A Semi-Formal Framework for Describing Interaction Design Spaces

Judy Bowen
The University of Waikato
New Zealand
jbowen@waikato.ac.nz

Anke Dittmar
University of Rostock
Germany
anke.dittmar@uni-rostock.de

ABSTRACT

Interactive system design is typically more successful if it is an iterative process involving collaboration between multi-disciplinary teams with different viewpoints. While some sub-teams may focus on the creative aspects of the user interface design and other subgroups on the implementation of required functionality, all must ensure that they are working towards the same goal. They must also satisfy the requirements and needs of all stakeholders. Although many suggestions have been made as to how such design might be supported in a more formal way (such as by using a model-driven process), less focus has been given to managing the co-ordination of design sub-teams following a creative process. In this paper we propose a semi-formal framework to describe and to compare design spaces, and the external design representations within those spaces. The framework is based on ideas from interaction design and on formal refinement approaches. It suggests a distinction of design options into alternatives and variants to describe and guide processes of idea generation and convergence within, and between, different design sub-spaces and sub-groups. We provide a small example to illustrate our approach and to show how it can be implemented by using standard formal approaches alongside less formal design notations and human-computer interaction processes.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*theory and methods.*; D.2.4 [**Software Engineering**]: Formal Methods.

Keywords

Interaction design; design spaces; refinement; formal methods; user-centred design.

1. INTRODUCTION

The field of human-computer interaction (HCI) accommodates diverse approaches or schools of thought to interactive system design, each with their own justification and their own focus of attention. In user-centred design (UCD), for example, the design process is depicted as an iterative process with some form of user and other stakeholder involvement. External design representations such as scenarios and prototypes are seen as essential means to establish a shared understanding among participants during the design process [17]. Design representations are also studied in interaction design research, but with more interest on their role in the designers' exploration of design spaces and in their decision making [16]. Engineering approaches to HCI, in contrast, put much emphasis on technical aspects, e.g., on a separation between the user interface (UI) part and the application part of the interactive artifact. Here, formal models such as task and UI models and systematic transformation and refinement steps are employed to ensure desired properties of the system under consideration [6]. Although the above mentioned, and other, approaches generally acknowledge that interactive system design requires different viewpoints and an elaborated division of labour, there are gaps or even tensions between them due to their specific goals, assumptions, activities, resources, and outcomes. For example, Dix et al. [12] point out that "the ideal model of iterative design, in which a rapid prototype is designed, evaluated and modified until the best possible design is achieved... is appealing" but that it is also important to use "more principled approaches" or formal techniques to be able to overcome bad initial design decisions and understand the reasons behind usability problems, not just detect the symptoms.

In this paper, we are interested in developing a better understanding of the interplay of such heterogeneous design practices for successful interaction design. The starting point is the assumption that however intangible a design process might be there always emerges at least a minimum set of requirements (expressed in some way) which must be satisfied by the final design. Therefore, the collaborative and iterative design process must not only acknowledge different design practices resulting in diverse design representations ranging from UI sketches to formal system models, but it must also ensure that where design decisions are made a) they are consistent with existing requirements or satisfy new constraints, and b) their why/when/what is recorded for future reference or backtracking. What we consider is the designers' ability to compare different design representations and understand how they are related and whether or not they satisfy some initial or evolving design specifications.

Our work is guided by the concept of design space as it is used in interaction design. It describes the designers' moves as intertwined steps of problem setting and problem solving. Design is seen as a goal-directed exploration of emerging, and often externally represented, options and constraints. It is an iterative generation and convergence of concepts or design ideas [5]. At the same time, the presented work is anchored in refinement approaches from formal approaches to HCI and from computer science in general [27]. As a

EICS'16, June 21-24, 2016, Brussels, Belgium
© 2016 ACM. ISBN 978-1-4503-4322-0/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2933242.2933247

result, the paper provides a semi-formal framework that extends the common ideas of interaction design spaces with a more elaborated description of their structure, the relationships between external design representations, and the nature of design options and decision making. In particular, the new framework takes into consideration the collaboration of heterogeneous sub-teams who populate design sub-spaces. Participants in such complex design spaces are engaged in multiple designer-user relationships when it comes to the creation and use of the different design representations. Sub-teams are provided with various design representations from outside their sub-space which inform their local activities and which they partly have to refine. They in turn share their results with others to achieve the overall design goals.

The framework suggests a distinction between *alternatives* and *variants* as two different types of design options. Basically, design options that are considered within a design sub-space are called alternatives if a decision is locally made to keep only one of them in the further design process and share it with others. In contrast, a set of design options are called variants if they refine the same design concept and if they are kept in the design process by providing an adequate representation to other sub-spaces. Alternatives and variants are an important means to balance idea generation and convergence within and across design sub-spaces. They help to support different viewpoints and to avoid premature commitments. To give a simple example, UI designers may provide a user interface which offers different ways (variants) to perform a certain task and it is the user who decides in a specific task situation which of the possible execution paths to follow.

The paper starts with a background and related work section before introducing the general ideas of the framework for interaction design spaces and illustrating them with a more substantial example design scenario. The semi-formal description of the framework describes the ideas more precisely. The concepts presented, such as refinement relationships between external design representations and alternatives and variants acknowledge both the local work of design sub-teams and their collaboration. They thus make possible a more effective integration of the designers' different expertise and contributions to a design project. The framework particularly aims at supporting a better integration of formal HCI methods and more informal design approaches. It is suggested in the discussion part that the framework helps to reflect upon existing design practices or to plan future ones. The paper closes with some conclusions and with a brief outline of future work.

2. BACKGROUND AND RELATED WORK

In this section, we briefly review existing concepts of design spaces that help explain the nature of design activities and their outcomes. We discuss the role of external design representations in collaborative settings and we examine the concept of refinement as a means to relate such design representations.

2.1 Design Spaces

There are different views on the concept of design space. In the literature on engineering interactive systems and UI design, a design space is often understood as being defined along a set of dimensions. For example, Nigay and Coutaz [23] suggest a design space for multi-modal systems in terms of level of abstraction, use of modalities and fusion. Such orthogonal dimensions and corresponding values identify potential design constraints and provide classifications of particular interactive systems (e.g., a classification of multi-modal systems in [23] or of ephemeral user interfaces in [14]). Design spaces, in this sense, provide a vocabulary for characterising (certain aspects of) systems. They are tools for choosing the right design in a specific design context.

