Reggio G, Leotta M, Ricca F *et al.* DUSM: A method for requirements specification and refinement based on disciplined use cases and screen mockups. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33(5): 918–939 Sept. 2018. DOI 10.1007/s11390-018-1866-8

DUSM: A Method for Requirements Specification and Refinement Based on Disciplined Use Cases and Screen Mockups

Gianna Reggio, Maurizio Leotta*, Filippo Ricca, and Diego Clerissi

Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS)
University of Genova, Genova 16146, Italy

E-mail: {gianna.reggio, maurizio.leotta, filippo.ricca}@unige.it; diego.clerissi@dibris.unige.it

Received March 8, 2018; revised July 30, 2018.

Abstract In this work, we present DUSM (Disciplined Use Cases with Screen Mockups), a novel method for describing and refining requirements specifications based on disciplined use cases and screen mockups. Disciplined use cases are characterized by a quite stringent template to prevent common mistakes, and to increase the quality of the specifications. Use cases descriptions are formulated in a structured natural language, which allows to reach a good level of precision, avoiding the need for further notations and complex models. Screen mockups are precisely associated with the steps of the use cases scenarios and they present the corresponding GUIs (graphical user interfaces) as seen by the human actors before/after the steps executions, improving the comprehension and the expression of the non-functional requirements on the user interface. DUSM has been proposed and fine-tuned during several editions of a software engineering course at the University of Genova. Then, by means of a series of case studies and experiments, we validated the method and evaluated: 1) its effectiveness in improving the comprehension and, in general, the quality of the produced requirements specification, and 2) its applicability in the industry, where the method has been found useful and not particularly onerous.

Keywords requirements specification, use case, screen mockup, empirical validation, graphical user interface (GUI), user interface requirements

1 Introduction

Representing software requirements is a largely treated topic in literature, and a variety of methods, techniques and approaches have been proposed and applied in different domains. Among them, 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[1]. However, some common problems may emerge even while trying to describe requirements by means of use cases. Ambiguities, incompleteness, and inconsistencies may in fact cause difficulty in requirements comprehension and, consequently, defects in the software system under development.

Screen mockups (also known as user interface sketches, user interface prototypes or wireframes) can be instead used for improving the comprehension of functional requirements, for prototyping the user interface of a subject system[2-3], and for representing the non-functional requirements concerning the user interface[4].

However, a drawback in enriching the use cases with the screen mockups is the burdensome task of guaranteeing the consistency between the graphical representation of the screen mockups and the textual descriptions of the use cases.

As a matter of fact, the consistency between screen mockups and use cases cannot be guaranteed if the former are not disciplined by a structure or some rules. Requirements specifications based on use cases may in fact be scarcely structured, (e.g., composed of lists of freely formed natural language sentences), or they may be presented as quite detailed and structured templates (for example, Cockburn[1]), or even expressed through UML models[5] or formal specifications[6].

Regular Paper
Special Section on Software System 2018
*Corresponding Author

We believe that a good compromise can be represented by disciplined natural language specifications, where the text must follow very detailed and stringent patterns[7-8] and the screen mockups are used to clarify the steps of the scenarios. For this reason, we have conceived a method for describing and refining requirements specifications based on disciplined use cases and screen mockups: DUSM (Disciplined Use Cases with Screen Mockups). The term disciplined means that a use case is: 1) characterized by a stringent template and complemented with a glossary to reduce ambiguities, 2) aligned with the screen mockups that will help in functionalities understanding, and 3) able to help the requirements analyst 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 a list of well-formedness constraints.

Screen mockups are quite common in many IT (information technology) companies and several proposals are emerging to integrate/use them in conjunction with use cases (or, more in general, with requirements)[9-10].

To the best of our knowledge, DUSM is the first method that fully and precisely integrates screen mockups with textual requirements. Indeed, in our method screen mockups are not only adornments for textual requirements specifications, but artifacts 1) made consistent with the specifications by following a set of well-formedness constraints, and 2) checked against a list of possible bad smells that can lead to the detection of ambiguities, inconsistencies and incompleteness in the specifications. Another important aspect is that the design of screen mockups is based on screen mockup templates, which uniquely represent all the different and main aspects of a system GUI (graphical user interface). Hence, there is a reduction in the effort needed to produce the screen mockups, since it depends on the number of the screen mockup templates instead on the higher number of the use cases steps.

This paper extends and refines our previous preliminary work[11-12] in several directions. Indeed, here we add: 1) a comprehensive method for describing and refining requirements specification, 2) the definition of the form of DUSM requirements specifications by means of a metamodel, 3) the list of well-formedness constraints and bad smells for those specifications, and 4) the validation of DUSM by means of a case study.

The paper is organized as follows. Section 2 describes the process adopted by DUSM for specifying and refining requirements. Section 3 and Section 4 describe our requirements specification based on disciplined use cases, and how to integrate screen mockups, respectively. Section 5 describes the ACME case study, while Section 6 reports the empirical assessments of DUSM. Finally, Section 7 presents the related work, followed by conclusions and future work in Section 8.

2 Building Requirements Specification with DUSM

The starting point of DUSM is what we call a free use cases specification (see FUCspec in Fig.1), i.e., a use cases specification based on whatever template (for example, the one proposed by Cockburn[1]), in general allowing a lot of freedom. In case not yet available, the free specification may be easily produced by stakeholders or domain experts with or without the assistance of the analyst.

Once the free specification has reached a stable form, the analyst may render the use cases disciplined (see DUCspec in Fig.1), design the screen mockup templates[1], derive from them all the needed screen mockups, and add them to the use cases (see DUSM-spec in Fig.1). Finally, (s)he will verify that all the well-formedness constraints advocated by DUSM are verified. The result of such activity is (often) the detection of problems in the specification that can be classified in: inconsistencies (i.e., two different points of the specification express two contrasting statements about something), ambiguities (e.g., the specification uses words without stating their precise meaning relying on some common, but not always shared, understanding), and incompleteness (i.e., it is not possible to have a clear understanding of how the system should work because some parts are not properly specified). Notice that even the addition of the screen mockups may generate many questions about the system under specification and reveal undetected problems. In all these cases, the analyst should ask the stakeholders and/or the domain experts additional details to obtain a better specification, following a reworking phase of the previously produced one, where all the relevant changes are implemented, such as adding/removing/refining elements (e.g., some steps may need to be restructured, hence requiring further screen mockups).

[1]A screen mockup template is a way for enforcing a standard layout and look and feel across multiple screen mockups generated starting from it (see Subsection 4.3).

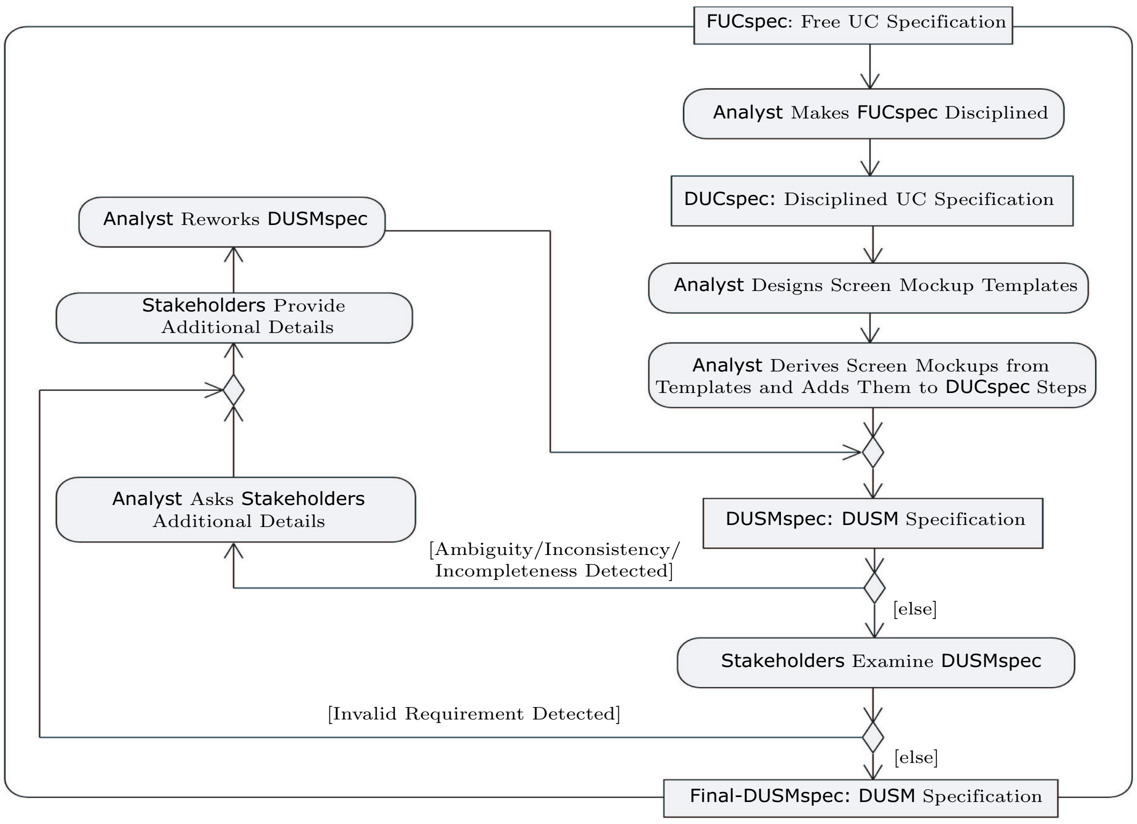

FUCspec: Free UC Specification
Analyst Makes FUCspec Disciplined
DUCspec: Disciplined UC Specification
Analyst Designs Screen Mockup Templates
Analyst Derives Screen Mockups from Templates and Adds Them to DUCspec Steps
DUSMspec: DUSM Specification
[Ambiguity/Inconsistency/ Incompleteness Detected]
[else]
Stakeholders Examine DUSMspec
[Invalid Requirement Detected]
[else]
Final-DUSMspec: DUSM Specification
Analyst Asks Stakeholders Additional Details
Stakeholders Provide Additional Details
Analyst Reworks DUSMspec

Fig.1. DUSM method as a UML activity diagram (actions are round-cornered rectangles, while object nodes are rectangles).

