Improving the Quality and the Comprehension of Requirements: Disciplined Use Cases and Mockups

Gianna Reggio, Filippo Ricca, Maurizio Leotta

Abstract:

In this paper we sketch a novel method for writing requirements specifications that enriches disciplined use cases with screen mockups. Disciplined use cases are characterized by a quite stringent template, strongly structuring also the form of the scenarios' steps (e.g., the subject of each step must be explicit). That structuring allows to impose further constraints helping to prevent common mistakes and to increase the quality of the specifications (e.g., defining a detailed glossary helps to avoid confusion and ambiguities). Disciplined use case are still expressed using natural language, but the strong structuring allows to reach a good level of precision without having to introduce new notations. Screen mockups associated with the scenarios' steps, present the corresponding GUIs as seen by the human actors before/after the step executions, improving the comprehension of the requirements, and allowing also to precisely present user interface's non-functional requirements.

Digital Object Identifier (DOI):

`http://dx.doi.org/10.1109/SEAA.2014.79`

Improving the Quality and the Comprehension of Requirements: Disciplined Use Cases and Mockups

Gianna Reggio, Filippo Ricca, Maurizio Leotta
DIBRIS, Università di Genova, Italy
gianna.reggio@unige.it, filippo.ricca@unige.it, maurizio.leotta@unige.it

Abstract—**In this paper we sketch a novel method for writing requirements specifications that enriches disciplined use cases with screen mockups. Disciplined use cases are characterized by a quite stringent template, strongly structuring also the form of the scenarios' steps (e.g., the subject of each step must be explicit). That structuring allows to impose further constraints helping to prevent common mistakes and to increase the quality of the specifications (e.g., defining a detailed glossary helps to avoid confusion and ambiguities). Disciplined use case are still expressed using natural language, but the strong structuring allows to reach a good level of precision without having to introduce new notations. Screen mockups associated with the scenarios' steps, present the corresponding GUIs as seen by the human actors before/after the step executions, improving the comprehension of the requirements, and allowing also to precisely present user interface's non-functional requirements.**

Keywords-**Requirements Specification, Use Case, Screen Mockups.**

I. Introduction

It is well known that a substantial portion of software defects originate in the requirements engineering phase of the software development process [15]. Defects originated in this phase are typically caused by ambiguous, incomplete, inconsistent, unexpressed, unusable and over-specific requirements. Defects might also stem from communication problems among stakeholders [8]. To face these issues, a number of methods/techniques have been proposed in literature for representing requirements. Use cases are a widely used technique to specify the purpose of a software system, and to produce its description in terms of interactions between actors and the subject system [4].

Even presented by means of use cases, the requirements could still remain difficult to use and/or to comprehend and may result in problems in the software system under development (e.g., contradictory requirements). Screen mockups (also called user interface sketches or user interface mockups) are used for prototyping the user interface of a subject system [6], [9]. Mockups can be used in conjunction with use cases, associating them with the scenario steps, to improve the comprehension of functional requirements and to achieve a shared understanding on them, and simultaneously allowing to represent the non-functional requirements concerning the user interface [5].

Enriching the use cases with the screen mockups arises the problem of guaranteeing that the mockups are coherent/consistent with the textual part of the use cases, and that they truly provide information on how to structure the GUI's supporting the functionalities presented by the use case (e.g., the screen mockup associated with the step "the user inserts the password" that shows a form with two text boxes is not consistent). Screen mockups not corresponding to what is expressed by the use cases scenarios may become a factor of confusion and add new ambiguities. Furthermore, the screen mockups may or not convey the same information of the textual part of the use cases. For example, if the mockup associated with the step "The system asks to pay X dollars" presents two text boxes showing net price and VAT, then, it also adds the additional low level requirement "the system has to inform the client also about the paid VAT". If the screen mockups present relevant information on the subject system not provided by the textual part of the use cases, they are not any more a support to better understand the requirements, but they become a fundamental part of the requirements specification.

