How a Human-Centered Approach Impacts Software Development

Xavier Ferre and Nelson Medinilla

Universidad Politecnica de Madrid
Campus de Montegancedo
28660 - Boadilla del Monte (Madrid), Spain
{xavier, nelson}@fi.upm.es

Abstract. Usability has become a critical quality factor in software systems, and it requires the adoption of a human-centered approach to software development. The inclusion of humans and their social context into the issues to consider throughout development deeply influences software development at large. Waterfall approaches are not feasible, since they are based on eliminating uncertainty from software development. On the contrary, the uncertainty of dealing with human beings, and their social or work context, makes necessary the introduction of uncertainty-based approaches into software development. HCI (Human-Computer Interaction) has a long tradition of dealing with such uncertainty during development, but most current software development practices in industry are not rooted in a human-centered approach. This paper revises the current roots of software development practices, illustrating how their limitations in dealing with uncertainty may be tackled with the adoption of well-known HCI practices.

Keywords: uncertainty, software engineering, waterfall, iterative, Human-Computer Interaction-Software Engineering integration.

1 Introduction

Software development practices are mostly rooted in Software Engineering (SE), since SE as a discipline is pervasive in software development organizations all over the world. Its concepts are the ones with which the majority of developers are familiar, and this is especially true of senior management at software development organizations. HCI, on the other hand, has been traditionally considered as a specialist field, and its view of development is not as present in software development organizations as the SE perspective. According to Seffah, "HCI structure and techniques are still relatively unknown, under-used, difficult to master, and essentially not well integrated in software development teams" [24]. Nevertheless, there is an increasing interest about usability, due to the importance of graphical user interfaces nowadays [11], and about HCI methods to manage it, which will likely achieve wider user and greater impact in the near future [27].

Therefore, usability awareness has greatly risen in software development in the last decade. There is now a certain consensus on the aim of building usable systems,

J. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2007, LNCS 4550, pp. 68–77, 2007.

leading to a need for integration of usability methods into SE practices, providing them the necessary human-centered flavor. The term "Human-Centered Software Engineering" has been coined [25] to convey this idea. In contrast, HCI practitioners need to show upper management how their practices provide value to the company in the software development endeavor, in order to get a stronger position in the decision-taking process.

HCI and SE need to understand each other so that both can reciprocally complement with effectiveness. While SE may offer HCI practitioners participation in decision-making, HCI may offer their proven practices that help in dealing with the uncertainty present in most software development projects. In the next section the diverging approaches of HCI and SE are analyzed. Next, in section 3 the role of uncertainty in software development is outlined, elaborating on problem-solving strategies and how they apply to software development. Section 4 presents how joint HCI-SE strategies may be adopted for projects where uncertainty is present. Finally section 5 presents the conclusions gathered.

2 HCI and SE Development Approaches

SE is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [13]. In the pursuit of these objectives, SE has highlighted software process issues, and it has also traditionally focused on dealing with descriptive complexity.

On the other hand, HCI is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use in a social context, and with the study of major phenomena surrounding them [22]. Usability is the main concern for HCI, and it is multidisciplinary by essence. The HCI view on software development is, in a certain sense, broader than the SE one, which mostly focuses on the running system in isolation. In contrast, HCI does not handle with comparable deepness specific issues, like software process or software architecture.

Fig. 1 shows how SE and HCI differ about their main subject of interest in software development. While HCI cares about the impact created by the software in the user and his social context, SE focuses mainly on the correctness of the running software system itself.

Software engineers mostly consider usability as a user interface issue, usually dealt with at the end of development, when the `important´ part of the system has already been built. Alternatively, HCI experts carefully study the users and their tasks, in order to better fit the system to the intended users, and they consider that once the system interaction has been defined software engineers may begin `building´ the system.

There is a high degree of misunderstanding between both fields, along with some lack of appreciation for the work performed by the other discipline. Practitioners of both fields think it is them who do the "important job" in software development. Comparing HCI to SE it may look like as lacking maturity. In this direction, Mayhew states that integration of usability engineering with the existing software development lifecycle has not yet been solved, mostly due to the state of maturity of the Usability Engineering discipline [20]. Alternatively, SE methods may look too system-centered for an effective user-system interaction, as understood in HCI.