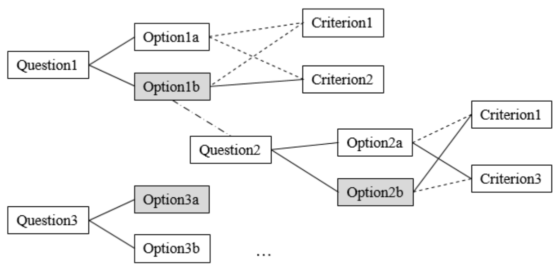

Figure 1: Representation of a design space as QOC-diagram. The gray boxes refer to the chosen solution with each such box standing for the selected option for one design question.

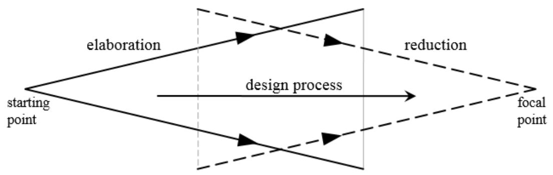

Figure 2: The designer's moves in a design space: Laseau's overlapping funnels.

Other, and in the context of this paper more relevant, approaches understand design spaces as related to specific design problems or projects. For MacLean et al. [20], a design space is "an explicit representation of alternative design options, and the reasons for choosing among those options". The authors point out that "HCI design is much more informal, more open-ended and more poorly understood than some other domains". Therefore, they and other advocates of design rationale suggest that the result of a design process should be conceived as a design space rather than a single specification or product. The QOC-notation [21] is one of the most popular notations for design space exploration and supports a structured representation of design ideas (Options) along design Questions. Criteria help to assess the pros and cons of each option. Figure 1 shows a schematic QOC-diagram.

According to Buxton [5], designers need to bring creativity to both the creation of distinct options and the definition of criteria or heuristics to choose between those options. While design representations such as QOC-diagrams depict how options and criteria are related to each other they do not explicitly represent the process of their creation and use. The funnel model of Figure 2 (e.g., in [5]) puts more emphasis on the designer's goal-directed actions. The generation of design options (concept generation) is represented by the elaboration funnel and has to be in balance with decision making (concept convergence) in the reduction funnel. The chosen design solution is represented by the focal point. Refined versions of the funnel model assume some front-end work resulting in a product design specification. This is the starting point for an alternation between concept generation and concept convergence step-wise leading to finer levels of granularity in the design (see [5] for an overview). Westerlund [26] points out, though, that "the initial brief, assignment or problem that is one constraint on the design space will not be stable during the process". He also criticises design space models using the funnel-metaphor for their focus on one goal and one final solution, which may impede a diversity of design ideas.

Although the above brief review shows that there is no common definition of design spaces it reveals a shared understanding of it as a conceptual tool to understand and guide design activities. Some authors still distinguish between problem space and solution space. However, it is generally acknowledged that design problems are typically 'wicked' problems [25] requiring an intertwined 'problem setting' and 'problem solving' and that external representations are of importance in such creative processes. Therefore, similar to other approaches, no such distinction is made in this paper.

2.2 External Design Representations

External representations are ubiquitous in interaction design [13]. Gaver [16] describes the dual function of design representations (referred to by him as proposals) as follows. "Proposals may vary widely in their specificity, from evocative and unrealisable sketches, to abstract representations of intention, to relatively complete specifications or scenarios. In each case, the role of design proposals is both to create and constrain." Westerlund [26] even considers a design space as the set of "all possible design proposals that would be regarded as meaningful to use by some people in relevant contexts". In his view, proposals that work lie within the design space, proposals that do not work are outside the design space. Designers externalise ideas to further shape or discard them, but externalisation also enables designers to work collaboratively. Design representations can be shared, negotiated, and agreed on [15], and then they can serve as a brief for further elaboration and refinement of what the interactive system under design will and will not be [16]. Two aspects are discussed in more detail below: first, multi-disciplinary design teams and their use of design representations, and second, implications for the refinement of design proposals until the final product is reached.

2.2.1 Multi-disciplinary Design Teams

Designing interactive software requires knowledge of the application domain and expertise in different areas of psychology, software engineering and many other fields. Typically, multi-disciplinary teams considering a design from diverse viewpoints come up with more successful solutions [19]. Of course, such diversity can also result in 'culture clashes' due to different professional backgrounds [2]. Mackay [19] states, for example, that "scientists are trained to seek explanations of existing phenomena, engineers are trained to provide technical solutions to well-defined problems, and designers are trained to explore a design space and find solutions that 'work'." The author recommends complementing educational programs in single disciplines by instructional formats that support an understanding and appreciation of other disciplines and discuss possible interactions with them [19]. Bellotti et al. [2] argue that for an effective collaboration, a revision of each others' assumptions can be necessary. As an example, they refer to the conventional notion in the software engineering community that "formal methods are only useful if used within a structured development context from the beginning of a project, through refinement, to implementation". Different views of formal methods, and more specifically refinement, are discussed below.

Design representations for multi-disciplinary work need to support the creation of a shared understanding among all collaborators[1], but at the same time they have to serve the local needs of specialised sub-teams. One way to increase each others' receptivity is to develop methods that 'couple' models and representations from different approaches such as task models and UI models in model-based design [6, 24] or formal modeling and prototyping [9]. However, the co-evolution of different types of design representations that is essential to interleave the activities of heterogeneous design sub-teams in an effective way is still poorly understood [10]. In this paper, we use a relaxed version of refinement to relate design representations and to support the idea of design as a goal-directed activity in the sense that however intangible a design process might be there always emerges a set of requirements which is expressed in some way and must be satisfied. Additionally, our framework of interaction design spaces borrows inspiration from Morgan's idea of client-programmer relationships [22] mentioned in the next sub-section to describe collaboration at a sufficiently abstract level.

2.2.2 Refinement

The concept of refinement is central to many formal software engineering development methods. At its simplest it describes the transformation from initial specification to final implementation. Formally this requires a specification language with associated refinement theory or algebra which is used to describe how a less concrete specification can be transformed into something closer to an implemented system. This is repeated until finally the implementation is reached.

Different types of refinement can be used depending on the nature of the transformations required, e.g. data refinement, operation refinement, trace refinement etc. In the example we give here for our framework we will focus on data refinement, specifically step-wise refinement, where a relation (often referred to as a 'retrieve relation' or 'abstraction relation') can be used to define how the abstract and the more concrete specification are related to each other at each step, where each step is an allowable operation of the system being specified. The algebra ensures that properties that are true of the specification remain true in the implementation and so provide the structure for guaranteeing correctness throughout the development lifecycle, see [27, 8] for more detailed explanations.