As a matter of fact, the consistency between use cases and screen mockups cannot be guaranteed if the former are poorly structured and have a low level of precision. Requirements specifications based on use cases may have very different levels of precision[1], from scarcely structured scenarios made by lists of freely formed natural language sentences, to use cases presented following quite detailed and structured templates, for example SWEED[2] (a variant of the template presented by Cockburn in [4]) where a glossary of terms used in the definition of use cases is added, till to methods where the use cases are represented by means UML models [2] or even by formal specifications, see, e.g., [3].

We believe that disciplined natural language specifications, i.e. where the text must follow very detailed and stringent patterns, are a good compromise to express requirements [10], [7]. For this reason, we have conceived the "disciplined use cases" enriched by mockups, taking the SWEED template as starting point. Disciplined use cases are: (1) characterized by a high level of precision without having to introduce additional notations and the consequent effort required to learn and to use them, (2) suitable to be enriched in a consistent way with the screen mockups, and (3) useful to detect errors, incompleteness, bad smells (e.g., unused elements), and bad quality factors (e.g., too many extensions and too many steps in a scenario) in the requirements specification (thanks to many well-formedness constraints).

[1]Exactly or sharply defined or stated (Merriam Webster's Dictionary).
[2]http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.196.7782

Following our method allows to obtain consistent screen mockups that are fully integrated in the development process, and thus that are not simply a bunch of drawings added at the end of the specification. To the best of our knowledge, this is the main novelty aspect of our work.

The remainder of the paper is organized as follows. Sect. II and Sect. III, describe our method to specify the requirements using disciplined use cases and screen mockups followed by conclusions and future work (Sect. IV).

II. Disciplined Use Cases

We illustrate our proposal for specifying the requirements by applying it to the AL_L case study, described as follows. AL_L handles the **AL**gebraic **L**otteries; it will allow the clients to buy the tickets via the web, and to pay them by using credit cards, handled by an external system. AL_L will communicate with the clients also by means of emails. Moreover, the clients will be identified with the help of an external authentication service. The tickets of a lottery will be numbered by a finite subset of the integer numbers, and the winning ones will be those labelled with the higher numbers with respect to a specific total order determined by an algebraic expression[3]. The managers may distribute some free tickets to those that have already bought at least one ticket, again an algebraic expression will determine which tickets to give away and to which clients.

The form of the requirement artifacts is shown in Fig. 1 by a metamodel presented by a UML class diagram. A *requirements specification* consists of a UML use case diagram, of a description of each use case appearing in that diagram, and of a glossary that lists and makes precise all the terms used in the descriptions of the use cases. The use case diagram is the only part of the specification not expressed using the natural language. We have opted for including this diagram since it is really valuable for summarizing use cases and actors and it is quite simple to explain and produce.

The *glossary* is a list of entries, each one consisting of the name of the defined term and of a corresponding short description. The glossary entries are distinguished in those relative to *data* (e.g., the credit card data, the client info), and those about the attributes abstractly describing the state of System, indicated as *system attributes* (e.g., the list of the names of the currently registered clients, the number of tickets of the current lottery already sold). The main objective of a glossary is to: *(i)* shorten use cases; *(ii)* reduce ambiguities (e.g., to avoid using different ways to refer to the same entity); and *(iii)* clarify the meaning of the steps of a scenario (e.g., a richer description of the various entities could be given). Fig. 2 shows a fragment of the requirements specification glossary of the AL_L case study (the complete one can be found in [11]).

The *use case diagram* summarizes the main ways to use the System (i.e., it presents its *use cases*), making clear which *actors* take part in them. Recall that actors are roles for entities interacting with System, not specific individuals. The actors are distinguished in: – *primary*, those having goals on System, i.e., those that obtain value from interacting with the System, and – *secondary*, those over which the System has a goal, i.e., those that support it in creating value for primary actors. The use case diagram for the AL_L case study is reported in [11].