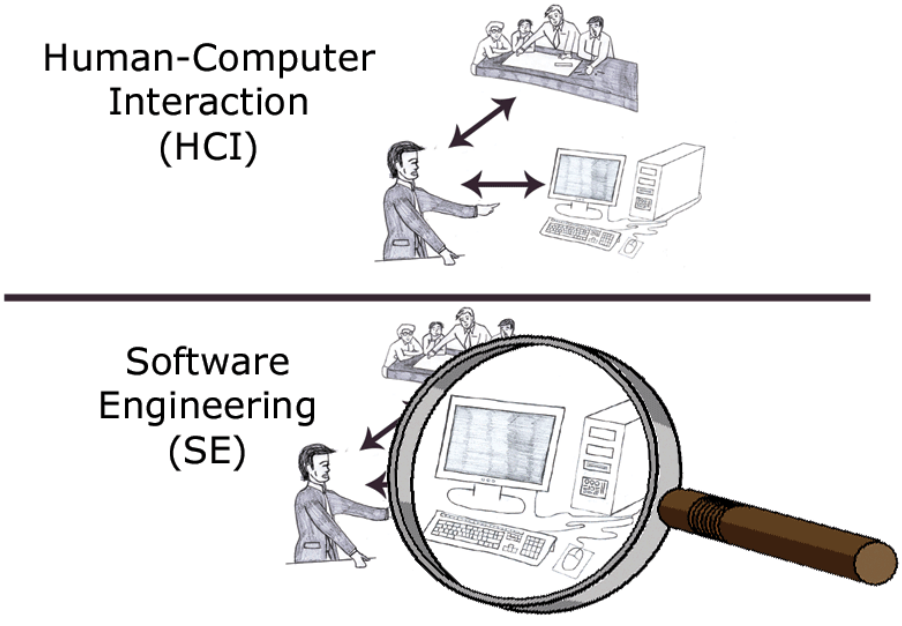

Fig. 1. Comparison between HCI and SE main focus

Despite this lack of mutual understanding, both disciplines need to collaborate, since there is a non-trivial overlapping between their respective objects of study and practice. In particular, requirements-related activities are considered a cornerstone of the work of both HCI and SE. The decision of *which system* is going to be built is quite important for usability purposes, so HCI has a lot to say about it, while requirements engineering is a SE subdiscipline with a recognized importance in the field, so software engineers will not be handing completely requirements-related activities to usability experts.

The traditional overall approach to development in SE has been the waterfall lifecycle. In relation to requirements, it is based on requirements which are fixed (frozen) at early stages of development. Nevertheless, the waterfall lifecycle is considered nowadays in SE as only valid for developing software systems with low-medium complexity in domains where the development team has extensive experience. As an alternative to the waterfall, iterative development is currently identified as the development approach of choice, even if its practical application finds some opposition. On the contrary, HCI has traditionally adopted an iterative approach to development. Therefore, some promising opportunities for SE-HCI collaboration come out.

Conflicts may arise between both kinds of practitioners, but they must be solved if usability is to be considered a relevant quality attribute in mainstream software development. Fortunately, recent trends in SE show a higher acceptance of uncertainty in software development, and this can provide a higher appreciation for HCI practices, as explained in the next sections.

3 Uncertainty in Software Development

Uncertainty is currently accepted as a necessary companion of software development [3],[19]. However, SE has traditionally considered uncertainty as harmful and eradicable. The aim was to try to define a "safe" space where no uncertainty could affect the work of software developers.

The development of software systems of higher complexity levels has led to the need of changing this approach. In order to deal with complexity, the traditional SE view considers only descriptive complexity (quantity of information required to describe the system, according to Klir & Folger [17]). It is a useful dimension to work in the software universe but, on most occasions, it is not enough on its own to explain the software universe. Descriptive complexity needs to be combined with the complexity due to uncertainty, which is defined by Klir & Folger as the quantity of information required to solve any uncertainty related to the system [17].