Once the analyst has terminated her/his work, the resulting disciplined specification enriched by screen mockups (i.e., DUSMspec) may be given to the stakeholders to get the final approval. They will have no problem in reading and understanding it, since it is essentially structured natural language text. Moreover, the presence of the screen mockups provides a kind of paper prototyping, allowing them to validate also the user interface. Any change request may be easily processed by the analyst, because the strong structuring of the DUSM specification offers a good support to propagate the changes on the whole specification. This characteristic combined with a better understanding of the entire specification will be valuable also in case of future evolution of the system requirements.

3 DUSM: Disciplined Use Cases Specification

Fig.2 shows the form of the DUSM requirements specifications by means of a metamodel presented by a UML class diagram. A requirements specification consists of a UML use case diagram, a description of each use case appearing in that diagram, and a glossary that lists and makes precise all the terms used in the use cases. We have included the use case diagram since it is really valuable for summarizing use cases, actors and their mutual relationships. Moreover, it is also quite simple to be explained and produced. It is important to highlight that the components of our requirements specification are all well-known (e.g., use case diagram). What is novel here is how they are combined together (e.g., use cases and screen mockups) by means of rules and constraints.

Even though some of the parts of a requirements specification shown in Fig.2 are well-known, for the sake of completeness, in the following subsections we briefly sketch them, using the ACME case study (shortly described below) as a running example. In this way, we will be able to better show the novel aspects of DUSM, and explain the meaning of the constraints given on the components of the requirements specification. The complete requirements specification can be found in ACME documentation[13] and is reported as follows.

ACME is a multi-users resource management system organized in functional areas, where users are assigned to tasks called jobs and can perform some actions, based on their access levels to the system functional areas, to complete them. An access level (hidden for a not acces-

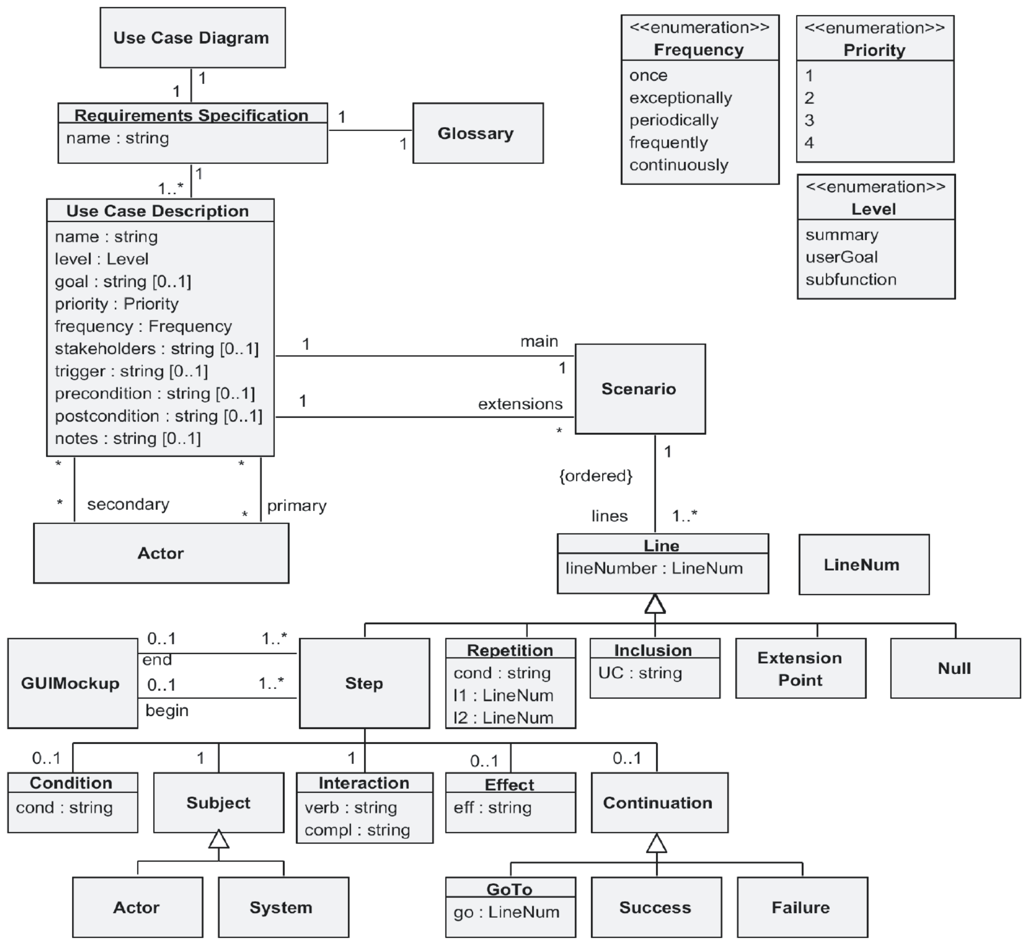


Fig.2. DUSM requirements specification metamodel.

sible functional area, view for a read-only one, and edit for read-write permissions) is linked to a user through the unique group (s)he is assigned to. ACME works on two different levels of security: enabled and disabled. When security is enabled, each user must authenticate (with username and password) to perform any task. In case security is disabled, logging is not needed. A group in ACME can be thought of as a role which establishes all possible operations that can be completed. ACME presents two default groups: Administrator and External User. Each of these groups has different access levels for each of the ACME functional areas. An Administrator can CRUD (create, read, update, delete) any group, job and user and is responsible for the users assignments to groups. Instead, an External User can perform some actions depending on the access levels assigned to him/her; different from the Administrator group, access levels for External User are not given by default. Every group and user is identified by a unique name. Among its attributes, a job is characterized by: a short description, a creator, and a timestamp of creation/update. An example of job may be the reservation of a room or the reporting of a bug, depending on which context ACME is applied. The user who owns the rights (i.e., the access level for a functional area) could act like an Administrator for a specific job.

In this paper we use System to denote a generic software system of whom we want to specify the requirements. In our running example, the system is referred to as ACME.

3.1 Use Case Diagram

The use case diagram summarizes the System use cases, making clear which actors take part in them. The actors are distinguished in: primary, those having goals on System, i.e., entities obtaining value from interacting with the System, and secondary, those over which the System has goals, i.e., entities supporting System in creating value for primary actors. Sticky-man icons are used for visually representing the primary actors, and boxes are used instead for the secondary actors. Use cases are classified with respect to their granularity level: *summary* (representing a goal of the System), *userGoal* (representing functionalities from the user perspective), and *subfunction* (moving out an isolated part of a scenario to a separate use case). The granularity of a use case is depicted in the use cases icon (i.e., ellipses) of the UML use case diagram by means of three corresponding stereotypes. However, to improve readability, it can be omitted in case all the use cases have the *userGoal* granularity level. *Inclusion* and *extension* relationships between use cases[2] may appear in the use case diagram, as well as specialization relationships between actors[3], represented by the classic arrow with closed head.

A fragment of the use case diagram part of the ACME requirements specification is shown in Fig.3 (the complete version, amounting to 31 use cases, can be found in the ACME specification[13]). In the case of ACME, there are three primary actors (Administrator, External User, and User), and no secondary actors; notice that both Administrator and External User specialize User, thus both may take part in all the use cases of User. The level of all the use cases is *userGoal*, but for improving the readability of the diagram the corresponding stereotype is omitted.

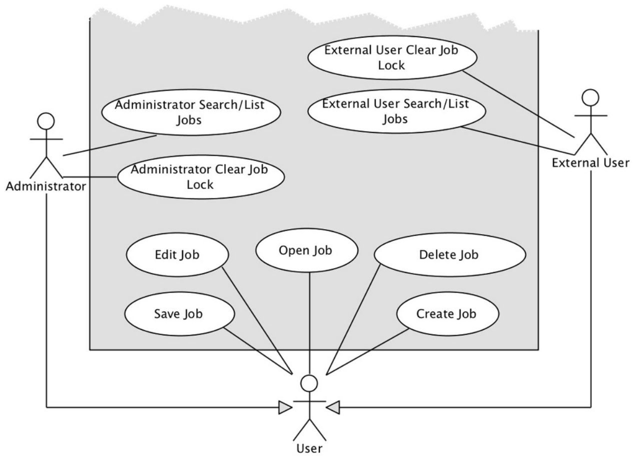

External User Clear Job Lock
Administrator Search/List Jobs
External User Search/List Jobs
Administrator Clear Job Lock
Administrator
External User
Edit Job
Open Job
Delete Job
Save Job
Create Job
User

Fig.3. ACME requirements specification: use case diagram fragment.

The list of well-formedness constraints and bad smells related to use case diagrams is shown in Fig.4.

3.2 Glossary

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 entries (e.g., credit card data, order information), and those about the attributes abstractly describing the state of System, indicated as system attributes (e.g., names of the registered clients, items currently in the catalogue). The main objectives of the glossary are to: 1) shorten use cases, 2) reduce ambiguities (e.g., to avoid using different ways to refer to the same entity), and 3) clarify the meaning of the steps of a scenario (e.g., a richer description of the various entries could be given). The well-formedness constraints and the bad smells that must be considered for glossary are shown in Fig.5.

Fig.6 shows a fragment of the glossary part of the ACME requirements specification, where the entry names are written using a specific font (the complete one, amounting to 28 entries, can be found in [13]).