The idea of refinement has been generalised leading to more liberal approaches to be used over a wider set of problems. Morgan [22] suggests banishing the distinction between specifications, sub-specifications and programs and considering all of them as contracts which have to be negotiated between clients and programmers. "A program has two roles: it describes what one person wants, and what another person (or computer [if the program is executable code]) must do" [22]. Refinement, in Morgan's understanding, is about maintaining utility rather than hiding substitutions of abstract specifications of a system's functions and behaviour by more concrete ones: the client gets at least what they had before or even better. According to this view, transformations are considered in terms of levels of abstraction, removal of nondeterminism, concretisation of data types etc. A similar, less formal approach has been taken more recently in [3]. Here, Bowen and Reeves apply a lightweight notion of refinement to UI design by relating informal representations such as UI sketches and UI prototypes to formal models of the overall system to ensure that, in a process based on division of labour, UI and system designers are working towards the same end goal. We are similarly interested in integrating different design practices.

What we will describe is not refinement in its traditional sense, but the ability to consider different design representations which may emerge from a design process and understand how they are, and need to be, related to each other. In this paper we take some of the key formal ideas - such as data refinement and retrieve relations, and frame them within a design environment. We are not so much focussed on ensuring correctness of an implementation in terms of its behaviours, but rather an adherence to design and functional requirements in an evolutionary and collaborative design space. We

[1]It should be emphasised, though, that there is no need for heterogeneous design sub-teams to find full consensus.

discuss this further later.

3. THE FRAMEWORK

We first describe the objectives of the suggested framework, introduce the key ideas and concepts, and illustrate them with an example design scenario. The semi-formal description that follows states the ideas more precisely and thus provides more opportunity for reflecting upon them.

3.1 Objectives

This paper aims at better understanding the interplay of heterogeneous design practices for successful interaction design. The concept of design space is taken as a starting point since it explains design as a goal-directed activity but with interleaved problem setting and problem solving. The creation of design ideas and the definition of appropriate criteria to assess them are equally valued for finding a good solution. The objective of the framework is to enrich this understanding of design spaces by a more detailed consideration of some of the above discussed points. This gives rise to the following questions:

- While the mediating role of external design representations is generally acknowledged, current design space approaches put less emphasis on the question of how representations in an iterative design process have to be related to ensure that the final representation or product has required properties and the overall design goal has been achieved. We have seen that formal approaches to software engineering and interaction design provide methods for a stepwise refinement of models. How can we integrate such refinement ideas into a collaborative design process with multiple viewpoints?
- Few design space approaches explicitly consider consequences of division of labour in multi-disciplinary work. How can we support specialised and collaborative work, reconcile partly heterogeneous practices, and make competent decision making possible in a distributed process of idea generation and convergence within and across sub-teams?

3.2 Key Ideas

Central elements of a design space are external design representations, in whatever form they appear (written specifications, UI sketches, prototypes, task models, user models, QOC diagrams etc.). In what follows, we briefly refer to all such representations as *designs*.

3.2.1 User-Designer Relationship

Each design space has an entry point and an exit point indicating the underlying user-designer relationship. Figure 3 illustrates a single design space with designs depicted by ellipses. Via the entry point, the user provides the designer with some designs representing requirements. The designer in turn is expected to provide designs at the exit point which somehow satisfy those requirements. The designer's activities within the design space result in new (local) designs which they can relate to each other but also discard again. We introduce the notion of a 'valid' design space where everything contained within it (all possible designs) could lead to a satisfactory final design. These may be partial designs, in that they only consider some parts of a solution, in which case they do not satisfy *all* requirements, but rather they do not break any.

3.2.2 Alternatives and Variants

Designs within the design space may be independent options satisfying the same requirements but in different ways. They may also be assessments of such options supporting the designer's decision making. A set of options is called *alternatives* if the designer is able to make a decision and only one of them leaves the design space as the currently selected one at the exit point. Options are called *variants* if all of them are provided to the user to let them decide which one to apply in a specific situation. In other words, a single solution at the exit point represents a closed process of generating possible solutions and choosing a good one while variants stand for a somewhat open decision process. In the latter case, the designer provides options to the user which share some common elements, but not all, to satisfy the requirements. However, the designer is aware that it is beyond their competency to make a selection or that a selection would unnecessarily limit the user's activities, including their creativity. This awareness is especially important in multi-disciplinary work as discussed with the introduction of complex design spaces below.

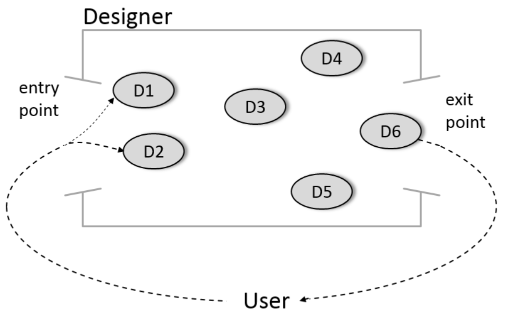

Figure 3: User-designer relationship and (unrelated) designs in a single design space.

3.2.3 Relating Designs: Creativity, Design Goals and Iterations

There is an implication in Figure 3 that we move from an entry point on the left towards the exit point on the right, and that designs exiting are more 'refined' (closer to an implementable solution) than those at the entry point. However, this does not mean that within the design space we work from left to right. Within the space the designers move back and forth in levels of abstraction and may experiment with different (perhaps conflicting) ideas before finalising some elements and leaving the design space. When we talk about 'designing' we mean all, and any, activities that may occur as part of the creative process of transforming an initial idea into an implementable solution. In the context of this paper, we are most interested in how designers relate designs in the design space to ensure that there is a 'chain' of refined designs from the entry point to the exit point to ensure desired properties of the final result (i.e., to achieve the design goals). Iterations are necessary if no refined design can be found or agreed on (a perhaps more usual than unusual situation). Iterations in this sense require some involvement of the user and often a revision of the designs in the entry point. Such 'cycles' are indicated in Figure 3.

3.2.4 Types of Refinement

We talk about 'refinement' in this paper in the most general sense. That is, the concretisation of any form of designs in the direction of an implementation. We therefore consider some of the principles of refinement (removal of non-determinism, principle of substitutivity, satisfaction of expressed requirements and relations between designs) as *guides* within the design process or as a means to compare designs and understand their differences. In order to consider any sort of relationship between different designs we do, of course, need a vocabulary to do so, as well as an understanding of what properties we are interested in which will form that relation. Three types of refinement are suggested.

1. Refinements based on formal methods to ensure that we build the system in the right way (correctness).
2. Lightweight notions of refinement for a transition between informal and formal designs.
3. Refinements that are based on validation techniques to ensure that we build the right system.