Ignoring uncertainty in software development obstructs the objective of better coping with highly complex problems to be addressed by software systems, since it narrows the interpretation of both the problem and the possible strategies for building a successful solution. Complexity due to uncertainty adds a new dimension to the software space, as shown in Fig. 2. When extending the software universe dimensions to two, some hidden issues that hinder software development projects are uncovered, and new solutions emerge.

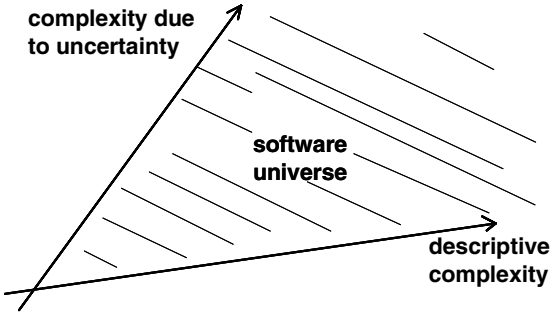

Fig. 2. Extension of the software universe when considering the uncertainty dimension

Dealing with uncertainty is unavoidable in software development. But it is not just an undesired companion in the software development journey, it can be used as a tool that offers a powerful mean of attacking recurring problems in software development. Having uncertainty-based means in the toolbox of software development teams, offers them a richer background and vision to better tackle their work in the complex software universe.

The usage of uncertainty as a tool in software development takes several forms: The introduction of ambiguity in the solution and the adoption of problem-solving strategies that manage uncertainty.

3.1 Ambiguity as a Way of Introducing Uncertainty in the Solution

Abstraction is a simplification tool that expresses just the essential information about something, leaving out the unnecessary details. This omission deliberately introduces uncertainty, which manifests in the form of ambiguity. An abstraction is precise with respect to the essence of the topic conveyed, but it is necessarily ambiguous with respect to the particulars, which are intentionally taken out of the picture.

When making design decisions, uncertainty also plays a major role in providing solutions which are easier to maintain, modify or extend. For example, the hiding information principle [21], promotes the introduction of uncertainty in the design, by not providing details on how a particular module is implemented. Modularization on its own does not provide benefits for this purpose, since a careful design of the modules and their headers is necessary for attaining the necessary relation of indifference between modules.

Any design decision that attempts to introduce some degree of ambiguity in the solution being developed uses uncertainty as a tool for allowing easier future modifications. As a collateral effect, development usually gets more complex and more difficult to manage when employing uncertainty-based strategies; in a similar way to object-oriented design being more complex than the structured development paradigm, but providing a more powerful and less constrained instrument for the development of complex software systems.

3.2 Problem-Solving Strategies and Uncertainty

Human beings use different strategies according to the extent of the uncertainty they must confront. A linear or industrial strategy may be employed with zero or negligible uncertainty; a cyclical or experimental strategy when having medium uncertainty (something is known); and an exploratory or trial and error strategy when high uncertainty needs to be dealt with. The higher the uncertainty level provided by the strategy, the higher will be its power for dealing with uncertainty (in the problem).

Linear strategy (step after step) follows a path between a starting point to an ending one, given that both points and the path between them are known in advance. That is, it is necessary to know the problem, the solution, and the way to reach such solution. If all these requirements are met, the lineal strategy is the cheapest one. In order to make possible its application, any uncertainty needs to be eradicated before beginning the resolution process. The paradigm that represents the linear strategy in software development is the waterfall life cycle. It follows the sequence requirements, analysis, design, implementation, and testing, which is a direct translation of the Cartesian precepts enunciated in the Discourse on Method [8]: Evidences, analysis, synthesis and evaluation. The idea behind these principles is to undertake in the first place the *what* and afterwards the *how*. This separation between requirements and design is an abstract goal and not a human reality [1]. The so called incremental strategy is a variant of the linear one where the problem is divided into pieces, which are then undertaken one by one.