References to a glossary entry, in the form of entryName*, can be used in the definitions of other entries (see, e.g., Job Info* in the definition of Jobs), in the steps of the use cases scenarios, as well as in the other parts of the use cases descriptions. References to glossary entries are case-insensitive and can be replaced by a declension in order to adapt them to the sentences.

3.3 Use Case Description

A use case description consists of general information (e.g., name, level and goal), plus a set of scenarios (see Fig.2). The main success scenario describes the basic execution of the use case, whereas the extensions (any number, also none) are scenarios defining all the other possible executions of the use case.

The information about a use case description[4] are as follows.

• *Name*: a verbal phrase in the form of present infinite without "to", that identifies the use case (e.g., Open Job). Sometimes it may be helpful to explicit the

[2]Inclusion specifies that one use case includes the functionality of another use case mainly for reuse purposes, while the extension relationship specifies that one use case (extension) extends the behaviour of another use case.

[3]If actor A1 specializes actor A2, then A1 can take part also in all the use cases in which A2 takes part.

[4]There are some differences between the disciplined use cases and the use cases proposed by Cockburn[1]. For example, we adopt a glossary and precisely define use case steps format to permit consistency checks (see Section 3). However, the exact list of the differences is not essential to understand the main contribution of the paper, consisting in a smooth and consistent integration among use cases and screen mockups.

Well-Formedness Constraints
• A use case cannot be included in a lower granularity level use case.
WHY: *Violation of the meaning of the levels*
• A subfunction use case must be included at least in a userGoal use case.
WHY: *Violation of the meaning of the subfunction level, since it cannot be a complete functionality*
• There must be at least a userGoal use case.
WHY: System *must offer at least a functionality*
• The transitive closure of the inclusion relationship among use cases must be anti-reflexive.
WHY: *There cannot be undefined use cases*
• The transitive closure of the extension relationship among use cases must be anti-reflexive.
WHY: *There cannot be undefined use cases*
• The transitive closure of the specialization relationship among actors must be anti-reflexive.
WHY: *There cannot be undefined actors*

Bad Smells
• A subfunction use case included only once in another higher level use case is suspicious.
WHY: *This case is sensible if it has been introduced only for shortening the scenarios of another use case*
• A summary use case not including any other use case is suspicious.
WHY: *If the use case is that simple, it should have the userGoal level*

Fig.4. DUSM use case diagram: well-formedness constraints and bad smells.

Well-Formedness Constraints
• The name of an entry in the glossary must be unique.
WHY: *It must be possible referring it in the use cases descriptions*
• Each entry in the glossary must appear at least once in a use case description.
WHY: *Otherwise, it is useless and should be eliminated*

Bad Smells
• A glossary not containing system attributes is suspicious.
WHY: System *not using any persistent data would provide only trivial functionalities, e.g., a converter from different measure units*
• A glossary not containing data entries is suspicious.
WHY: System *using only basic data as numbers and strings would provide only trivial functionalities, e.g., a converter from decimal to roman numbers*

Fig.5. DUSM glossary: well-formedness constraints and bad smells.

Data Entries
• Access Level: defines the permissions to access the functional areas* of ACME. It can be: *hidden* for a non-accessible area, *view* for a read-only one, and *edit* in case of read-write permissions.
• Group Name: unique identifier built out of up to 32 characters (letters, numbers, separators such as dash or underscore, and blank spaces among them)
• Job Info: the data characterizing a job (short and long description, creator, title, issue, type, related module, creation/update timestamps)
• Search Criteria: the information given by a user to find some jobs (short description, creator, creation/modification date range, type, title, module, issue)

System Attributes
• Administrator Logged: a boolean value, true iff there is a logged administrator
• Jobs: all the existing jobs characterized by Job Info*

Fig.6. ACME requirements specification: glossary fragment.

primary actor in the use case name, especially when several use cases share the same name but involve different primary actors and scenarios (see for example Fig.3).

• *Level*: i.e., summary, userGoal, or subfunction as introduced in Subsection 3.1.

• *Goal*: describes in a detailed way the aim of the use case (optional).

• *Priority*: expresses the impact that an incorrect or lacking implementation of the use case has on the System. We experimented that four values are sufficient, and thus the allowed values range from 1 (higher) to 4 (lower). 1 = System is no longer operative and no other ways to perform the supported activities exist; 2 = System is no longer operative, but the supported activities can be manually performed; 3 = System is operative, but one or more main functionalities are not available; 4 = System is operative, but one or more secondary functionalities are not available.

• *Frequency*: expresses how much frequently the functionality described by the use case will be used. We decided that the possible values are: *once*, *exceptionally*, *periodically*, *frequently*, and *continuousty.*

• *Stakeholders*: those having the right or the possibility to say something about the functionality described by the use case (optional).

• *Trigger*: defines which event starts the use case (optional).

• *Primary and Secondary Actors*: introduced in Subsection 3.1.

• *Pre/Post Condition*: states what we assume about the current state of the System before the execution/after the successful execution of the use case (optional). They should be expressed using the System attributes introduced in the glossary.

• *Notes*: comments to the use case, usually to record why something is made in some way (optional).

The list of well-formedness constraints on the use case description is shown in Fig.7.

Fig.8 presents a simple use case description; it refers to the use case Administrator Search/List Jobs of the ACME system. Its level is userGoal, and its goal is to allow the primary actor Administrator to examine the existing jobs. The information part is quite standard and does not need a detailed comment, and this use case has just one extension. Underlined terms represent hyperlinks to screen mockups, while a term followed by * refers to an item of the glossary, partially shown in Fig.6. More complex use cases can be found in the ACME requirements specification[13].

3.4 Scenarios

The abstract structure of the scenarios is presented in Fig.2 (see Scenario class and associated classes). In particular:

Scenario: a sequence of numbered and ordered *lines*, where each *line* is either a step, an indication of a repetition of some lines, an inclusion of another use case, an extension point, or even null, which means a line corresponding to doing nothing.

Line Number: allows to uniquely identify each line of a scenario.

Step: describes an interaction between the System and one of the actors of the use case. It has the form:

[If *condition*, then] *subject interaction*;

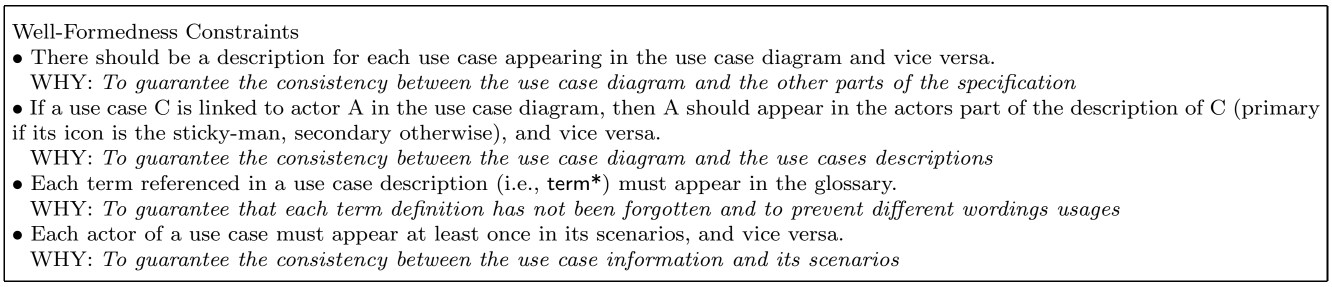
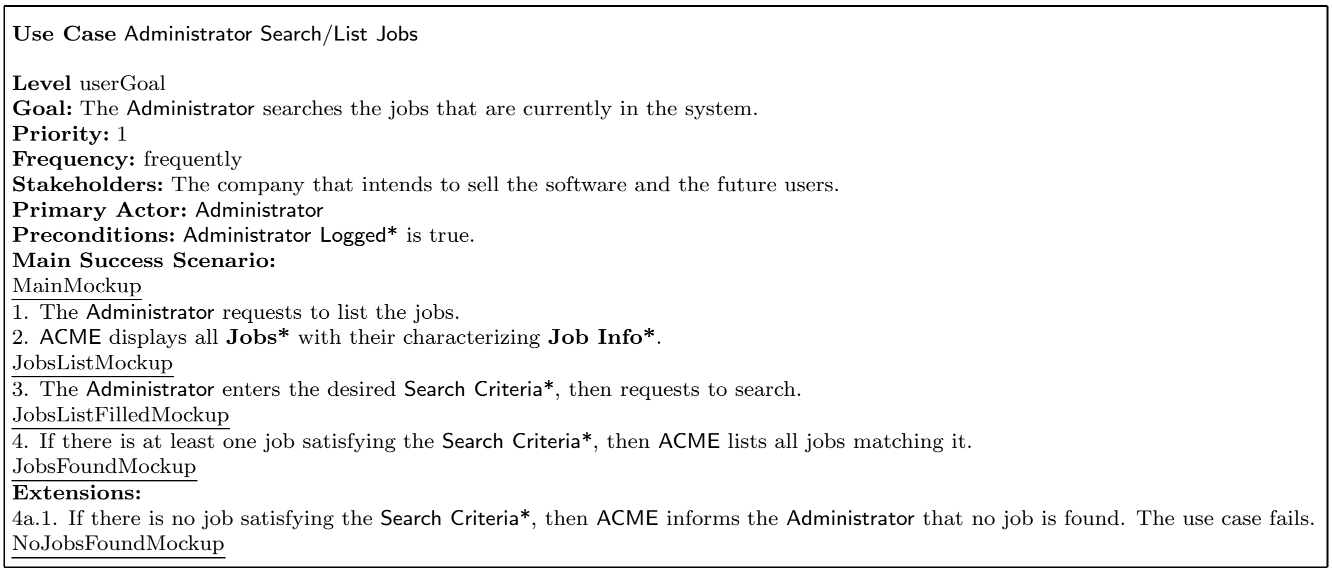

Well-Formedness Constraints
• There should be a description for each use case appearing in the use case diagram and vice versa.
WHY: *To guarantee the consistency between the use case diagram and the other parts of the specification*
• If a use case C is linked to actor A in the use case diagram, then A should appear in the actors part of the description of C (primary if its icon is the sticky-man, secondary otherwise), and vice versa.
WHY: *To guarantee the consistency between the use case diagram and the use cases descriptions*
• Each term referenced in a use case description (i.e., term*) must appear in the glossary.
WHY: *To guarantee that each term definition has not been forgotten and to prevent different wordings usages*
• Each actor of a use case must appear at least once in its scenarios, and vice versa.
WHY: *To guarantee the consistency between the use case information and its scenarios*

Fig.7. DUSM use case description: well-formedness constraints.