[3]Such expressions will be provided to the system managers by some control authority encrypted in some way, and so the managers cannot manipulate the prize drawing.

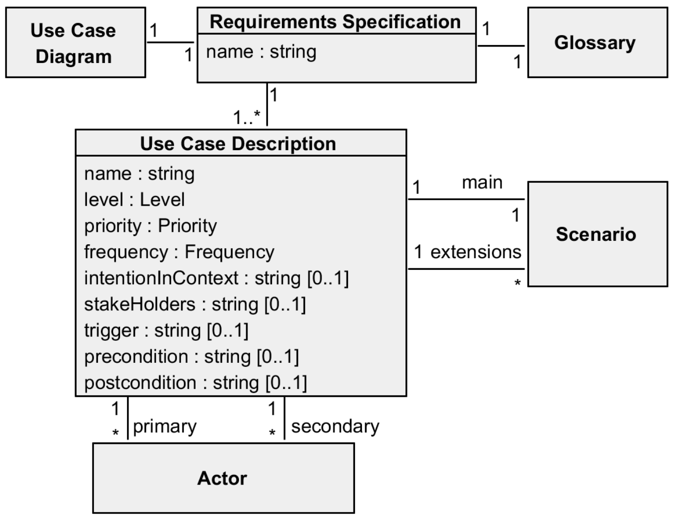

Fig. 1. Requirements Specification Structure

A *use case description* consists of general information about the use case, plus a set of scenarios (see Fig. 1). Fig. 3 presents an example of use case description; it refers to the use case Register of the AL_L system. It is periodical, with high priority, its level is user-goal, and its aim is to allow the clients to register to the AL_L system. A primary actor (Client) and two secondary actors (Authentication and Credit Card Service) take part in this use case.

A use case includes several *scenarios*, see Fig. 1. We present the abstract structure of a scenario in Fig. 4. The *main success scenario* describes the basic execution of the use case. The *extensions* (optional) are a set of scenarios, defining all the other possible executions of the use case. A *scenario* is an ordered sequence of steps, where each step describes an interaction between the System and one of the actors of the use case; thus, as said before, it represents a particular execution of the use case. Technically, a scenario is a sequence of numbered *lines*, where each line is either a basic step or an indication of

Data
(**3) *credit card data*: the information characterizing a credit card (Type (Mastercard, Visa, Diners), Number, Expiration Date)
(**4) *client info*: the information about a registered client (email, data of his/her credit card, numbers of the bought tickets)
(**5) *password*: more than 8 characters with the inclusion of at least one special character (e.g., #, $)
(**6) *email*: a valid email address (RFC 5322 compliant)
System attributes
(**9) *Running*: a boolean, true if a lottery is currently running
(**10) *Available tickets*: set of integer numbers, the numbers of the tickets of the current lottery still not assigned to some client
(**11) *Registered Clients*: the information about the registered clients

Fig. 2. AL_L Requirements Specification: a fragment of the glossary

Use Case Register
Level: User Goal
Priority: 1
Frequency: Periodically
Intention in Context: A client wants to register herself/himself to AL_L to be able to play in the lotteries
Primary Actor(s): Client
Secondary Actor(s): Credit Card Service, Authentication
Main Success Scenario:
showRegisterMockup
1. The Client asks AL_L to be registered, giving an email (**6) and the data about a credit card (**3).
2. If no one among the registered clients (**11) is using the given email, and the credit card data are well-formed (**3), AL_L asks the Credit Card Service to check them.
3. The Credit Card Service informs AL_L that the submitted credit card is valid.
4. AL_L asks the Authentication to register the client giving his/her email.
5. Authentication confirms the registration and gives to AL_L the client password (**5).
6. AL_L informs the Client that (s)he has been registered and gives her/his password. The information about Client (her/his email, credit card data) (**4) is added to the list of the clients (**11). The use case ends with success.