Cyclical or experimental strategy (successive approximations), when converging, comes progressively closer to an unknown final destination through the periodical refinement of an initial proposition (hypothesis). A cyclical strategy is adopted when

the solution is unknown, but there is enough information on the issue to be able to formulate a hypothesis. The paradigm for the cyclical strategy in the software world is the spiral model [2]. A common statement in software development is to describe each cycle in the spiral model as a small waterfall. This is inappropriate, since the spiral recognizes the presence of uncertainty throughout the (risk-driven) process, and the waterfall, whatever its size, requires eradicating the uncertainty at the beginning.

Arboreal or exploratory strategy (trial and error) is the way to reach an unknown destination without a best first guess, provided that the universe is closed. In the case of an open universe, the exploratory strategy does not ensure finding the solution, but none of the other strategies may ensure it, given the same conditions of uncertainty. An exploratory strategy is in place every time a solution is discarded and development goes back to the previous situation. The Chaos life cycle [23] is very close to an exploratory strategy, but it is limited by Raccoon's waterfall mindset.

3.3 Uncertainty and HCI

HCI has developed interactive software for decades, without the obsession about uncertainty eradication present in SE. In fact, HCI literature has some examples of insight regarding *real* software development. Hix & Hartson's [10] observations about the work of software developers show that they usually operate in alternating waves of two complementary types of activities: both bottom-up, creative activities (a synthesis mode) and top-down, abstracting ones (an analysis mode). Hix & Hartson also unveil the closeness that exists between analysis and design activity types, especially in the requirements-related activities. It is not sensible, then, to try to draw a clear separation between both activity types. With regard to methodologies in place in software development companies (based in a waterfall approach), they report that in some of their empirical studies they noticed that "iterative and alternating development activities occurred, but because corporate standards required it, they reported their work as having been done strictly top-down" [10]. The reality of development was hidden behind the mask of order of the waterfall.

According to Hakiel "There is no reason why a design idea might not survive from its original appearance in requirements elicitation, through high- and low-level design and into the final product, but its right to do so must be questioned at every step" [9]. This approach is a radical separation from the waterfall mindset mostly present in SE, which was traditionally presented as *the way* to develop software in an orderly manner.

The multidisciplinary essence of HCI has helped in providing a not so rigid approach to development in the field. As Gould and Lewis [12] say, when a human user is considered (as in the upper part of Fig. 1) a coprocessor of largely unpredictable behavior has been added. Uncertainty is a companion of any attempt to develop interactive systems of non-trivial complexity, since human beings are part of the supra-system we are addressing: the combination of the user and the software system, trying to perform tasks which directly relate to the user goals.

User-centered or human-centered development is the HCI approach to the development process, and it has traditionally introduced uncertainty when labelling himself as iterative. In this sense, [5], [10], [16], [22], and [26] agree on considering iterative development as a must for a user-centred development process. Therefore,

iterativeness is at the core of HCI practices. A real iterativity, in the sense that evaluation is often considered as formative; not just an exam for identifying mistakes, but a tool for giving form to the interaction design, and maybe for identifying new requirements.

4 Common HCI-SE Problem-Solving Strategies

As presented in the previous section, uncertainty is a tool for problem resolution; in particular, it is a tool for interactive software development. Uncertainty-based approaches have been adopted in the resolution strategies of both HCI and SE, without labeling them as such.

When trying to integrate usability and HCI methods into mainstream development, the extensive HCI experience in dealing with uncertainty may be incorporated into SE practices, making them better prepared to cope with the development of complex systems with a high usability level.

Non-linear problem-solving strategies present important challenges with respect to estimation and planning, along with the danger of continuously iterating without advancing towards the solution. A certain degree of flexibility is necessary for dealing with these issues, as HCI usually employs. Accordingly, some degree of uncertainty will have to be introduced in the formal procedures advocated by SE methodologies.

4.1 Iterative Development

Iterative-cyclical strategies are currently at the center of debate in SE, with agile and iterative practices (see, for example, [18]). When adopting cyclical strategies of this kind, the introduction of HCI practices may be undertaken with greater success than former proposals for integration into waterfall lifecycles, like [7].

