on the three courses; Section 3 discusses human-centred concerns in relation to the courses; finally, Section 4 presents some concluding remarks.

2 The degree and the courses

In this section we briefly describe the integrated master's on Informatics Engineering and the three courses mentioned in the previous section.

2.1 The Informatics Engineering degree

The degree is a combination of a *licentiate* and a master's degree in Informatics Engineering (see Figure 1). It aims to prepare computer engineers, capable of intervening in all phases of the development of computer solutions, from analysis to installation, through design and implementation. The degree lasts ten academic semesters, from which six semesters are for the *licentiate*, and four for the master's course. It awards a total of 300 ECTS[2] [2] credits.

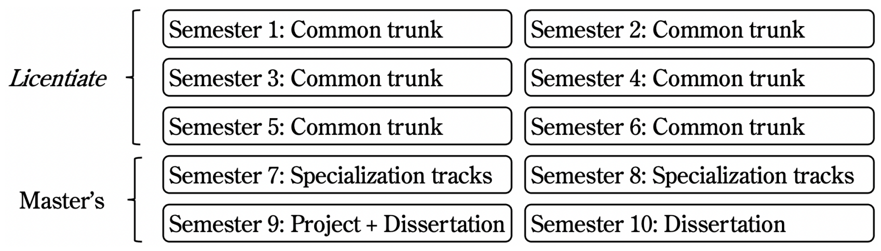

Fig. 1. The integrated master's structure

The *licentiate* part correspond to a general training in Informatics Engineering, constituting a common core in which the study plan is essentially composed of mandatory courses. At the master's level, students must choose two specialisation tracks in the fourth year, and carry out a project and dissertation in the fifth. All courses are 5 ECTS, except for the fifth year project (15 ECTS) and the dissertation (45 ECTS).

[2] European Credit Transfer and Accumulation System.

The Applications Architectures and Interactive Systems courses are part of a specialisation track on Web applications engineering.

2.2 The Software Systems Development Course

This is a course at *licentiate* level, appearing in the fifth semester of the degree. The course follows on from an introductory Object Oriented Programming course, and aims to provide students with object-oriented modelling skills (in UML [1]) in order to enable them to design and develop applications of a larger scale and complexity.

In terms of the learning outcomes that are intended to be achieved, students should be able to:

1. Characterise the different phases of the Unified Process [6] and related activities;
2. Interpret the different types of UML diagrams;
3. Assess which diagram is the most appropriate according to the different modelling needs;
4. Critically evaluate models (requirements / structural / behavioural) described in UML;
5. Design software systems using UML;
6. Implement software systems based on UML models.

The ACM/IEEE-CS curricula on Software Engineering [8] defines three cognitive skill levels:

- Knowledge: the ability to remember previously learned information
- Comprehension: the ability to understand the information
- Application: the ability to use learned information

Compared to these skill levels, learning outcome 1 refers to the level of Knowledge, learning outcome 2 to Comprehension, and learning outcomes 3 to 6 to the Application of acquired knowledge. Therefore, there is an emphasis on the need to reach a level of cognitive proficiency that supports the ability to apply the acquired knowledge.

The teaching approach consists of a fifty-fifty mix of theoretical lessons and practical sessions. Assessment considers three elements: a written exam, a group project and a continuous evaluation component. One goal of the course is that students develop an application according to the three tier architecture, covering the data, business and user interface layers. While a databases course is being taught in parallel, covering the aspects related to the construction of the data layer, students have no previous exposure to HCI topics.

2.3 The Web applications engineering track

As mentioned above, the master's component of the degree is organised into specialisation tracks, each composed of four courses on the track's topic. The

Web applications engineering track consists of courses covering from database infrastructures to user interfaces development. Two courses are relevant here: the Applications Architectures (AA) course, which covers the business layer of Web applications (e.g. applications servers, components, services), and the Interactive Systems (SI) course, which covers the user interface layer of Web applications.

In terms of learning outcomes, after the **Interactive Systems course**, students should be able to:

1. design user interfaces with consideration for usability aspects
2. apply model-based approaches to design user interfaces
3. select and apply the most appropriate interface assessment techniques for a given context
4. apply appropriate user interface development techniques
5. develop the user interface layer of applications with support for controlled and independent evolution of the logic and data layers

Students in the **Applications Architectures course** should be able to:

1. know the main structural and behavioural patterns used for software system development
2. know how to design parameterizable software architectures
3. identify the main characteristics of the most commonly used application servers
4. design computational layers that allow controlled and independent evolution of presentation and data layers

The teaching approach consists mainly on practical sessions. In both cases, the emphasis is on developing the capabilities to perform independent work and acquire relevant knowledge independently (the Comprehension and Application skills). Assessment includes a written exam, a group project, shared between both courses, and a continuous evaluation mark, reflecting the students performance during the sessions.

3 Human-centred concerns

As already mentioned, during the period under analysis the degree lacked a course on Human-Computer Interaction. This had several implications in the courses described above. They are discussed in this section.