Use Case Administrator Search/List Jobs

Level userGoal
Goal: The Administrator searches the jobs that are currently in the system.
Priority: 1
Frequency: frequently
Stakeholders: The company that intends to sell the software and the future users.
Primary Actor: Administrator
Preconditions: Administrator Logged* is true.
Main Success Scenario:
MainMockup
1. The Administrator requests to list the jobs.
2. ACME displays all **Jobs*** with their characterizing **Job Info***.
JobsListMockup
3. The Administrator enters the desired Search Criteria*, then requests to search.
JobsListFilledMockup
4. If there is at least one job satisfying the Search Criteria*, then ACME lists all jobs matching it.
JobsFoundMockup
Extensions:
4a.1. If there is no job satisfying the Search Criteria*, then ACME informs the Administrator that no job is found. The use case fails.
NoJobsFoundMockup

Fig.8. ACME Administrator Search/List Jobs disciplined use case with linked screen mockups.

[*effect*] [*continuation*].

Here

• [...] means that ... is optional.

• *condition* is a natural language fragment stating the condition under which the step may be executed. It is optional. If absent, it is intended as the always true condition. It should concern the System attributes (i.e., the current state of System), and the data appearing in the interaction and effect part of the current step or of the previous ones; 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. The System should be indicated with the same name used in the system box in the use case diagram.

• *interaction* is a sentence describing either what flows from the actor towards the System or vice versa. It must have the form "verb complements". The complements may be System, an actor, System attributes, and the data appearing in the previous steps.

• *effect* is a sentence written in the passive form without explicit "by clause" (e.g., omit "by ACME" in the effect "a new user characterized by User Info* is added to Users* by ACME") describing a transformation of the System attributes using the data appearing in the interaction; thus, again it will be written using the terminology introduced in the glossary. It is optional. If absent, then the step does not influence the System attributes.

• *continuation* defines how the use case flow continues after the end of the step. It may have one of the following forms:

– "The use case continues to *lnum*", where *lnum* is the number of a line of the use case (see GoTo class in Fig.2).

– "The use case fails." and "The use case ends with success." for marking the end of the use case, distinguishing whether it is a success or a failure. For the sake of readability, "The use case ends with success." is usually omitted.

The continuation is optional. If absent, then the use case continues to the next line.

Repetition: a natural language fragment having form: "The lines from $lnum_1$ to $lnum_2$ are repeated until *cond*", where $lnum_1$ and $lnum_2$ are the numbers of two lines of the scenario including the repetition, and *cond* is a condition like the one that is part of a step. Repetitions allow to describe scenarios of unbound length.

Extension Point: denotes where the behaviour of an extending use case (i.e., a use case related by the extension relationship in the use case diagram) will be inserted.

Inclusion: written by reporting the name of the included use case, that should be linked to the described use case by the inclusion relationship in the use case diagram.

Null: a line where nothing is done (i.e., neither an actor nor the System does something); it is needed for representing particular flows of activities. For example, in ACME a user could optionally choose to view some details of a job while performing some other activities; since both the alternatives of viewing job details and not viewing them (i.e., do nothing) may affect ACME response to the user in different ways, two scenarios have been modelled.

Extensions: of a use case, see Fig.2, are scenarios defined modifying an existing one, by giving a different sequence of lines starting from a given line (the extended line), where the lines of a use case are identified by *line numbers*. The lines of the main success scenario are labelled by natural numbers (i.e., 1, 2, 3, ...). The lines of the first scenario extending a line whose number is X are numbered with $Xa.1$, $Xa.2$, ..., those of the second scenario $Xb.1$, $Xb.2$, ... and so on.

For what concerns scenarios well-formedness constraints and bad smells, please refer to Fig.9.

4 DUSM: Screen Mockups

4.1 Screen Mockups Associated with Use Case Steps

Screen mockups are drawings that show how the user interface of System is supposed to look during the interaction between the System and the human actors. They 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 need to be expressed (e.g., requiring to use specific logos or brand related colours)[3]. Mockups can be used in conjunction with use cases, associating them with the steps of the scenarios, to improve the comprehension of functional requirements, and to achieve a shared understanding on them.

At the same time, screen mockups allow to express and improve the comprehension of the non-functional requirements concerning the user interface[4], thanks to the freedom in choosing the most effective way to represent them.

Well-Formedness Constraints

• The subject of a step of a scenario different from System must appear among the use case actors.
WHY: *To guarantee the consistency between the information part of a use case and its scenarios*
• Each System attribute listed in the glossary must both:
– be updated in the effect part of at least a step of a use case scenario,
– be read either in the condition or in the interaction or in the effect of at least a step of a use case scenario.
WHY: *To guarantee the minimality of the specification. If it is neither read nor updated it is useless, and thus it should be removed. If it is updated and never read, it is either useless or the steps/use cases reading it are lacking. If it is read but never updated the steps/use cases modifying it are lacking*
• If a step has a condition *cond* different from true, then there should be some extensions starting from the same step with conditions $cond_1$, ..., $cond_n$ s.t. the logical disjunction of *cond*, $cond_1$, ..., $cond_n$ is true. It is possible to use the Null line in the case nothing is done for a certain condition.
WHY: *To specify what* System *should do in all possible cases*
• If a use case C includes C1 in the use case diagram, then at least a line corresponding to "include C1" must appear in the scenarios of C, and vice versa
WHY: *To guarantee the consistency between the use case diagram and the use cases descriptions*
• If a use case C extends C1 in the use case diagram, then at least a line corresponding to an extension point for C must appear in the scenarios of C1, and vice versa
WHY: *To guarantee the consistency between the use case diagram and the use cases descriptions*
• The line number appearing in a GoTo continuation of a step (see Fig.2) should refer to an existing line in the use case scenarios.
WHY: *There cannot be dangling GoTo references*
• The line numbers appearing in a repetition line should refer to existing lines in the same scenario.
WHY: *There cannot be dangling repetition references*
• The success and failure continuation may appear only at the end of complete scenarios.
WHY: *To express the final outcome of the various use cases executions*
• The failure continuation cannot appear in the main success scenario.
WHY: *The main success scenario must describe a successful way to execute the use case*

Bad Smells

• A complete scenario without at least a step where the subject is System is suspicious
WHY: *The* System *must provide some feedback to a request of an actor, unless the actor is not human*
• The case where the initial steps of a set of extensions to the same line do not have the same subject is suspicious.
WHY: *Usually such cases correspond to awkward behaviours of the specified* System.
• An extension of a step S starting with S_1 s.t. the subject, the interaction and the effect of S_1 coincide with those of S is suspicious
WHY: *S and S_1 must be joined in a unique step by combining their conditions*

Fig.9. DUSM use case scenario: well-formedness constraints and bad smells.

DUSM suggests to associate one or two screen mockups with each step of a scenario (see Fig.2), where a human actor is involved. Fig.10 shows two simple screen mockups associated with some steps of the Administrator Search/List Jobs use case (presented in Fig.8) of the ACME application.

More precisely, let S be a step of a scenario:

• If the subject of S is System, then at most one mockup (*end mockup*) may be associated with the end of S; it will show what can be seen on the System's GUI after the step execution.

• If the subject of S is a human actor, then at most two mockups may be associated with S:

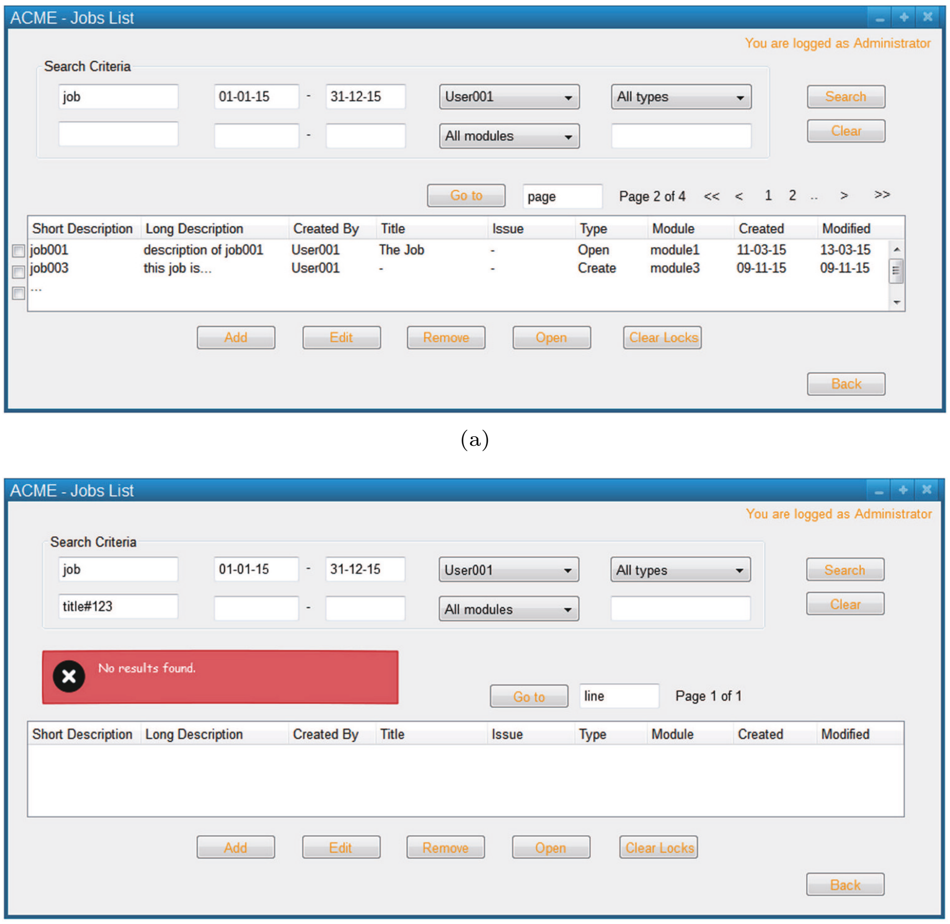
– one at the beginning of S (*begin mockup*), which will show what the actor sees just immediately before to execute the step;

– one at the end of S (*end mockup*), which will show what the actor sees just immediately before to complete S (e.g., before pressing the "send" button after having filled various fields/having ticked some check boxes/opened some menus).