The aim of integrating usability engineering and HCI practices into mainstream software development, which mostly refuses to deal with uncertainty, have led to more formal solutions, in a SE sense, but leaving out the uncertainty present, for example, in iterative approaches. Such as Mayhew's Usability Engineering Lifecycle [20], which is based on a two-step process where analysis activities are performed in a first phase, and then design and evaluation activities are performed iteratively on a second phase; but there is no place for resuming analysis activities. Therefore, it is based on a frozen requirements paradigm, with reminiscences of a waterfall mindset.

Nevertheless, iterativeness has been at the heart of usability engineering practices because usability testing has been the central point around which the whole development effort turns. It is necessary to test any best-first-guess design. Observational techniques and sound analysis are performed with the aim of getting a high quality first design, but usability testing with representative users is then performed to check against reality the logical constructs the design is made of.

The expected functionality and quality levels of the final system can be specified, but there is a certain degree of uncertainty in building the solution, the software system, in the sense that when undertaking the construction of some part of the system we do not exactly know how far we are from the specified solution. This is especially true when dealing with usability. Any design decision directed to usability

improvement needs to be tested with representative users, in order to check the actual improvement in usability attributes like efficiency in use. When the system under scrutiny includes the final user on top of the computer system, as it is necessary for the management of the final product usability, flexibility is required for adapting the partial prototypes according to evaluation outcomes.

4.2 Exploratory Strategies and the Definition of the Product Concept

Exploratory strategies are not yet dealt with in SE literature and practice. Traditional information systems, like payroll systems, are well defined and most SE methodologies are directed to building them. Input-process-output models fit very well this kind of problems: automation of procedures previously performed manually, with well defined rules and algorithms. The product concept is clearly delimited in this kind of systems, so requirements can be written down with less risk of misunderstandings between the customer and the development team. Actually, IEEE body of standards has a standard for establishing the user requirements, the Concept of Operations [15] or ConOps, but it is seldom used in software development, unlike the more system-oriented (or developer-oriented) IEEE recommended practice for software requirements specification [14], which receives much more attention from the SE field.

On the other hand, the HCI field has a long tradition of dealing with ill-defined problems, developing new products with a high degree of innovation. Even if the creation of these systems has not been their main focus of activity, dealing with problems with neither an obvious solution nor indications of how development should proceed, has been part of HCI practitioners' work. Accordingly, several HCI techniques are specially well suited for defining the product concept. These techniques favor participative and creative activities, which fit very well the purpose of creating a model of how the system works, from the user point of view, studying if it fits with user needs/expectations. Examples of this techniques are Personas [6], Scenarios [4], Storyboards and Visual Brainstorming [22].

As long as current interactive systems development goes on changing to new paradigms of interaction, with an ever increasing degree of novelty required, these HCI techniques will have to be either adopted by software engineers, or applied by HCI experts belonging to the development team.

5 Conclusions

In this paper we have shown how uncertainty plays a major role in software development in the construction of non-trivial interactive software systems. While uncertainty in the problem may be harmful, uncertainty in the solution may be useful when used as a tool for dealing with the former kind of uncertainty (the one in the problem).

HCI has been traditionally applying flexible processes that allow participatory design, and it has promoted the usage of prototypes aiming at greater flexibility for making changes to the (partial) software solution. Aditionally, some HCI techniques are especially well suited for the development of innovative software systems, which

are ill-defined by definition, and they may be adopted for exploratory problem-solving strategies. Even if this is part of standard HCI practices, the convenience of this approach has not been formalized in a way that helps HCI methods integration into mainstream software development practices.

Recent awareness about the obstacles that traditional approaches, like the waterfall life cycle, imposes on the endeavor of successful systems development, leads to a more favorable attitude to the introduction of HCI methods, which ultimately lead to better quality products. In particular, HCI may play an important role in introducing practices that improve the usability of the final product, while additionally preparing businesses to better deal with uncertainty in software development.

Understanding the roots of current software development practices and knowing their deficiencies in dealing with uncertainty is essential for any software development business. A model for software development that considers uncertainty is needed, in order to change from a field that is based only on the expertise of gurus to a software development field with sound foundations for the selection of development practices.