3.1 Implications in the Software Systems Development Course

Originally, the course was very much focused on modelling and developing the business layer of applications. The success of the course notwithstanding, this created a number of challenges at several levels, namely:

- At the requirements analysis level
- At the design level
- At the implementation level

Additionally, the students' motivation must also be considered.

Requirements analysis Students faced challenges in focusing on the problem, instead of the solution. Indeed, when discussing requirements, it was easy to quickly get caught on the technical details of an idealised solution, without paying proper attention to the problem under analysis, which typically involved considering the users of the (to be developed) system and their needs.

Additionally, students had difficulties in establishing a clear separation between functional requirements and user interface specifications, which proved to be a very strong constraint. Indeed, given the lack of a user interface design step, it was all too common for students to express user interface related concerns, such as dialogue control, in the use case specifications thus creating too complex and unfocused specifications.

The above points to the need of having a basic knowledge of Human-Computer Interaction, in particular, Human-Centred Development processes. Exploring the links between requirements engineering and HCI techniques will help highlight the role and impact of users in the definition of system requirements. Including an explicit user interface design step, will help separate requirements from the design of a system that satisfies them.

Design Students faced challenges in deriving an architecture for the business logic layer from requirements models. The first step in the approach that was taught was to derive an application programming interface (API) for the business logic layer, starting from the use case specifications. This API needs to adequately support a user interface satisfying the requirements expressed in the use cases. However, without a model of the user interface control logic and an understanding of the specific nature of the user interface layer programming model (e.g. its event-based nature), students faced difficulties deriving this API.

Attempts to use a unified modelling approach to support the process, covering from use cases to business logic, were faced with problems. On the one hand, the step from functional requirements to business logic API, without any intermediating model for the user interface, required making assumptions about how functionalities would be provided to users, since that would have an impact on which operations would be needed at the business logic (e.g. "will undo support be needed?"). On the other hand, since different modelling approaches are better suited for each layer (cf. state machines for expressing dialogue control in the user interface vs. sequence diagrams for expressing cooperation between objects at the business logic layer), expressiveness problems were felt when trying to use a single notation.

The above points to the need to introduce user interface design and, in particular, prototyping, as a means to bridge from functional requirements to the business logic. The goal is to support a process where the business logic API is driven by the needs of the user interface needed to satisfy the identified requirements. This will surely create a smoother process in what concerns the elicitation of the required API, and the correct identification of where the relevant handlers are in the user interface.

Implementation At the level of the implementation, the lack of knowledge of user interface development technologies proved a difficulty when attempting to develop the three tiered implementation that was sought. Although relevant architectural patterns for the user interface layer (e.g. the Model-View Controller pattern [5]) were in fact covered in previous courses, user interface programming models, toolkits and frameworks had no formal coverage. Hence, there was a need to address basic knowledge of user interface programming, from layout management to event-based programming. In this case, Java technologies were used.

Managing students' motivation Given the profile of the degree, students attending the course tended to be mainly focused on the technical aspects of software development. This was coupled with limited experience in developing complex software systems, which the course was designed to address. Hence, not only had the students be shown the potential benefits of using a model-based approach in the development of complex software systems. They had also be made aware of the relevance of considering the users during the design and development of such systems.

The group project can play a major role in addressing these challenges. The project presented an opportunity to challenge the students to develop a reasonable complex system in the course of the semester. Additionally, by choosing appropriate themes for the project, it can be used to stress the need to consider user concerns. The goal is to propose topics where user tasks have a reasonable level of complexity, so that students are prompted to consider how best to support them.

A typical example is a system to support electronic prescriptions, since they have to obey a number of rules on how medication can be combined that do not necessarily match the goal of the doctor to prescribe a treatment. One issue, then, is to know who the responsibility of creating valid prescriptions will be shared between user and system.

3.2 Implications in the Applications Engineering track

Although basic knowledge of user interface development (e.g. the notion of usability and notions of rapid prototyping) started being addressed in the Software Systems Development course, the Interactive Systems course of the Applications Engineering track was the first truly HCI course in the degree. This had several implications to that course and the track.

A first tension emerged when defining the syllabus, between covering Human-Centred Development topics vs. covering the technical aspects of (Web-based) user interface programming. On the one hand, technical skills are, of course, needed in order to allow for the project to be carried out, and to support the integration with the other courses in the track. Indeed, as mentioned above, students are already mainly focused on the technical aspects of software development. That said, a confounding factor is the current diversity, and rate of change, of frameworks supporting the development of Web-based user interfaces. They present a challenge, not only because the choice of a specific framework is a decision whose merits might change from year to year, but also because of the continued evolution of any specific framework, which means they are in fact a moving target. This, coupled with a diverse range of previous competences at the technical level by the students, raised further difficulties, making it harder to identify the best approach to cover the technical aspects.