The begin mockups of the steps corresponding to extensions starting from the same step, obviously, must coincide.

A single mockup may be associated with various steps; examples are the begin mockup of different extensions, and the end mockup of a step coinciding with the begin mockup of the next step. See, for instance, JobsListMockup in Fig.8 that is shared between steps 2 (as end mockup of a system's action), and 3 (as begin mockup of an actor's action). A single mockup can also appear in different use cases, as in the case of the one corresponding to the main window of a system.

Any screen mockup associated with a step must be consistent with it, i.e., it should present the same informative content, otherwise the introduction of the mockups will be the cause of further ambiguities in the requirements specifications, instead of improving their quality. An example of inconsistency is the linkage of a screen mockup containing exactly two text boxes to a step having form "Insert name, birth date and sex". Similarly, the mockup associated with "System informs


ACME - Jobs List

You are logged as Administrator

Search Criteria

job | 01-01-15 - 31-12-15 | User001 | All types | All modules

Go to | page | Page 2 of 4 << < 1 2 .. > >>

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
job003	this job is...	User001	-	-	Create	module3	09-11-15	09-11-15
...								

(a)

ACME - Jobs List

You are logged as Administrator

Search Criteria

job | 01-01-15 - 31-12-15 | User001 | All types | title#123 | All modules

No results found.

Go to | line | Page 1 of 1

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified

(b)

Fig.10. ACME screen mockups. (a) JobsFoundMockup. (b) NoJobsFoundMockup.

that a username must contain capital letters and digits only" cannot present a popup showing just a generic "Error" message. DUSM provides detailed constraints (see Fig.11) that guarantee the consistency between the steps and the associated mockups. These constraints may be also considered as guidelines to help the production of the mockups.

However, the given constraints do not lead to a standard straightforward way to design the screen mockups to add to the steps. Clearly, if three entities are mentioned in a step, e.g., name, birth day and sex, the associated mockup should contain three widgets, which may be textboxes, drop menus, checkboxes, buttons and so on. Moreover, the layout and the aspect of the mockup is entirely left to the analyst. This freedom in the design of the mockup(s) associated with a step allows to express the non-functional requirements on the application GUI.

For example, referring to the ACME case study, Fig.12(a) shows the main window, i.e., the begin mockup of many steps starting the main functionalities of the application, as we designed it. It is quite simple and offers buttons for calling all such functionalities. If we require that, instead, the interface should adhere to an existing theme, we could draw the mockup as Fig.12(b), where functionalities are accessible through menus, using a different colour theme and a different font.

Placeholders for the begin/end mockups will be inserted before or after the steps in the use case scenarios respectively. Obviously, whenever the begin mockup of a step coincides with the end mockup of the previous step, its placeholder will appear 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 hyperlink in an HTML document). In Fig.8, for example JobsFoundMockup and NoJobsFoundMockup are links to the pictures reported in Fig.10.

The use of the placeholders allows the readers of the

Well-Formedness Constraints

Steps with actor subject – Begin mockup

Let $S_1, \ldots, S_n$ ($n \geqslant 1$) be all the steps in the specification having an actor as subject s.t. their begin mockup is M ($S_1, \ldots, S_n$ may appear in different scenarios of even different use cases).

• If the interaction part of some S_i ($1 \leqslant i \leqslant n$) refers to some communication from the actor to System, then M should show how it has been realized.

e.g., $n = 2$, S_1="User confirms", S_2 ="User refuses". Thus, "Confirm" and "Refuse" buttons appear in M;
S_i ="User enters email & password". Thus, email & password fields and "Enter" button appear in M.

• If M contains some means for realizing some communication from the actor to the System, then there should exist $1 \leqslant i \leqslant n$ s.t. S_i refers to such communication.

e.g., M contains a "Confirm" button. Thus, the interaction part of S_i, for some i, should speak of confirmation;
M contains a "password" field. Thus, the interaction part of S_i, for some i, should speak of a password.

Steps with actor subject – End mockup

Let $S_1, \ldots, S_k$ ($k \geqslant 1$) be all the steps in the specification having an actor as subject s.t. their end mockup is M ($S_1, \ldots, S_k$ must be steps having the same interaction part appearing in different scenarios of even different use cases).

• If the interaction part of S_1 (that is coincident with those of $S_2, \ldots, S_k$) refers to some communication from the actor to System, then M should show how it has been realized.

e.g., S = "User selects advanced mode". Thus, an open menu "advanced mode" is shown in M;
S = "User selects the amount to deposit". Thus, a text box filled with a given amount is shown in M;
S = "User inserts the code received by SMS". Thus, a text box about the code is shown filled in M.

• If the interaction part of S_1 (that is coincident with those of $S_2, \ldots, S_k$) includes a reference to some specific information (flowing from the actor to System), then such information must appear in M.

e.g., S = "User inserts the password". Thus, the password appears in M;
S = "User selects category and number of nights". Thus, both the category and the number appear in M.

• If M shows how some communication is going to be realized (from the actor to System), then the interaction part of step S_1 (that is coincident with those of $S_2, \ldots, S_k$) should refer to it.

e.g., M contains a filled "password" field. Thus, the interaction part of S_i, for some i, should describe the insertion of password.

Steps with System subject – (End) mockup

Let $S_1, \ldots, S_m$ ($m \geqslant 1$) be some steps in the specification having System as subject s.t. their end mockup is M ($S_1, \ldots, S_m$ should be steps having the same interaction part appearing in different scenarios of even different use cases).

• If some information appears in M, then it should be derived by the interaction parts of the previous steps or by System attributes.

e.g., "You are logged as John Doe" message is shown in M. Thus, the name of "John Doe" is recoverable by System attributes or it is provided by the user in some previous step.

• If the interaction part of S_1 (that it is coincident with those of $S_2, \ldots, S_m$) refers to some communication from System to actor, then M should show how it has been realized.

e.g., S = "System lists all the logged users". Thus, "The current logged users are:" string appears in M together with a list of users;
S = "System confirms deletion". Thus, a pop-up containing "deletion confirmed" appears in M.

WHY: *To guarantee the consistency between the use cases descriptions and the screens mockups*

Bad Smells

• Having several screen mockups in the extensions and none in the main success scenario is suspicious.
WHY: *It does not present user interface requirements in a systematic way; it is acceptable only in the case the mockups for the main scenario are obviously similar to those of another use case*

• Having few mockups distributed in almost every use case with human actors is suspicious.
WHY: *In this way it is difficult to grasp a coherent set of requirements on the GUI. It is better to concentrate the effort and thus adding all the mockups to a small number of use cases*

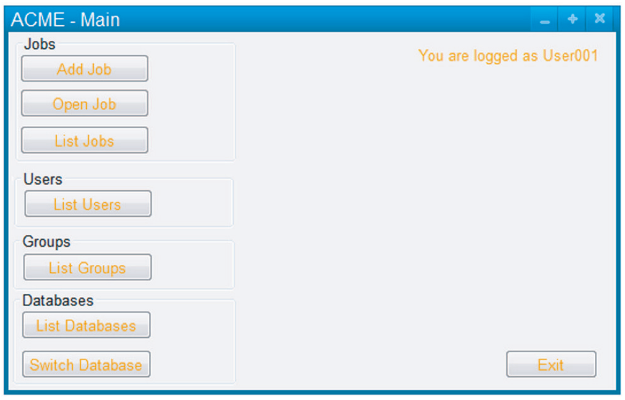
Fig.11. DUSM screen mockup: well-formedness constraints and bad smells.

use case to choose whether to examine the screen mockups or ignore them, whenever they are 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 ACME specification[13]) corresponding to the so-called "paper prototype" of System.

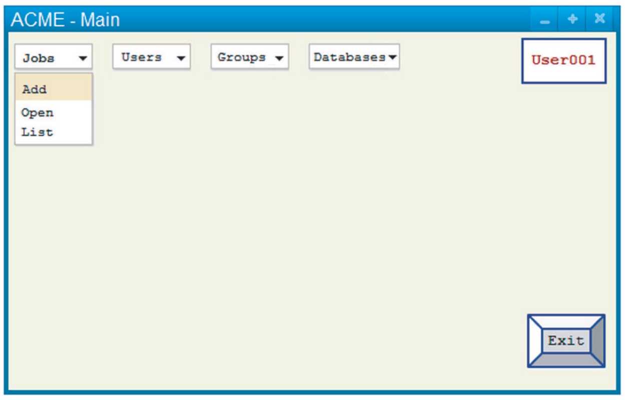
4.2 How to Produce the Screen Mockups

Although a number of tools for drawing screen mockups exist (see Table 1 for a partial list), several professionals prefer to sketch screen mockups on paper. This kind of approach has some drawbacks, which are mainly related to the continuous evolution of requirements[14]. Mockups created with computer drawing tools mitigate this last concern, thus making them a viable alternative.