References

1. Blum, B.I.: Software Engineering A Holistic View. Oxford University Press, New York, USA (1992)
2. Boehm, B.W.: A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Engineering Notes 11-4, 14–24 (1986)
3. Bourque, P., Dupuis, R., Abran, A., Moore, J.W., Tripp, L., Wolf, S.: Fundamental principles of software engineering- a journey. Journal of Systems and Software 62, 59–70 (2002)
4. Carroll, J.M.: Scenario-Based Design. In: Helander, M., Landauer, T., Prabhu, P. (eds.) Handbook of Human-Computer Interaction, 2nd edn. pp. 383–406. Elsevier, North-Holland (1997)
5. Constantine, L.L., Lockwood, L.A.D.: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley, New York, USA (1999)
6. Cooper, A., Reimann, R.: About Face 2.0: The Essentials of Interaction Design. Wiley Publishing, Indianapolis (IN), USA (2003)
7. Costabile, M.F.: Usability in the Software Life Cycle. In: Chang, S.K. (ed.): Handbook of Software Engineering and Knowledge Engineering, pp. 179–192. World Scientific, New Jersey, USA (2001)
8. Descartes, R.: Discourse on the Method of Rightly Conducting One's Reason and of Seeking Truth (1993), http://www.gutenberg.org/etext/59
9. Hakiel, S.: Delivering Ease of Use. Computing and Control Engineering Journal 8-2, 81–87 (1997)
10. Hix, D., Hartson, H.R.: Developing User Interfaces: Ensuring Usability Through Product and Process. John Wiley and Sons, New York (NY), USA (1993)
11. Glass, R.L.: Facts and Fallacies of Software Engineering. Addison-Wesley, Boston (MA), USA (2003)
12. Gould, J.D., Lewis, C.: Designing for Usability: Key Principles and What Designers Think, Communications of the ACM, 300–311 (March 1985)
13. IEEE: IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE, New York (NY), USA (1990)

14. IEEE: IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. IEEE, New York (NY), USA (1998)
15. IEEE: IEEE Std 1362-1998. IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document. IEEE, New York (NY), USA (1998)
16. ISO: International Standard: Human-Centered Design Processes for Interactive Systems, ISO Standard 13407: 1999. ISO, Geneva, Switzerland (1999)
17. Klir, G.J., Folger, T.A.: Fuzzy Sets, Uncertainty and Information. Prentice Hall, N.J. (1988)
18. Larman, C.: Agile and Iterative Development. In: A Manager's Guide, Addison-Wesley, Boston (MA), USA (2004)
19. Matsubara, T., Ebert, C.: Benefits and Applications of Cross-Pollination. IEEE Software. 24–26 (2000)
20. Mayhew, D.J.: The Usability Engineering Lifecycle. Morgan Kaufmann, San Francisco (CA), USA (1999)
21. Parnas, D.L.: On the Criteria To Be Used in decomposing System into Modules. Communications of the ACM. 15-12, 1053–1058 (1972)
22. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T.: Human-Computer Interaction. Addison Wesley, Harlow, England (1994)
23. Raccoon, L.B.S.: The Chaos Strategy. ACM SIGSOFT Software Engineering Notes, 20-5, 40–46 (1995)
24. Seffah, A., Andreevskaia, A.: Empowering Software Engineers in Human-Centered Design. In: Proc. of the ICSE'03 Conference, Portland (OR), USA, pp. 653–658 (2003)
25. Seffah, A., Gulliksen, J., Desmarais, M.D. (eds.): Human-Centered Software Engineering - Integrating Usability in the Development Process. Springer, Heidelberg (2005)
26. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction, 3rd edn. Addison-Wesley, Reading (MA), USA (1998)
27. Vredenburg, K., Mao, J.Y., Smith, P.W., Carey, T.: A Survey of User-Centered Design Practice. In: Proc. of CHI-2002, Minneaopolis (MI), USA, pp. 471–478 (2002)