Extensions:
2a.1 If the credit card data are ill-formed (**3) AL_L informs Client that the registration has failed. The use case ends with failure.
showRegisterFailedMockup
2b.1 If someone among the registered clients (**11) is using the given email, AL_L informs Client that the registration has failed. The use case ends with failure.
showRegisterFailedMockup
3a.1 The Credit Card Service informs that the submitted credit card is invalid.
3a.2 AL_L informs the Client that the registration has failed. The use case ends with failure.
showRegisterFailedMockup

Fig. 3. Register Use Case. Underlined terms represent hyperlinks to screen mockups. (**...) refer to items of the glossary shown in Fig. 2

a repetition of some steps or the inclusion of another use case or an extension point.

A basic step has the following form:

[cond] subject interaction effect continuation

– *cond* is a natural language fragment stating the condition under which the step may be executed (it should start with "If"). It is optional, if it is not present, it is intended as the always true condition. It should be about the System state attributes (i.e., the current state of System) and the data appearing in the interaction and effect part of the step, and thus it will be expressed using the terms introduced by the glossary.

– *subject* may be either an actor (primary or secondary) of the use case or the System.

– *interaction* is a sentence describing either what flows from the actor towards the System or vice versa, it should be expressed using a verb in the present tense third person.

– *effect* is a sentence describing a transformation of the System state attributes, and thus it will be written using the terminology

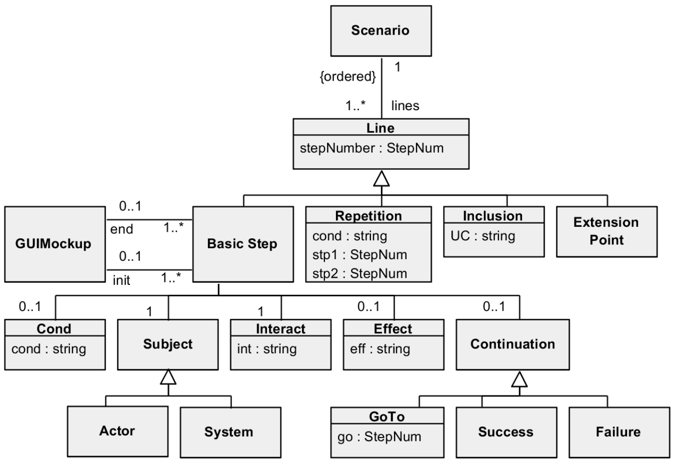

Fig. 4. Structure of a Scenario

introduced in the glossary. It is optional, and if it is not present, then the step does not influence the System attributes.
– *continuation* defines what to do at the end of the step. It may have one of the following forms:
- "The use case continues to" stp_n, where stp_n is the number of a step of a scenario of the use case,
- "The use case ends with success/failure".

It is optional, and if it is not present, then the use case continues to the next step.

A *Repetition* is a natural language fragment having form: "The steps from" stp_1 "to" stp_2 "are repeated until" *cond*, where stp_1 and stp_2 are the numbers of two steps of the scenario including this line, and *cond* is as the conditions for the steps.

An *Extension point* denotes where the behaviour of an extending use case (i.e., a use case related by the extension relationship) will be inserted.

An *Extension* is a scenario defined modifying an existing one, by giving a different sequence of steps starting from a given step (the extended step).

An *Inclusion* line is just written by reporting the name of the included use case, that should be linked to the described use case by the inclusion relationships in the use case diagram.

The steps of the main success scenario are labelled by natural numbers (i.e., 1, 2, 3 ...). The steps of the first scenario extending a step *X* are numbered with *Xa.1*, *Xa.2*, ... and those of the second scenario *Xb.1*, *Xb.2*, ... and so on (see for instance Fig. 3).

The use case Register for the AL_L case study, shown in Fig. 3, has three extensions corresponding to the cases: – the provided email is already used by a registered client (2a), – the data about the credit card were ill-formed (2b), and – the credit card is invalid (3a).

It is possible to verify that a use case follows the disciplined style by checking whether it satisfies a set of well-formedness constraints, reported in [11].