Screen mockups could be also built using HTML, a programming language (e.g., Java), or an IDE (e.g., NetBeans, VisualStudio). These last choices have the benefit to reuse the source code of screen mockups later in the development phase and to obtain quite realistic mockups, while the main drawbacks concern the effort and the skills needed to build and maintain them: any kind of stakeholders (even without development experience) should be able to build mockups spending a few minutes[15-16].



(a)



(b)

Fig.12. Two different screen mockups for ACME main window.

Therefore, the use of specific tools for drawing screen mockups, e.g., Pencil, represents a viable trade-off between sketching screen mockups on paper and implementing them. The screen mockups shown in Fig.10 and Fig.12, and all the mockups appearing in the requirements specifications of the ACME[13] and AL_L (a system that handles the ALgebraic Lotteries[11]) case studies have been created using Pencil.

It is important to highlight that DUSM does not force to add all the possible mockups to all steps of the scenarios of all the use cases (see the multiplicities on the associations linking the mockups to the steps in Fig.2); it is left to the analyst the decision to omit those not conveying any useful information (e.g., mockups associated with steps where the user interaction is trivial or too similar to already inserted ones). Furthermore, the relevance of the various use cases (expressed by the priority and frequency attributes) helps to select which ones should be enriched with screen mockups; for example, it is of scarce utility to produce the mockups for a use case whose frequency is "Once", whereas it is almost mandatory in case of "Continuously". Similarly, it is obviously better to add mockups to use cases with priority 1 than to add those with priority 4.

4.3 Relations Among Screen Mockups

The many mockups associated with use case scenarios are not a bunch of totally unrelated items. Quite naturally, they are organized in a forest-like structure where there is an arc from M_2 to M_1 iff M_2 is derived from M_1, i.e., it has been produced by modifying M_1 (e.g., when a menu or an auxiliary window is opened, a text-box is filled, or a button become dimmed). For example, in use case Administrator Search/List Jobs shown in Fig.8, the screen mockup named JobsListFilledMockup is derived from JobsListMockup by filling some text fields. Technically, we say that there is a dependency relationship between the mockups, assuming the classical definition: M_2 depends on M_1 iff a modification in the latter leads to a modification in the former (e.g., if the layout is modified in M_1, then also M_2 should be modified).

To produce each tree in the mockups forest requires

Table 1. Partial List of Mockup Drawing Tools

Tool	URL
Balsamiq Mockups	https://balsamiq.com
Pencil Project	https://pencil.evolus.vn
OverSite	http://taubler.com/oversite
GUI Design Studio	https://www.carettasoftware.com
Axure	https://www.axure.com
Moqups	https://moqups.com
MockFlow	https://www.mockflow.com
Mockingbird	https://gomockingbird.com/home
SketchFlow	https://www.microsoft.com/silverlight/sketchflow
BlueGriffon	http://www.bluegriffon.org
Visual Paradigm UeXceler	https://www.visual-paradigm.com/training/agile-development-with-uexceler/

to express some (non-functional) requirements on GUI, e.g., by deciding the layout and the colour theme of a window. Thus, before to produce the mockups, all the main requirements on GUI should be expressed by providing a template for each type of GUI that will be used, to later be able to produce the trees in the forest. Therefore, a template is just a screen mockup that uniquely represents a portion of the system GUI; it can be seen as a node of a tree where the children of that node are all the variations of their parent. All the templates should be obviously coordinated: for example, it is better to avoid to signal an error in a mockup using a popup while in another one using a message bar at the bottom of a window.

Therefore, to proceed to build the mockups, DUSM suggests to:

• determine the needed screen mockup templates and organize them with respect to the dependency relationship;

• produce the needed mockups modifying either a template or an already produced mockup.

The suggested way to produce and organize the mockups will be obviously a big help in the case of evolution of the requirements specification (e.g., dependency will help to propagate any change in the requirements on the user interaction). Furthermore, we see that the effort needed to produce the mockups is not depending linearly on the number of the use cases and on the number of their steps, but on the number of the templates, i.e., on the variability of the System's GUIs.

5 ACME Case Study

DUSM has been proposed and fine-tuned during several editions of a software engineering course at the University of Genova[17], where two of the authors of the present paper were teaching. Each year, students had to realize a Java desktop application, whose requirements were given as a use case based specification. First, students had to model a design by means of UML, and then, they had to implement it in Java[17]. Initially, no screen mockups were used, and even if standard requirements on the GUIs were provided (i.e., usability requirements), the use cases often resulted difficult to understand and ambiguous. We discovered that, after the introduction of screen mockups, the amount of misunderstanding about the use cases decreased. Then, we applied DUSM several times, also in real industrial contexts, to specify requirements from scratch and to refine already existing use case specifications.

In this paper, we present the application of DUSM to the already existing free requirements specification of an application named ACME (the object of the study). The goal here is restructuring the ACME specification[18] (from free to disciplined) and, at the same time, answering the following two research questions.

*RQ*1: is the application of DUSM able to reveal and remove inconsistencies/ambiguities/incompleteness from the original ACME free specification?

*RQ*2: is the application of DUSM able to produce a specification that is simpler w.r.t. the original ACME free specification?

The metric number of inconsistencies/ambiguities/incompleteness revealed has been used to answer RQ1, while the percentage of verbosity reduction, clearly preserving the semantic content, has been used to address RQ2. We decided to measure the verbosity reduction in terms of words saved in the disciplined version w.r.t. the free version.

It is important to highlight that RQ1 concerns quality of the entire specification and comprehensibility factors. Indeed, removing inconsistencies/ambiguities/incompleteness from the specification should improve the comprehension as well as the quality of the entire document. Instead, simplicity mentioned in RQ2 mainly correlates with comprehensibility: a short specification is much more comprehensible than a long/verbose one. For the computation of this last measure, we have not considered the introduced novel screen mockups and their content.

5.1 ACME Free Specification: Object of the Study

We believe that the ACME specification is suitable to be used as a case study for the following reasons.

• It has been produced by a professional developer (therefore it does not originate from the academia).

• It is not trivial (the documentation is around 17 000 words and 22 use cases[18]).

• It has been produced following a quite formal template and revised many times (last version is 6.1), also after discussing with the client. Therefore, this free specification should be already of a good quality, and thus a good test for seeing if DUSM helps to improve it; otherwise, the validation may be biased (e.g., starting from a very informal and never revised specification).

• The domain of ACME is easy to grasp, thereby there is no need of know-how of specialized domains to apply the method.

• The use cases steps are complemented with the textual description of GUI; thus ACME is an application where expressing the requirements on the user interactions is relevant.

The use cases in the ACME requirements specification are quite complex, with verbose and tricky scenarios built up by many alternative paths, long steps and references among them (i.e., GoTo continuation jumps). In many cases, the steps of the scenarios are made heavier due to the inclusion of detailed textual descriptions of the associated GUI (see, e.g., Fig.13(a)), and of the data shared among actors and ACME.

Since the approach followed by the author of ACME specification does not consider the specialization between actors, when a use case has two primary actors, it may be either that they cooperate to realize the functionality described in the use case or that they both separately (thus, not simultaneously) take part in the use case. This feature clearly has worsened use cases understandability. Moreover, no use case diagram has been adopted to represent the use cases and their relationships, while only an incomplete and inconsistent use cases list has been provided.

5.2 Application of DUSM to ACME: The Procedure

In accordance with DUSM activities of Fig.1, ACME specification was restructured/refined by performing the following tasks: 1) glossary definition, 2) use cases description refactoring, and 3) screen mockups integration. Moreover, during this restructuring task, the use case diagram was also inferred (as DUSM requires it), partially shown in Fig.3.

The entries in the glossary were recovered from the existing use cases descriptions, focusing on *Notes*, *Pre conditions*, *Post conditions*, and scenarios steps. Each term was classified as *Data Entry* or *System Attribute*; a definition was formulated by looking in many different

...

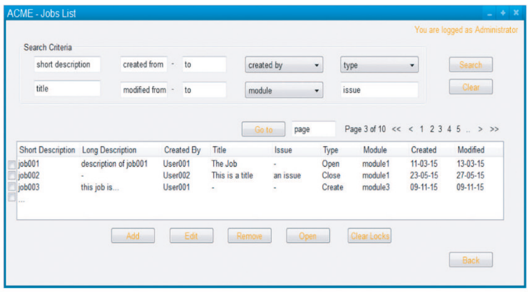
1. The user selects the "List Jobs" option
2. The system displays a window that lets the user filter job listings by:
  1. Short description
  2. Title
  3. Created (date range)
  4. Modified (date range)
  5. Created By
  6. Module
  7. Type
  8. Issue
3. The user enters the desired search criteria, then requests to search
4. The system lists the found jobs, sorted by the following fields:
  1. Short description
  2. Long description
  3. Created by
    1. If the user account has been deleted from the system, display Unknown User instead
  4. Title
  5. Issue
  6. Type
  7. Module
  8. Created (date)
  9. Modified (date)
5. The user can sort the data by any of the displayed columns
6. The user can use standard "VCR buttons" to move to the First, Previous, Next, Last and User-specified page number

...

(a)

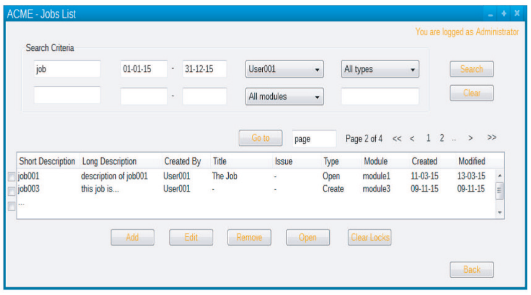
...

1. The Administrator requests to list the jobs.
2. ACME displays all Jobs* with their characterizing Job Info*.



Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
job002	-	User002	This is a title	an issue	Close	module1	23-05-15	27-05-15
job003	this job is...	User001	-	-	Create	module3	09-11-15	09-11-15
...								