On the other hand, students mostly lacked the Human-Centred Design competences to design and develop quality user interfaces, making it a necessity to cover those topics. Through a number of iterations of the course, the option became to give prominence to Human-Centred Design, with a roughly two-thirds/one-third split. The goal being to make students aware of the need to consider users during the design and development of interactive computing systems, and provide them with the fundamental skills to do so.

Two complementary approaches were used to put the above into practice. More guided lectures on the HCD part, focusing on HCI fundamentals, task analysis, prototyping and expert inspection methods. Tutorial based lectures, allowing students to progress at their own pace, on the programming technologies related part. Additionally, students were free to use their Web technology of choice in the group project, with concrete frameworks being introduced as examples in the tutorials (e.g., Bootstrap[3] and Vue.js[4]). A tutorials based approach has the additional advantage of making it easier to adapt to new technologies or the evolution of the ones used.

With regard to the Applications Architectures course, we attempted to make up for this lack of knowledge about HCI processes and principles by resorting to the construction of low-fidelity prototypes, focusing particularly on the identification of the call points to the business layer API.

4 Concluding remarks

We have illustrated the relevance of an HCI perspective on a software engineering degree. Decisions regarding the design of the user interface can have serious implications on the implementation of the whole system, not just the user interface implementation. Even in the case of non-interactive systems, input from the HCI body of knowledge can help in understanding the impact of the system in the overall context, when this context involves humans.

In practice, however, reconciling the two views on development becomes difficult. This can be attributed to a number of factors. We highlight the following two:

– different views on where the development focus lies – Software Engineering is mainly interested in solving the technical difficulties faced when implementing a given functionality; HCI is mainly about solving the problem of which

[3] https://getbootstrap.com, last visited November 15, 2021.
[4] https://vuejs.org, last visited November 15, 2021.

functionality should be provided, and how to optimise the way in which it is provided to users;
- communication difficulties – not always the same terms are used to describe the same concepts in the two communities, this hinders communication; at best it can slow it down, and at worst it can be misleading, with the two parties thinking that they are talking about the same thing when in fact they are not. This is particularly relevant in a teaching context, since concepts need to be presented in a clear and non confusing way.

Partly as a result of the experience with these courses, the decision was made to create a new *licentiate*-level course on Human-Computer Interaction. This new course will free up the master's level course to address more advanced topics, but means the tension between HCD methodology and programming technology will now be transferred to a course were students have less background.

In order to balance them, it is relevant to understand how the current approach fairs. Through informal interviews and questionnaires it was possible to clearly identify two groups of students. Those with some previous knowledge of Web programming technologies, tended to feel that the lectures were too guided and would like more room for exploration. Those without that previous knowledge, tended to prefer a more guided approach. These results point to two very different perspectives. Considering the profile of the students in the new course, a more guided approach seems favourable. The issue remains of how to establish a balance between HCD and user interface programming.

A pilot study was also carried out, applying the Usability Experience Questionnaire (UEQ) [7] to asses the tutorials, in the latest edition of the course. While, preliminary results pointed to a somewhat neutral to positive evaluation of the experience of using the tutorials, the number of responses (37) was too small to allow confidence in the results. One relevant question is whether this is the best approach to evaluate the students' experience with the tutorials and their value as a learning tool. There were indications of relevant inconsistencies in some the answers that raise questions at this level.

Overall, the expectation is to have a split between HCD and user interface programming topics that will be closer to fifty-fifty. The rationale is that students will need more guidance in the programming part. It should be possible to achieve this balance with minimal impact on the HCD topics, since *licentiate* courses have 33% more contact hours when compared to master's courses, which reply more heavily on independent work by the students. At the master's level, the goal is now to address more advanced topics in the engineering of interfaces, in particular related to the development of safety critical interactive systems.

References

1. Arlow, J., Neustadt, I.: UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design. Addison-Wesley Professional, 2nd edn. (2005)
2. Directorate General for Education, Youth, Sport and Culture: ECTS users' guide 2015. European Commission (2015)

3. ISO: ISO 9241-210:2019 Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems. International Organization for Standardization
4. John, B., Bass, L., Adams, R.J.: Communication across the HCI/SE divide: ISO 13407 and the Rational Unified Process. In: Stephanidis, C. (ed.) Proceedings of the Tenth International Conference on Human-Computer Interaction. pp. 484–488 (2003)
5. Krasner, G., Pope, S.: A description of the Model-View-Controller user interface paradigm in the Smalltalk-80 system. Journal of Object Oriented Programming **1**(3), 26–49 (1988)
6. Kruchten, P.: The Rational Unified Process. Addison-Wesley Professional, 3rd edn. (2004)
7. Schrepp, M.: User experience questionnaire handbook (2019), https://www.ueq-online.org/Material/Handbook.pdf
8. The Joint Task Force on Computing Curricula: Software Engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering. Tech. rep., ACM & IEEE-Computer Society, New York, NY, USA (2015)