III. Screen Mockups

One or two *screen mockups* may be associated with each basic step, where a human actor is involved; they are a

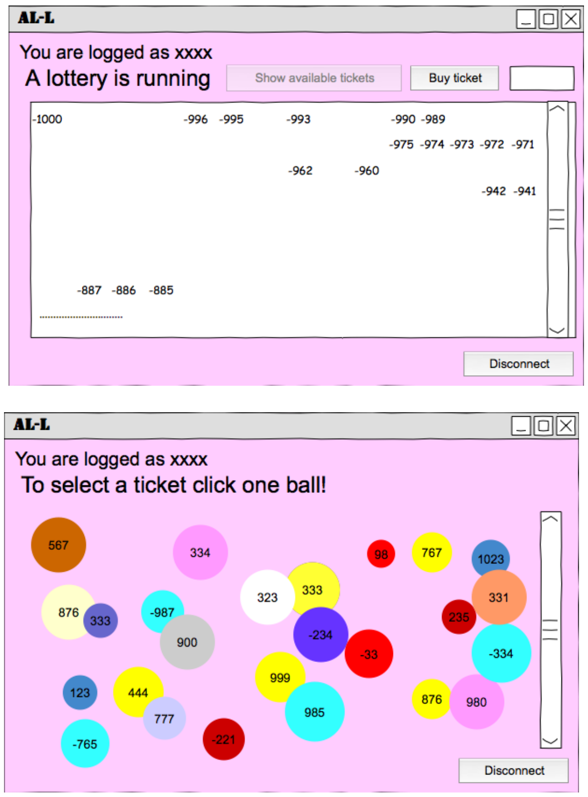

Fig. 7. Use Case Buy Ticket. Two different initial screen mockups for the step "Registered Client asks to buy the ticket with number N."

visualization of the GUI that will be shown at that point. Mockups are drawings that show how the user interface of System is supposed to look during the interaction between it and the end-user (user-system interaction). Mockups may be very simple, just to help the presentation of the user-system interactions, or more detailed with rich graphics, whenever specific constraints on the graphical user interface are highlighted (e.g., requiring to use specific logos or brand related colours) [9]. Fig. 5 and 6 show two screen mockups for the use case Register (see Fig. 3) of the AL_L application.

When resources are limited and it is not possible to develop the mockups for all the use cases, we recommend to privilege the ones with frequency at least *Periodically*. Moreover, mockups should be associated with the scenario steps considered more relevant (it is the requirements analyst who makes this decision) to show what the human actor will see/will do at that moment, when (s)he will interact with the System.

More in detail, let M be a mockup associated with a step S. If the subject of S is the system, then M will show what can be seen on the GUI at the end of the step. If instead the subject of S is an actor, M may be associated: – with the end of S, and in this case it will show what the actor will see just immediately before to complete S (e.g., after having filled various fields just before to press the button "send"); and – with the beginning of S, and in this case M will show what the actor will see just immediately before to start to execute the step. The initial screen mockups of the steps corresponding to extensions of S obviously must coincide.

Placeholders for the mockups will be inserted before or after the various steps in the use case descriptions, precisely the initial ones before the corresponding step, and the end ones after the step. Obviously, whenever the initial mockup for an actor step coincides with the end one of the previous step, its placeholder will appears only once in the scenario. Placeholders may be realized in different ways depending on the technology used to write the use cases (e.g., a link to a picture in a Word document and a "click to enlarge the picture" in a HTML document). In Fig. 3 the underlined terms are links to the pictures reported in Fig. 5 and 6 respectively.

The use of the placeholders allows the readers of the use case to choose when to examine the screen mockups, ignoring them when interested only in the flow of the various steps. Instead, by replacing all the placeholders with the corresponding pictures we get an alternative visualization of the use cases (see some examples in [11]) corresponding to the so called "paper prototype" of System.