3. The Administrator enters the desired Search Criteria*, then requests to search.
4. If there is at least one job satisfying the Search Criteria*, then ACME lists all jobs matching it.



Search Criteria: job; 01-01-15 - 31-12-15; User001; All types; All modules

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
job003	this job is...	User001	-	-	Create	module3	09-11-15	09-11-15
...								

...

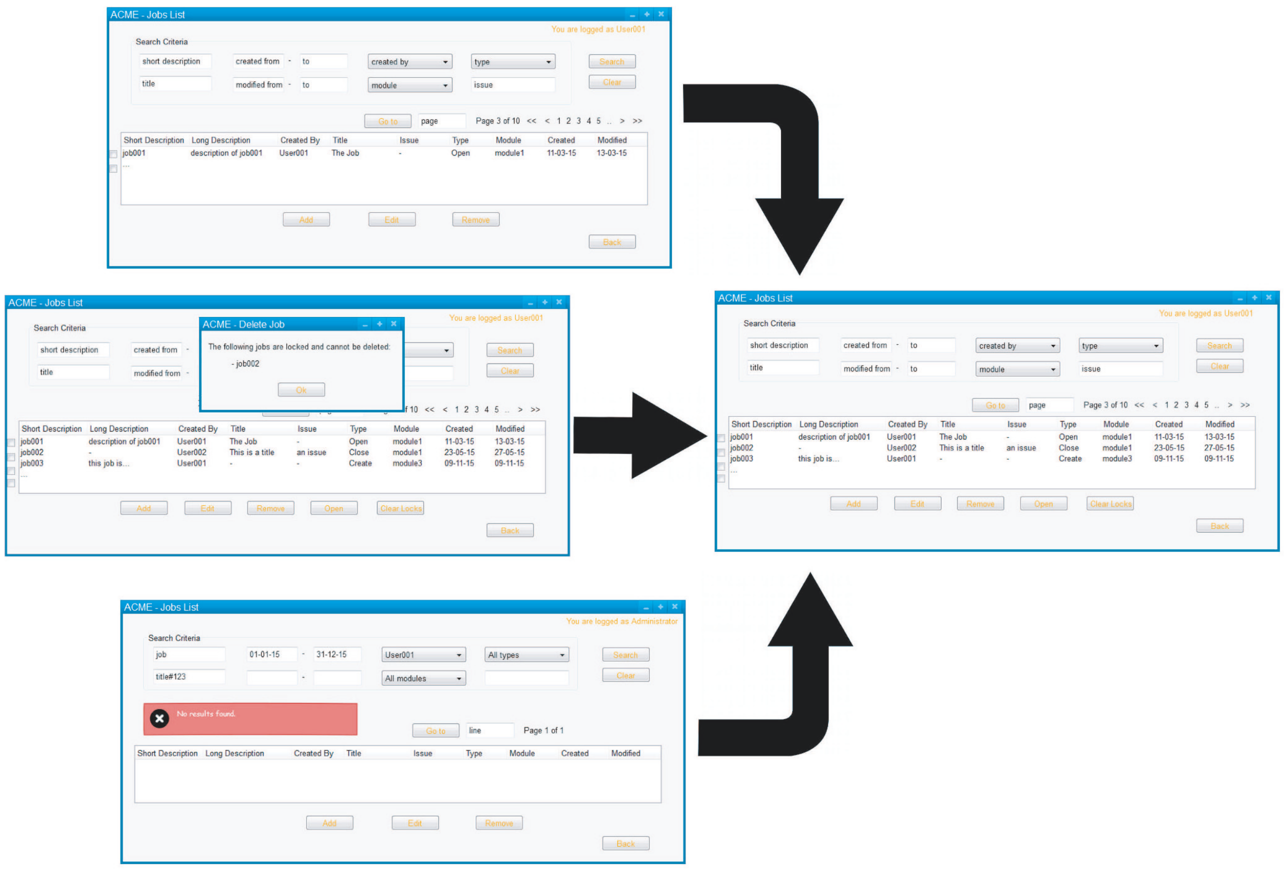
(b)

Fig.13. Comparison between (a) free[18] and (b) the corresponding disciplined[13] specifications shown on a simplified fragment of the Search/List ACME Jobs use case (for the actor Administrator in the disciplined case).

points of the original document, and then references to the glossary entries were used in any place such entries were mentioned (see Fig.8 in Subsection 3.3 for an example of glossary usage).

For what concerns use cases descriptions, their steps and scenarios were completely restructured because of their complexity. In some circumstances, new use cases emerged due to the separation of actors that were wrongly linked to a single use case. For example, Search/List ACME Jobs from original ACME specification shows both Administrator and External User as actors, thus it was split into Administrator Search/List Jobs (see Fig.8) and External User Search/List Jobs (see also complete ACME specification[13] for further examples). Moreover, the steps were shortened by using the proposed form (i.e., [if *condition*, then] *subject interaction*; [*effect*] [*continuation*].), along with the usage of the terms from the glossary. To further reduce the granularity of a step, a split into more steps was applied when needed (e.g., when a step includes more than one unique interaction).

At last, screen mockups were added in order to simplify those steps expressing a non-trivial user interaction (e.g., a step where a user has to fill a form), reducing verbosity and expressing a prototypical GUI. For drawing the screen mockups we used the Pencil tool. We started by identifying and representing the basic templates (amounting to 15) from which we could produce, once needed, some variations that differed in some details, by extending, changing or removing the visual parts of the original ones. Fig.14 shows an example of the dependency relation that exists between a template and some possible variations. In this example, the template is the first mockup that was encountered during the restructuring process. The mockups successively encountered that were similar to the first one (e.g., same functionality of ACME, but different permissions/state) were associated with it in a dependency relationship. It is interesting to notice that, in some cases, this relationship may involve a "downgrade" of



Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
...								

The following jobs are locked and cannot be deleted:

- job002

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
job002	-	User002	This is a title	an issue	Close	module1	23-05-15	27-05-15
job003	this job is...	User001	-	-	Create	module3	09-11-15	09-11-15
...								

job	01-01-15	-	31-12-15	User001	All types
title#123		-		All modules	

No results found.

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified

Short Description	Long Description	Created By	Title	Issue	Type	Module	Created	Modified
job001	description of job001	User001	The Job	-	Open	module1	11-03-15	13-03-15
job002	-	User002	This is a title	an issue	Close	module1	23-05-15	27-05-15
job003	this job is...	User001	-	-	Create	module3	09-11-15	09-11-15

(a) (b)

Fig.14. Screen mockups dependency: (b) an ACME template and (a) some of its children.

the original template (i.e., something is removed), an "upgrade" (i.e., something is added) or both.

We took note of problems in the original ACME requirements specification (i.e., ambiguities, inconsistencies, and incompleteness) any time we encountered them; and when a question for the stakeholders had to be answered to solve the problem, we played their role to produce a possible answer.

As an example, we provide a simplified fragment of the Search/List Jobs use case (Fig.13(a)) adapted from the ACME free specification[18] and the corresponding Administrator Search/List Jobs use case (Fig.13(b)) adapted from ACME disciplined specification, where the latter is obtained from the former. To make the free use case disciplined, a split of the two involved actors (Administrator and External User) was needed.

The complete refined ACME specification can be viewed in [13].

5.3 Results: Research Questions RQ1 and RQ2

During the application of DUSM to our case study, a set of assumptions have been made and a number of inconsistencies, ambiguities and incompleteness have been detected, classified and noted down for answering RQ1. Fig.15 lists all the various kinds of inconsistencies, ambiguities and incompleteness that we have found, along with the number of their occurrences. This list has been generated thoroughly inspecting the original ACME specification.

For example, there are three occurrences of *Same error signaled/handled differently in alternatives of the same use case* inconsistency entry, which means that the original specification signals or handles errors in different ways among the various alternatives of a given use case (e.g., sometimes an error message is shown and sometimes is not; moreover, the content of that message is not always the same). As another example, in the ambiguities category, *Same name for two different things* has been detected four times during our analysis (e.g., both External User and User terms were used to indicate the former actor). One of the biggest problems that we faced to make ACME disciplined is the lack of information to clarify the terminology; in fact, *Lacking information about features/terms* incompleteness error has been detected 15 times. Incompleteness, in particular, left us with open questions that we had to answer in order to make the ACME specification disciplined, since the stakeholders cited in Fig.1 were not available. For this reason, we generated a list of open questions (turned, then, into found problems) and possible answers (turned into possible solutions), presented in Table 2. For instance, *Unclear core functionalities* means that some main tasks, in particular the job-related ones, were left unspecified. We proposed our simplified solution trying to understand and define the term "job", in accordance with the information gathered through the documentation. Similarly, the *Unclear actors in use cases* entry means that we found some use cases where it was not clear about which actor could interact with ACME and which functionalities were enabled; in these cases, we let all the actors complete any kind of interaction with no restrictions.