'Traditional' formal methods make refinements of the first type possible but work on formal system specifications only. Refinements of the second type support an integration of informal and formal interaction design approaches to mitigate their disadvantages. On the one hand, formal methods are often perceived as limiting the creativity of interaction designers, on the other hand, informal approaches may lack the required discipline to provide quality design [11]. An example of a lightweight refinement mechanism is given in [3] which allows UI designers to retain valued techniques such as UI sketching and prototyping whilst providing a formal underpinning. Rittel [25] and others recommend embedding formal design methods ('first generation' methods) into what they call 'second generation' methods to validate a design. Second generation methods for refinements of the third type include design rationale methods that help the understanding of a design problem from multiple viewpoints and the finding of a good solution by comparing different possibilities. Refined designs could be, for example, QOC-diagrams [21] (see background section) or Softgoal Interdependency Graphs (SIG) [7] which not only show the transformation of a set of non-functional requirements into functional requirements but also the argumentation process behind it. We later describe some of the refinement methods in more detail while applying them in our example design scenario.

3.2.5 *Complex Design Spaces*

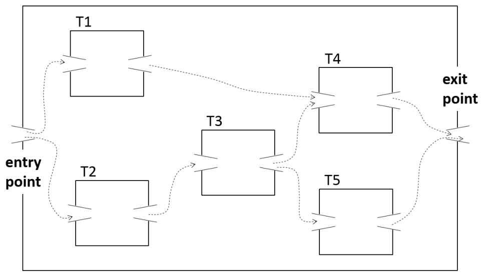

Figure 4: Complex design space with sub-spaces.

So far, we have considered simple user-designer relationships and our model of a single design space may be sufficient here, but not for the description of collaborative design processes. In multi-disciplinary work, different sub-teams may be working relatively independently on different elements of the problem before combining their results together at a later time. In other words, there is no one monolithic design space but rather it consists of sub-spaces belonging to the sub-teams. A current prototype, for example, may be the output from one design sub-space and become the input to a different one together with other designs provided by other sub-teams. Figure 4 illustrates this situation. T1 - T5 are all design sub-spaces (such as that shown in Figure 3) which are linked in several ways. Firstly they are all part of a larger design space and as such represent designs for the same set of initial requirements (or evolving requirements due to iterations), secondly designs enter and exit these sub-spaces within the larger design space. As such there is a recursive nature to the approach where within T1 - T5 there may be further design spaces, and so on.

Our model of complex design spaces takes into consideration that single design sub-spaces, and hence sub-teams, will not exist for a whole development life-cycle and that a design process typically is characterised by a complex network of user-designer relationships. One person can be active in different groups and play the role of user in some design contexts and the role of designer in others. In participatory design, for example, 'users' (that is people who use the end product) are actively involved in the design process and take over designer roles as well. As another example, task-modelers in a task-based design approach may be in a user-designer relationship with UI-designers expecting them to consider their task descriptions in the UI-designs.

Another consequence of multi-disciplinary collaboration that becomes more visible with complex design spaces is that sub-teams must be aware of both their local design goals and the goals of the overall design process. Iterations can take place at different levels of the hierarchy of design sub-spaces and need to result in the revision of corresponding designs. Specialised sub-teams must be able to distinguish between decisions that should be made by themselves within their local design space to reduce complexity in a reasonable way and decisions which should be left to others because their viewpoints and ideas might be important. In other words, sub-teams must be able to recognise alternatives and variants in order to properly distribute the generation of ideas and to inform their assessment and decision making within and across design sub-spaces. We will illustrate this in the example design scenario below.

3.3 Illustration of the Framework: An Example Scenario

Here we introduce an example design problem and show how the framework we have presented may be used to support the design. The example is that of implementing a graphical tool for UML class diagrams, such as might be used by computer science students learning UML. From initial discussions with potential end-users of such a tool we elicit the following requirements:

- **D1:** The tool should support creating and editing of a restricted set of UML class diagrams. Users should be able to add and remove classes which are characterised by unique names. Users should also be able to add and remove inheritance relationships between classes.
- **D2:** The tool should support common modelling strategies of the users.

For reasons of brevity we deal only with the tool's add-functions in the rest of this example but readers should assume the corresponding delete-functions are similarly described and included. Figure 5 illustrates the design sub-spaces for our design scenario with the sub-teams and their goals. The design team, *T* is split into 2 sub-teams, *T1* and *T2*. Sub-team *T2* is similarly split into two further sub-teams, *T21* and *T22*. In the framework, collaboration within and between different sub-groups is understood by looking at how designs are distributed and refined. In the following, we discuss the creation and use of the designs indicated in Figure 5. The designs in the entry and exit points of the sub-spaces are depicted by gray ellipses, those created within sub-spaces are colored in light gray.

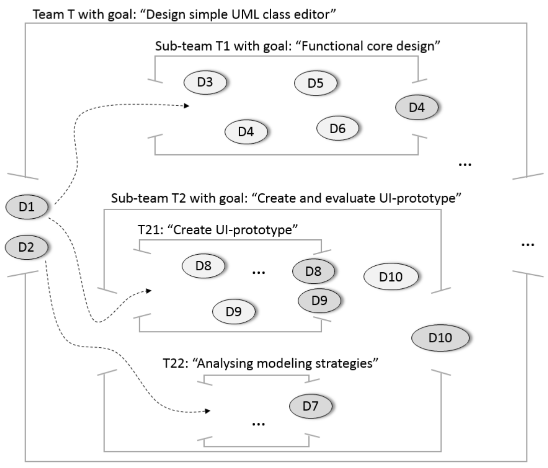
Team T with goal: “Design simple UML class editor”
Sub-team T1 with goal: “Functional core design”
D3
D5
D4
D4
D6
...
D1
D2
Sub-team T2 with goal: “Create and evaluate UI-prototype”
T21: “Create UI-prototype”
D8
...
D8
D10
D9
D9
D10
...
T22: “Analysing modeling strategies”
D7
...

Figure 5: Design teams and sub-spaces in the scenario.

3.3.1 Sub-space T1 “Functional core design”

T1 are responsible for formally specifying the proposed system. As we can see from Figure 5, their starting point for this is **D1**. They begin by considering the system observations as a set of classes which can be related in a superclass/subclass hierarchy. The key components of this specification, **D3**, are given below using the Z specification language [1]:

UMLModel

$$\begin{array}{l} class : \mathbb{P}\, CLASS \\ superclass : CLASS \leftrightarrow CLASS \end{array}$$

AddClass

$$\begin{array}{l} \Delta UMLModel \\ i? : CLASS \\ \hline i? \notin class \\ class' = class \cup \{i?\} \\ superclass' = superclass \end{array}$$

AddInheritanceRelation

$$\begin{array}{l} \Delta UMLModel \\ super? : CLASS \\ sub? : CLASS \\ \hline super? \in class \wedge sub? \in class \\ super? \neq sub? \\ superclass' = superclass \cup \{super? \mapsto sub?\} \\ class' = class \end{array}$$

First a type called *CLASS* is given, then the system is defined by an observation called *class*, which is any element of the powerset for this type, and another called *superclass* which is a relation between classes. The *AddClass* operation allows for a new class to be added if it does not already exist and the *AddInheritanceRelation* operation enables two distinct classes to be put into a super/sub class relationship.