The disciplined style of the use cases allows to check/to enforce to some extent the coherence of the mockups with the textual parts of the use cases. Indeed, they should satisfy a comprehensive set of constraints completely listed in [11]; here we only report a sample constraint in Fig. 8. The two mockups shown in Fig. 5 and 6 satisfy all the constraints, and thus are well-formed.

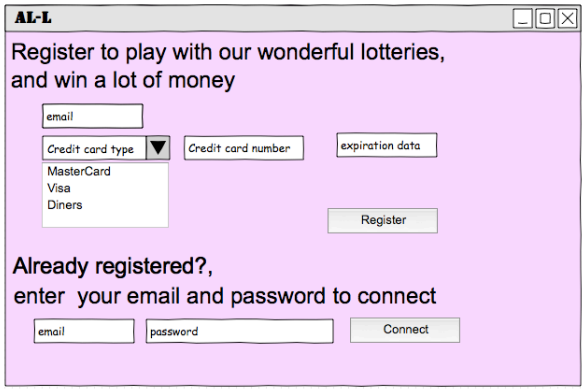

Fig. 5. Use Case Register "showRegisterMockup" Mockup

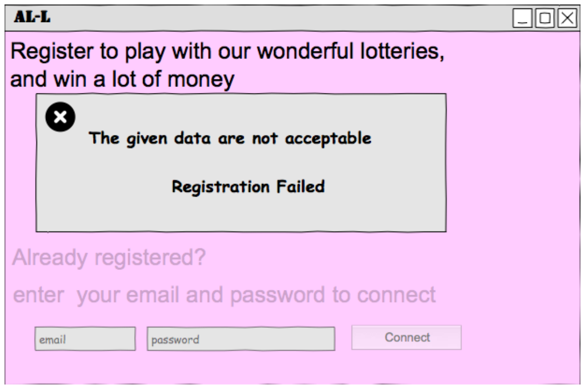
AL-L
Register to play with our wonderful lotteries,
and win a lot of money
The given data are not acceptable
Registration Failed
Already registered?
enter your email and password to connect
email
password
Connect

Fig. 6. Use Case Register "showRegisterFailedMockup" Mockup

Let M be a screen mockup showing the GUI before steps S_1, ..., S_n, i.e., the initial steps of n extensions starting from the same point, and all of them having an actor as subject.

– If the interaction part of S_i ($1 \leq i \leq n$) refers to some communication act (from the actor to system), then some means to represent such act must appear in M *(e.g., this constraint is satisfied if there are two alternative steps "Client confirms", and "Client refuses", and two buttons "Confirm" and "Refuse" appear in* M*).*

– If M contains some means for realizing some communication act (from the actor to system), then there should be S_i ($1 \leq i \leq n$) referring to such communication act.

Fig. 8. Sample Well-formedness Constraints for Screen Mockups

For documentation it is possible to mark the screen mockups to make explicit the relationships between the textual parts of the associated steps and the graphical ingredients of the mockups, to show that they are well-formed.

The screen mockups shown in Fig. 5 and 6 are quite simple since the registration functionality does not need a complex user interface neither an original one. Instead, in the case of the use case Buy Ticket (see its description in [11]) the user interface related to the steps for selecting the ticket number, can be created in many different ways. It is important to allow to express the (non-functional) requirements about the GUI by choosing one specific way. We show in Fig. 7 two different possibilities.

IV. Conclusion and Future Work

In this work we have proposed a new kind of requirements specification based on disciplined textual use cases enriched with screen mockups. The stringent constraints on the text presenting the use cases and the glossary guarantee a good quality level and reduce ambiguities, incompletenesses, inconsistencies. Moreover, the addition of screen mockups strictly corresponding to the use case steps largely improves the comprehension of the requirements [13], without being the source of additional problems. Furthermore, mockups allow to express the non-functional requirements about the user graphical interface.

Our approach is light-weight [12], [14], since it does not require the use of any specific tool, nor the use of specific formal or semi-formal notations, and can be learned in just a few hours. It has been successfully validated by an industrial project, several empirical experimentations [13], and by being used for many years in the non-trivial student's projects during the software engineering course at the University of Genova [1].

Moreover, disciplined use cases with screen mockups are a suitable starting point for:

– easily producing acceptance tests for the System covering also the graphical user-interface aspects, and the high level of precision of the scenario steps will allows to easily determine the input and output to be used in the tests;

– formal inspection of the designed system: e.g., for each data glossary item, check how it has been implemented; for each system attribute item, check how it has been realized by means of persistent data; for each screen mockup, check how it has been implemented, and the associated steps will drive the inspection of the events related to its widgets.

As a future work, we plan to extend our proposal by completing it with a method to capture the requirements and express them in the form described in this work. Moreover, we also plan the realization of a supporting tool, helping in the creation and visualization of the requirements specifications, and most important in checking automatically the satisfaction of the various well-formedness constraints. A prototyping version of such tool has been realized as student project in a software engineering course.

References

[1] E. Astesiano, M. Cerioli, G. Reggio, and F. Ricca. A phased highly-interactive approach to teaching UML-based software development. In *Proceedings of Educators Symposium at MoDELS 2007*, pages 9–18. Research Reports in Software Engineering and Management, IT University of Goteborg, 2007.

[2] E. Astesiano and G. Reggio. Knowledge structuring and representation in requirement specification. In *Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE 2002, pages 143–150. ACM, 2002.

[3] C. Choppy and G. Reggio. Improving use case based requirements using formally grounded specifications. In M. Wermelinger and T. Margaria-Steffen, editors, *Fundamental Approaches to Software Engineering*, volume 2984 of *LNCS*, pages 244–260. Springer, 2004.

[4] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2000.

[5] J. Ferreira, J. Noble, and R. Biddle. Agile development iterations and UI design. In *Proceedings of Agile Conference*, AGILE 2007, pages 50–58, 2007.

[6] H. R. Hartson and E. C. Smith. Rapid prototyping in human-computer interface development. *Interacting with Computers*, 3(1):51–91, 1991.

[7] M. Leotta, G. Reggio, F. Ricca, and E. Astesiano. Towards a lightweight model driven method for developing SOA systems using existing assets. In *Proceedings of 14th International Symposium on Web Systems Evolution*, WSE 2012, pages 51–60. IEEE, 2012.

[8] B. Meyer. On formalism in specification. *IEEE Software*, 3(1):6–25, January 1985.

[9] M. O'Docherty. *Object-Oriented Analysis and Design: Understanding System Development with UML 2.0*. Wiley, 1 edition, June 2005.

[10] G. Reggio, M. Leotta, F. Ricca, and E. Astesiano. Business process modelling: Five styles and a method to choose the most suitable one. In *Proceedings of 2nd International Workshop on Experiences and Empirical Studies in Software Modelling*, EESSMod 2012, pages 8:1–8:6. ACM, 2012.

[11] G. Reggio, F. Ricca, and M. Leotta. Improving the quality and the comprehension of requirements: Disciplined use cases and mockups (complete version). Technical Report DIBRIS-TR-14-02, DIBRIS - University of Genova, Italy, February 2014. Available at http://sepl.dibris.unige.it/TR/UseCasesMockups.pdf.

[12] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano. On the effort of augmenting use cases with screen mockups: results from a preliminary empirical study. In *Proceedings of 4th International Symposium on Empirical Software Engineering and Measurement*, ESEM 2010, pages 40:1–40:4. ACM, 2010.

[13] G. Scaniello, F. Ricca, M. Torchiano, G. Reggio, and E. Astesiano. Assessing the effect of screen mockups on the comprehension of functional requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, (in press), 2014.

[14] G. Scanniello, F. Ricca, M. Torchiano, C. Gravino, and G. Reggio. Estimating the effort to develop screen mockups. In *Proceedings of 39th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA 2013, pages 341–348, 2013.

[15] R. Young. *Effective Requirements Practice*. Addison-Wesley, 2001.