Subsequently *T1* produce a second specification **D4**. They add operations to enable the identification of superclasses as well as to consider whether or not a model is valid. This states that if a class is the superclass in a hierarchical relationship with another class, then it cannot also be the subclass of that same class. This is added as a validity check rather than a constraint to enable users of the system to model in a manner that they think is correct and then find out if indeed they are correct, rather than blocking them by disabling invalid operations.

The new specification, **D4**, has a different system description which no longer considers *classes* as distinct entities, but rather as part of a relation of super/sub classes.

UMLModel

$$superclass : CLASS \leftrightarrow \mathbb{P}\, CLASS$$

Operations *AddClass* and *AddInheritanceRelation* are changed to reflect this.

T1 also define two new operations in **D4**: *isSuperClass* which is used by *IsValidUMLModel*, e.g.

isValidUMLModel

$$\begin{array}{l} \Xi UMLModel \\ valid! : BOOL \\ \hline valid! = \forall c1, c2 \in \operatorname{dom} superclass \wedge \\ \qquad isSuperClass(c1, c2) \bullet isSuperClass(c2, c1) = false \end{array}$$

D3 follows the initial requirements given in **D1**. *T1* could prove this formally using model-checking (over a restricted set of class names) to show that new classes can be added but only names that are unique are permitted (in fact we can see from observation alone that the predicate *i? ∉ class* in the *AddClass* operation meets this condition). Similarly, *T1* can show that inheritance relations can be added. Although **D4** introduced differences from **D3**, it is a data refinement of **D3** and this can be shown formally using the stepwise refinement approach described earlier. A retrieve relation is created between the *UMLModel* schemas of **D3** and **D4** and *T1* show that this relation continues to hold after each of the operations: *AddClass* and *AddInheritanceRelation*. The relation describes how every element in the *class* set of **D3** is in the domain of the *superclass* relation of **D4**, and that for every pair in **D3**'s *superclass* relation there is a corresponding pair that can be extracted from the *superclass* relation of **D4**.

Consider the case where three classes: ‘A’, ‘B’ and ‘C’ have been added where ‘A’ is the superclass of ‘B’ and ‘C’. In **D3** this is represented by the *class* observation having the value *{A, B, C}* and the *superclass* observation having the value *{A ↦ B, A ↦ C}*. In **D4** the single *superclass* observation has the value *{A ↦ {B,C}, B ↦ ∅, C ↦ ∅ }*. Once this relation is established sub-team *T1* prove the refinement by showing that if the relation holds after initialisation (as described above) then it continues to hold after the *AddClass* or *AddInheritanceRelation* operations occur. So, if a new class ‘D’ is added, using **D3**'s operation the *class* observation becomes *{A, B, C, D}* and the *superclass* observation remains unchanged. Using **D4**'s operation the *superclass* observation becomes *{A ↦ {B,C}, B ↦ ∅, C ↦ ∅, D ↦ ∅ }*.

For this small example we can ‘see’ that the retrieve relation continues to hold after the *AddClass* operation and can do the same for *AddInheritanceRelation*. Usually we are dealing with more complex data types and operations and *T1* would need to perform a formal proof of this, but inclusion of this is beyond the scope of this paper. The property is trivially true for the *isValidUMLModel* operation as the Ξ before *UMLModel* means that all observations of the system remain unchanged (we often refer to such an operation as *skip*). As these properties do then hold for **D3** and **D4** then the refinement likewise holds.

The team *T1* may continue developing other ideas (alternatives) to refine **D3**, for example by abstracting both classes and superclasses to a single entity (an observation called 'elements') to allow for an "Add" operation that can be used to either add a class or an inheritance relation (design **D5**). Or, they may instead have a design **D6** where an inheritance relation can be added where only the superclass already exists and the subclass is then automatically created as part of the operation. In order to prove refinement now, *T1* must consider more than just data refinement (as there are parameterised operations), however the principles remain the same. That is, using one of the existing refinement approaches for Z they continue to prove that the refinement holds between these different designs.

Eventually sub-team *T1* discuss pros and cons of the alternatives **D4** - **D6** and agree on **D4** to leave the exit point of their sub-space (see Figure 5). In other words, *T1* considered the three options as *alternative* refinements of **D3** among which they have to choose without the need to involve the whole team *T*.

3.3.2 Sub-space T2 "Create and evaluate UI-prototype"

Meanwhile, *T2* are responsible for the UI prototype. Specifically sub-team *T21* have the goal to create a UI-prototype and sub-team *T22* have to investigate UML modelling strategies which should be supported by the UI. They found two ways a user might choose to build their UML class diagram: A) create classes and then link them into hierarchies, and B) create a parent class and then create all of its subclasses. *T22* may provide their results to the whole sub-team *T2* (**D7** in Figure 5) as a task model in CTT-notation [24] (see Figure 6).

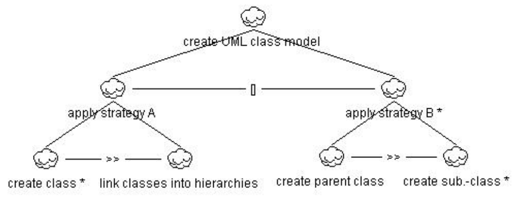

Figure 6: UML-modelling strategies (CTT-model).

Sub-team *T21* explore different options available for creating and visualising elements in the UI and enabling the user to manipulate them, for example:

- having a tool bar to drag and drop classes and hierarchies
- using mouse click and drags to create classes and links
- menu driven commands for all editing functions
- clickable visual representations of classes for editing

Figure 7 shows two of the UI-designs they are considering (**D8** and **D9** in Figure 5). In order to consider the relationship between the designs **D1**, **D8**, and **D9** (that is the relationship between the UI-designs and functional requirements) sub-team *T21* need a mechanism for describing the key elements of the UI-designs.

T21 use presentation models [4] for their notation, as they provide a lightweight mechanism for describing both designs and implementations of interactive systems with a formal underpinning. A presentation model describes an interface design by way of its component widgets, their types and their behaviours. So the presentation model for any design is a collection of tuples where each tuple represents one widget and consists of:

```
(widgetname, category, (behaviours))
```

The name is an identifier for the widget, the category indicates whether it is something a user interacts with to cause behaviour

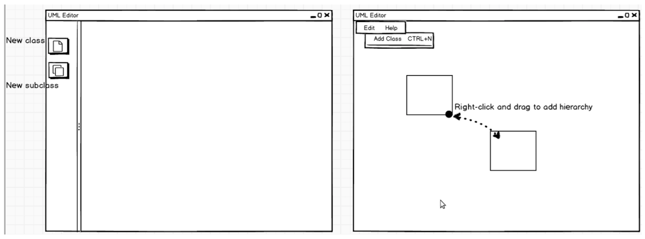

Figure 7: D8 and D9 Prototypes.

(like clicking on a button) or something which provides feedback to a user (like a display) and the behaviours are all behaviours that are generated by, or responded to, by the widget. In this way the presentation model describes both the visual elements of the interface and all behaviours. Its underlying semantics also provide a mechanism for a complete, unambiguous specification of the interactive system being designed, although that is not our primary focus in this work. The presentation models for the two prototype designs in Figure 7 are, therefore:

```
D8 is
addClassBtn, ActionControl, (S_AddClass)
addHierarchyBtn, ActionControl, (S_AddHierarchy)
D9 is
NewClassMenuItem, ActionControl, (S_AddClass)
RClickAction, ActionControl, (S_AddHierarchy)
```

Using the presentation models sub-team *T21* can, therefore, consider the aspects of the designs which they consider important when comparing them in some way - perhaps to look at concretisation of ideas, or compare different design approaches as they do here. *T21* can 'prove' (in a light-weight manner) that the behaviours of the UI-prototypes **D8** and **D9** are the same, based on the presentation models. They use the semantic function $PModel_{BEH}$ to extract the set of all behaviours from the models and then compare them, in this case:
$D8_{BEH} = \{S\_AddClass, S\_AddHierarchy\}$
$D9_{BEH} = \{S\_AddClass, S\_AddHierarchy\}$
So we see that both prototypes have the same intended behaviour and their difference is in the design of user controls and actions. Similarly, *T21* can show that both designs are valid *i.e.,* they satisfy the requirements of **D1**.

D8 and **D9** provide two different ways to allow the user to build the same set of UML class models. However, sub-team *T21* are reluctant to decide which of the design optionsis preferable. They are aware that the knowledge of sub-team *T22* about UI modelling strategies is needed to further evaluate them. Therefore, they decide to consider the designs **D8** and **D9** as *variants* (with respect to **D1**) to provide to the whole sub-team *T2*.

We stop the description of the example scenario at this point. Sub-team *T2* may continue, for example, with usability tests or with creating a design rationale (design **D10** in Figure 5) to analytically evaluate the UI-designs **D8** and **D9** and to decide on one of them. The task model **D7** may serve here as one of the criteria for assessment to ensure that the requirements of **D2** are considered. The design process may continue by bringing together the designs for the functional core and the UI-part and implementing them. This may include a redesign of the UI-prototype to allow users to perform UML-model validity checks as additionally suggested by sub-team *T1* in their design **D4**. We will refer back to the example scenario in the next section which introduces a semi-formal description of the proposed framework for interaction design spaces. In particular, we will further characterise the refinements carried out.

3.4 Semi-Formal Description of the Framework

In this section, we define a design space with respect to a given design project. Let $\mathscr{D}$ be the set of external design representations created and used in such a project (shortly referred to as designs) and $\mathscr{S}$ be the set of involved stakeholders. Furthermore, we assume a set of refinement approaches $\mathscr{R}$ containing formal, lightweight and valid refinement approaches ($\mathscr{R} = \mathscr{R}_{\mathscr{F}} \cup \mathscr{R}_{\mathscr{L}} \cup \mathscr{R}_{\mathscr{V}}$).

Definition 1 (Design Space)
A *design space DS* is a 7-tuple $(DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$, where

- $DT \subseteq \mathscr{S}$ is the designer team and $UT \subseteq \mathscr{S}$ is the user team of DS.
- $D \subseteq \mathscr{D}$ is a finite set of designs that is in DS accessible by DT (created and/or used by DT).
- $P^{entry} \subseteq D$ is a finite set of designs which is provided by UT via the *entry point* and represents external design constraints and requirements.
- $P^{exit} \subseteq D$ is a finite set of designs which is provided to UT via the *exit point* and represents the design outcome.
- $D \setminus (P^{entry} \cup P^{exit})$ is the set of *internal designs* of DS which is only accessible by DT but not by UT.
- DSS is a finite set of *design sub-spaces* $\{DS_1, DS_2, ..., DS_n\}$. Each design sub-space $DS_i = (DT_i, UT_i, D_i, P_i^{entry}, P_i^{exit}, DSS_i, DRel_i)$ with $i \in \{1,..,n\}$ is a design space itself, where
  - $DT_i \subseteq DT$ and $UT_i \subseteq DT$,
  - $D_i \subseteq \mathscr{D}$ and $D_i \cap D = P_i^{entry} \cup P_i^{exit}$,
  - The sets of internal designs of the design sub-spaces are pairwise disjoint.
- $DRel \subseteq Refine \times D \times D$ is a set of *refinement relations* between designs of D that is created by DT $(Refine \in \mathscr{R})$.

  We shortly write $d_1 \sqsubseteq d_2$ if $(\sqsubseteq, d_1, d_2) \in DRel$ (design d_2 refines design d_1). More specifically, we write $d_1 \sqsubseteq_F$ / $\sqsubseteq_L$ / $\sqsubseteq_V$ d_2 if $\sqsubseteq_F \in \mathscr{R}_{\mathscr{F}}$, $\sqsubseteq_L \in \mathscr{R}_{\mathscr{L}}$, and $\sqsubseteq_V \in \mathscr{R}_{\mathscr{V}}$ respectively.

Strictly speaking, Definition 1 gives a static view on design spaces and rather defines design space *states*. A dynamic view on design spaces would, for example, include the 'history' of a design space (e.g., as a sequence of states) and design activities changing the state of design spaces. Such activities include creating new designs within a design space, removing designs from it, modifying the set of refinement relations, providing designs via the entry point or exit point, modifying designs in the entry or exit point (via iterations), modifying designer and user teams etc. In the context of this paper, a static view of design spaces is sufficient to convey the main ideas.

Simple and complex design spaces.

A design space is called *simple design space* if the set of design sub-spaces is empty. In this case, all members of the designer team are familiar with all designs ever created and used within the design space. Otherwise the design space consists of a hierarchy of sub-spaces and there are further sub-teams in such *complex* design spaces creating (and possibly discarding) internal designs only visible to them. In the example in Figure 5, $\mathscr{D} = \{D1, D2, ..., D10\}$ is the set of all designs, $\{D1, D2, D4, D10\}$ is the set of designs accessible by the designer team *T*, $\{D1, D3, D4, D5, D6\}$ is accessible by the designer team *T1* and $\{D1, D2, D7, D8, D9, D10\}$ by *T2*. According to Definition 1, a stakeholder can be part of both a designer and a user team, and complex design spaces can even span intertwined networks of such user-designer relationships.

The following definition of the flattened version of a design space hides the complexity of complex spaces and describes the overall set of refinement relations of a design space DS that is created by both the designer team of DS and the sub-teams of possible sub-spaces of DS. It makes use of the transitivity property of refinements, that is, if there are designs d_1, d_2, d_3 where $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3$ ($\sqsubseteq \in \mathscr{R}$) then $d_1 \sqsubseteq d_3$ holds.

Definition 2 (Flattened version of a design space)
Let $DS = (DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$ be a design space. $DS^F = (DT, UT, D, P^{entry}, P^{exit}, DRel^F)$ is the *flattened version of* DS where the following condition holds for $DRel^F$:
For each $DS_i \in DSS$ let $DS_i^F = (DT_i, UT_i, D_i, P_i^{entry}, P_i^{exit}, DRel_i^F)$ be the flattened version of DS_i $(i = 1, ..., n)$. Then $DRel^F = DRel \cup \bigcup_{i=1,...,n} \{d \sqsubseteq d' \mid (\sqsubseteq, d, d') \in DRel_i^F, d \in P_i^{entry}, d' \in P_i^{exit}\}$.

Further characteristics of design spaces.

Based on previous definitions, a number of definitions are now introduced describing design spaces and their quality in more detail. First, we consider the completeness of design spaces.

Definition 3 (Complete design space)
Let $DS = (DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$ be a design space with $P^{exit} \neq \emptyset$ and $DRel^F$ be the set of refinement relations from the flattened version of DS. DS is said to be *complete* if each $DS_i \in DSS$ is a complete design space and if for each design $d \in P^{exit}$ the following condition holds: for all $d' \in P^{entry}$ there exists a $(\sqsubseteq, d', d) \in DRel^F$.

Definition 4 (Valid design space)
Let $DS = (DT, UT, D, P^{entry}, \emptyset, DSS, DRel)$ be a design space with $DRel^F$ be the set of refinement relations from the flattened version of DS. DS is said to be *valid* if each $DS_i \in DSS$ is a valid design space and if for all internal designs $d \in D \setminus P^{entry}$ the following condition holds: there exists a nonempty set $\{d_1, ..., d_n\} \subseteq P^{entry}$ $(n > 0)$ where $(\sqsubseteq, d_i, d) \in DRel^F$ $(i = 1, ..., n)$.

While a valid design space can potentially lead to a satisfactory solution, a complete design space provides a set of such solutions at the exit point. If a design space cannot be shown to be valid the designer team needs to discard some internal designs at least. However, if a design team cannot create a complete design space at all, it has to start a new iteration with the user team and re-negotiate the designs of the entry point.

Definition 4 of valid design spaces may lead us to consider *partially complete design spaces* where designs in the exit point refine only some of the designs in the entry point. In other words, no design(s) is provided that unifies the consideration of all requirements and constraints. Instead, a set (or sets) of refined designs leaves the entry point and is provided to other sub-teams. For the sake of space, we do not provide a formal definition of this here.

The design options of a design space provide information about the exploratory activities of the design team (concept generation). The distinction of alternatives and variants provide additional information about decision making processes (concept convergence).

Definition 5 (Design options)
Let $DS = (DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$ be a design space and $DRel^F$ be the set of refinement relations from the flattened version

of DS. $D^{opt} \subseteq D \setminus (P^{entry} \cup P^{exit})$ is a set of *design options* with respect to d ($d \in D \setminus D^{exit}$ and $d \notin D^{opt}$) if for each $d' \in D^{opt}$ we have $(\sqsubseteq, d, d') \in DRel^F$.

Definition 6 (Design alternative)
Let $DS = (DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$ be a complete design space and $DRel^F$ be the set of refinement relations from the flattened version of DS. A design $d' \in P^{exit}$ is called *design alternative* with respect to a design d if there exists a set of design options D^{opt} with respect to d and the following conditions hold: 1) there exists a $d_1 \in D^{opt}$ such that $(\sqsubseteq, d_1, d') \in DRel^F$, and 2) there exists no other design $d_2 \in D^{opt}$ such that there exists a $d'' \in P^{exit}$ with $(\sqsubseteq, d_2, d'') \in DRel^F$.

Definition 7 (Design variants)
Let $DS = (DT, UT, D, P^{entry}, P^{exit}, DSS, DRel)$ be a complete design space and $DRel^F$ be the set of refinement relations from the flattened version of DS. The designs $d_1, d_2 \in P^{exit}$ ($d_1 \neq d_2$) are called *design variants* with respect to a design d if there exists a set of design options D^{opt} with respect to d with $d'_1, d'_1 \in D^{opt}$ and $d'_1 \neq d'_2$ such that $(\sqsubseteq, d'_1, d_1) \in DRel^F$ and $(\sqsubseteq, d'_2, d_2) \in DRel^F$.

In the above example scenario, all of the mentioned design (sub-)spaces are valid and sub-spaces T1, T2, T21, and T22 are also complete. The functional requirements of D1 can be expressed formally via the specification(s) of sub-team *T1*. We can then use formal refinement theories such as those described in [27, 8]. Furthermore, designs D4, D5, and D6 are options with respect to D3 and D4 is the alternative that leaves the exit point (internal idea generation and convergence).

We can similarly consider these requirements (D1) in the designs created by sub-team *T21* using the refinement theory for presentation models [3] based on intended behaviours (shown above). This is an example of lightweight refinement, D8 and D9 are options with respect to D1. Both designs leave the sub-space as variants supporting idea generation and convergence across sub-teams. The requirements of D2 are considered more formally in D7. However, the task model is a result of discussions with end-users, observations etc. While this is not obviously a type of refinement (based on the three descriptions given previously) it does allow for other, established, design formalisms to be incorporated into our approach.

Typically we are most interested in the difference between designs at the entry and exit points of the design spaces as these are where decisions are in some sense finalised, but we might similarly be interested in specific designs within the space if we want to use a design rationale approach to assist with the design enumeration and choice activities. D10 is a QOC-diagram with D8 and D9 as options, and D7 used in the criteria part. As such it is a valid refinement of D8, D9, and D7 containing the decision for D8 and a record of the reasons behind this decision. The use of task models in QOC-diagrams has been discussed elsewhere, e.g., in [18].

4. DISCUSSION AND FUTURE WORK

Multi-disciplinary design work is challenging and requires complementing education and expertise in single disciplines with an appreciation and understanding of other disciplines [19]. The suggested framework describes design activities at an abstract level in terms of design spaces, designs, refinement relationships etc. It also generalises the relationship between designers and users by considering a network of such relationships with the participants typically acting in different roles (as discussed, e.g., in end-user programming contexts). This general view allows design teams and other stakeholders to reflect upon, or plan, the overall design process.

As mentioned in the background section, common design space models using the funnel-metaphor may impede a diversity of design ideas due to their focus on one goal and one final solution [26]. Alternatives and variants in our framework provide a more relaxed view on the designer's goal-directed activities by allowing the convergence of design ideas within and across design sub-spaces. Therefore, these concepts support the awareness of expertise and possible contributions of different collaborators and help avoid premature design decisions.

Of course, abstraction has its price. Note that we deliberately have not said anything about how, for example, requirements are expressed but refer to any kind of external design representation as a *design*. This may have implications when we wish to use (any type of) refinement if there is ambiguity or conflict in the requirements hidden by this ambiguity. Similarly, we have necessarily omitted detailed technical proofs of refinement relations because they do not fit in this paper. However both the choosing of suitable formal notations and refinement mechanisms, as well as the conducting of such proofs can bring their own challenges.

Our intention then is to show how we can rely on these existing principles rather than fully elaborate on them here. However, for more practical use the framework needs to be 'instantiated' by certain types of designs and refinements (as indicated in the example scenario). Interesting research questions in this context are about the composition or merging of designs and about the opposite of splitting out parts of a design. How can refinements of different types be combined? Future work also includes the consideration of other useful characteristics of design spaces than those given in the paper. One such could be the 'degree of multi-disciplinarity' that reveals the way different refinement approaches are intertwined. What we provide in this paper is a framework to enable reasoning about such heterogeneous design spaces. Empirical studies are necessary to elaborate the applicability of the approach and to further develop it.

5. CONCLUSION

In this paper we have presented a semi-formal framework for interaction design processes. This includes formal, informal and mixed notations for describing design spaces, designs and the relationships between designs. It is not our intention to provide a new process for design, but rather to find ways of capturing the different concerns and diversity of design representations that arise within existing design processes. We then use this as a mechanism for supporting typical creative and iterative design for interactive systems. This enables us to support such a process by enabling comparison of different designs and differing levels of formality as well as recording design decisions and relating all of these back to initial and emerging requirements.

6. REFERENCES

[1] ISO/IEC 13568. 2002. *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics* (first ed.). ISO/IEC.

[2] V. Bellotti, S.B. Shum, A. MacLean, and N. Hammond. 1995. Multidisciplinary Modelling in HCI Design in Theory and in Practice. In *SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. ACM, 146–153.

[3] J. Bowen and S. Reeves. 2006. Formal Refinement of Informal GUI Design Artefacts. In *Australian Software Engineering Conference (ASWEC'06)*. IEEE, 221–230.

[4] J. Bowen and S. Reeves. 2008. Formal Models for User Interface Design Artefacts. *Innovations in Systems and Software Engineering* 4, 2 (2008), 125–141.

[5] B. Buxton. 2007. *Sketching User Experiences : Getting the design right and the right design*. M. Kaufmann.

[6] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. 2003. A Unifying Reference Framework for Multi-target User Interfaces. *Interacting with Computers* 15, 3 (2003), 289–308.

[7] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. 2000. *Non-Functional Requirements in Software Engineering*. Boston: Kluwer Academic Publishers.

[8] J. Derrick and E. Boiten. 2001. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. Springer.

[9] A. Dittmar and M.D. Harrison. 2010. Representations for an Iterative Resource-based Design Approach. In *2nd ACM SIGCHI symposium on Engineering interactive computing systems (EICS '10)*. ACM, 135–144.

[10] A. Dittmar and S. Piehler. 2013. A Constructive Approach for Design Space Exploration. In *5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '13)*. ACM, 49–58.

[11] A. Dix. 1991. *Formal Methods for Interactive Systems*. Academic Press.

[12] A. Dix, J.E. Finlay, G.D. Abowd, and B. Beale. 2003. *Human-Computer Interaction (3rd ed.)*. Prentice-Hall.

[13] A. Dix and L. Gongora. 2011. Externalisation and Design. In *Second Conference on Creativity and Innovation in Design*. ACM, 31–42.

[14] T. Döring, A. Sylvester, and A. Schmidt. 2013. A Design Space for Ephemeral User Interfaces. In *7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 75–82.

[15] H. Dubberly and S. Evenson. 2008. On Modeling: The Analysis-synthesis Bridge Model. *Interactions* 15, 2 (2008), 57–61.

[16] W. Gaver. 2014. Science and Design: The Implications of Different Forms of Accountability. In *Ways of Knowing in HCI*, Judith S. Olson and Wendy A. Kellogg (Eds.). Springer, 143–165.

[17] J. Gulliksen, B. Göransson, I. Boivie, S. Blomkvist, and Å. Cajander. 2003. Key Principles for User Centred Systems Design. *Behaviour and Information Technology* 22, 6 (2003), 397–409.

[18] X. Lacaze, P. Palanque, E. Barboni, R. Bastide, and D. Navarre. 2006. From DREAM to Reality: Specificities of Interactive Systems Development With Respect To Rationale Management. In *Rationale Management in Software Engineering*, A.H. Dutoit, R. McCall, I. Mistrik, and B. Paech (Eds.). Springer, 155–172.

[19] W.E. Mackay. 2003. Educating multi-disciplinary design teams. In *Tales of the Disappearing Computer*. ACM Press.

[20] A. MacLean, V. Bellotti, and S. Shum. 1993. Developing the design space with design space analysis. In *Computers, Communication and Usability: Design issues, research and methods for integrated services*, P. F. Byerley, P. J. Barnard, and J. May (Eds.). Elsevier.

[21] A. MacLean, R.M. Young, V.M.E. Bellotti, and T.P. Moran. 1991. Questions, Options, and Criteria: Elements of design space analysis. *Human Computer Interaction* 6, 3 (1991), 201–250.

[22] C. Morgan. 1998. *Programming from Specifications (2nd ed.)*. Prentice Hall International (UK) Ltd.

[23] L. Nigay and J. Coutaz. 1993. A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion. In *INTERACT '93 and CHI '93*. ACM, 172–178.

[24] F. Paterno. 2000. *Model-Based Design and Evaluation of Interactive Applications*. Springer.

[25] H.W.J. Rittel and M.M. Webber. 1973. Dilemmas in a General Theory of Planning. *Policy Sciences* 4 (1973), 155–169.

[26] B. Westerlund. 2009. *Design Space Exploration - Co-operative creation of proposals for desired interactions with future artefacts*. dissertation. Kungliga Tekniska högskolan, Stockholm.

[27] J. Woodcock and J. Davies. 1996. *Using Z: Specification, Refinement and Proof*. Prentice Hall.