For answering RQ2, we compared the original ACME specification with the final one. Summarizing, the number of disciplined use cases has increased to 31 (+9), since some original use cases have been split between

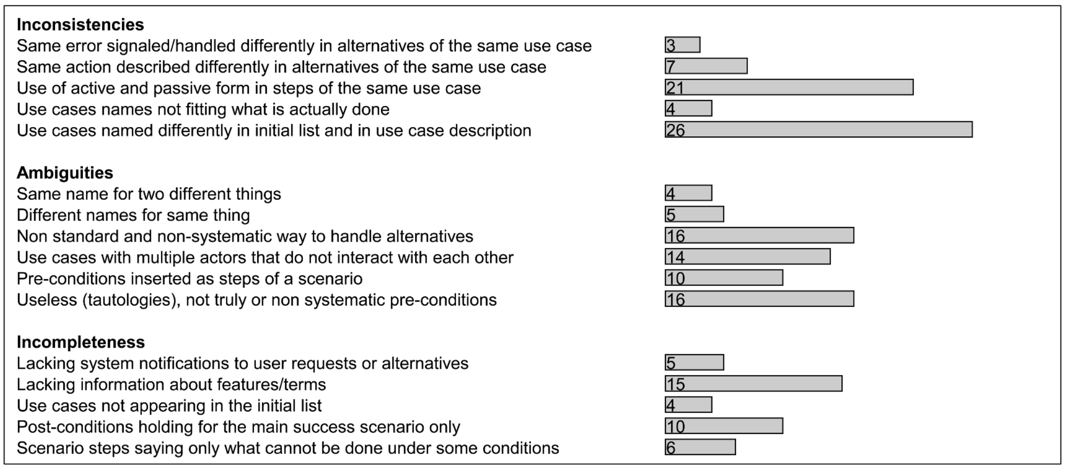


Fig.15. Inconsistencies/ambiguities/incompleteness found in ACME free specification (RQ1).

Table 2. Found Problems and Adopted Solutions to Discipline ACME Free Specification

Number	Found Problem	Possible Solution
1	Unclear actors in use cases	We decided to use both actors (Administrator and External User) sharing the same scenario, when it is not clear who is acting or what the differences among scenarios are. In these cases, User, which generalizes Administrator and External User, is the main actor
2	Unclear permissions in use cases interactions	We decided to let the actors to complete any kind of interaction if no restriction is made explicit
3	Unclear core functionalities	We decided to model the functionalities in accordance with the information gathered through the documentation, following the steps interactions and notes from the use cases description
4	Terminology abuse	We decided to adopt just one term among those which share the same semantics
5	Terminology incompleteness	We decided to remove or to simplify the interpretation of the terms that were referenced but never described
6	Lacking of alternatives in use cases scenarios	We decided to add further steps to complete all possible alternatives

the involved actors (see [13]), whereas the total amount of entries in the glossary is 28. We also produced a total of 114 screen mockups, starting from a baseline of 15 templates. Considering only the textual part of the specification, the DUSM adoption downsized the original ACME free specification by about 63%, from approximately 17 000 (exactly 16 917) to less than seven-thousand words (6 221).

6 DUSM Empirical Assessment

During the years, we have conducted several different empirical works to study the impact of introducing screen mockups (a key ingredient of DUSM) in a development process. Moreover, we studied DUSM in a real industrial context. In particular, we empirically assessed: 1) the effectiveness of screen mockups in improving the comprehension of functional requirements[19-20], and 2) the effort required to build the screen mockups[15-16]. We have also evaluated the applicability of DUSM in the industry by means of a case study. This section is left intentionally short; the reader can find more information in other work[12,15-17,19-20].

6.1 Screen Mockups Effect on the Comprehension

To assess whether screen mockups help in the comprehension of functional requirements represented with use cases, we conducted a family of four experiments with students having different profiles (e.g., undergraduate computer science students and graduate computer engineering students)[19-20]. The goal of each experiment in the family was evaluating the screen mockups effect on the comprehension of functional requirements in terms of effectiveness, efficiency and time to accomplish comprehension tasks. Effectiveness was measured in terms of comprehension level, i.e., the comprehension a participant achieved on the functional requirements of a system, while efficiency was defined as the ratio between comprehension level and task completion time. The experiments were conducted providing to the students two semantically equivalent requirements specification: use cases "as is" (control group) and use cases enriched by screen mockups (treatment group). The results showed that the use of screen mockups had a statistically significant large effect on both the functional requirements comprehension effectiveness (+69% improvement on average) and the efficiency to accomplish comprehension tasks (+89% improvement on average). The effect on the time to accomplish a comprehension task was not statistically significant. Such results were consistently observed in all of the four experiments in the family. As a consequence, we can state that one of the ingredients of DUSM is able to improve the comprehension of a requirements specification.

6.2 Cost Needed to Develop Screen Mockups

The adoption of screen mockups should also consider the cost for producing screen mockups in the requirements engineering process. In fact, it is crucial knowing whether (or not) the additional cost needed to develop screen mockups is adequately paid back by the improved comprehension of functional requirements. In that direction, we conducted a preliminary empirical investigation[15]. In the investigation we involved: Bachelor and Master students and several software professionals. We gathered information about the time to define and design screen mockups with the Pencil tool and then we collected the participants' perceptions about the effort to build screen mockups. The main finding of that study indicated that: the participants perceived screen mockups construction as a non-onerous activity that requires a low effort as compared with the effort needed to develop use cases. In a companion paper of [15], we presented an empirical study conducted to build estimation models for the effort needed to develop screen mockups starting from use cases[16]. We asked the participants to develop screen mockups from the use cases of four different software systems. Then, we built the estimation models on information about use case size measures (e.g., the number of steps or characters). Finally, we assessed the accuracy of our models. The main result of that study showed that it is possible to predict the effort to develop screen mockups starting from the textual description of the use cases. The aforementioned papers concern the production of screen mockups from scratch, on the contrary DUSM (see Subsection 4.3) produces the majority of the mockups by modifying some existing ones. Thus the above results could be considered as an upper bound of the total effort for producing the mockups. As a consequence, we can assume that the effort for adding the mockups to the use cases is not hindering the applicability of DUSM.

6.3 Industrial Case Study

DUSM has been applied with success by three of the authors during a joint project involving the University of Genova, Italy, and two local companies, having the goal of developing the EC2M system. Such system consists in an improved ECM[5] making use of ontologies to better classify, retrieve and share documentation among different branches of the companies. The functionalities offered by EC2M can be classified as interactive and non-interactive: the first ones allow the user(s) to interact with EC2M using GUIs (e.g., logging in and inserting/retrieving documents), while the others focus on the interactions between software systems using specific protocols (e.g., exchanging information or documentation using SOAP, Simple Object Access Protocol, and REST, Representational State Transfer). Since DUSM has been devised for describing the requirements specification of interactive software systems, in this project we applied the method only for the portion of requirements describing the interaction between the EC2M system and the user(s).

As described in Section 2, DUSM asks a free UC specification as input. Using the information gained in the course of the first two meetings with our industrial partners, we developed a preliminary version of the UC specification. The free UC specification reflects the requirements as informally expressed by the industrial partners (i.e., the two local companies) during the meetings. For this reasons, later, we discovered that there were several problems, for instance: 1) the meaning of the terms reported in the use cases was not always agreed by all the partners (the glossary is missing in this phase); 2) the granularity of the actions described in the use cases steps was not uniform (some of them were too abstract and some other were too detailed); 3) only a few extensions to the main success scenarios were reported.

Starting from the free UC (use cases) specification, we first developed a "disciplined UC specification" (i.e., complying with the well-formedness constraints listed in Figs.4, 5, 7, and 9) asking, when needed, the industrial partners some clarifications. In this way, we greatly improved the quality of the requirements specification, for example, by adding the glossary, levelling out the granularity of the steps (e.g., subdividing a single step in more steps), redefining some actors, and considering new scenarios. Once the "disciplined UC specification" was settled, we added the screen mockups, verifying that they were complying to the well-formedness constraints (see Fig.11). We chose to associate the mockups with the most relevant steps whose subject was EC2M. In this phase, the Pencil tool was used; it proved to have the capability to quickly create realistic screen mockups.

Finally, we organized a meeting where the "disciplined UC with screen mockups specification" was shown to the industrial partners. That occasion was very useful for identifying some misunderstandings between our understanding of the EC2M system and what the industrial partners really desired. Moreover, the screen mockups allowed to perform a sort of prototype verification helping the industrial partners to detect problems that were difficult to find by inspecting textual use cases only. After the meeting we fixed the identified problems.

The professionals involved in the creation of the functional requirements specification for EC2M were satisfied of DUSM. In particular, they found the screen mockups and the glossary very effective to improve the comprehensibility of the use cases, and useful to find the ambiguities in the requirements specification. From

[5]An enterprise content management (ECM) is a system used to capture, manage, store, and deliver enterprise content.

the analyst point of view, i.e., the role we played in this collaboration, we remark that the burden of applying DUSM was acceptable and comparable to the one we experimented in previous industrial projects, where other development processes and notations were used.

7 Related Work

Several studies having as subject use cases and screen mockups have been proposed in the literature. In the following, we present the work and tools found, classified, for the sake of simplicity, in only two categories: 1) use cases and screen mockups, and 2) constraints definition and quality evaluation of the produced requirements specification. We anticipate and highlight that no work in literature: 1) provides a systematic way to create and link screen mockups to use cases, and 2) provides a complete set of constraints over use cases, screen mockups, and their mutual relationships as DUSM does.

7.1 Use Cases and Screen Mockups

Nawrocki *et al.*[21-22] proposed the UC Workbench tool to support the activities concerning the use cases development, including the ability to use screen sketches in a web-based context. UC Workbench adopts well-known patterns[23] and uses the Formal USE Case Language (FUSE) to describe the scenarios by means of constructs (e.g., either-or). Then, users can associate externally produced low fidelity sketches to use cases steps. The tool supports additional features, such as automatic detection of bad smells in use cases and effort evaluation through use case points. The main difference with DUSM is that the tool does not enforce constraints on the connection among screen sketches and use cases steps, since there are neither hints nor rules to check the linkage. Moreover, their work focuses on web application requirements engineering only, while our method is independent from the system domain.

The majority of tools for managing requirements enriched with mockups are commercial. For instance, PowerStory[6] helps in keeping use cases based specification and wireframes synchronized. The user is able to formulate structured use cases, where actors are explicitly defined, and link low or high fidelity wireframes to use cases steps. The given wireframes can also be used to simulate the application execution (as in DUSM). Another requirements management tool is CaseComplete[7], which provides an environment to define consistent and accurate use cases based requirements specification. Use cases descriptions follow a well-structured template, where each step can be linked to a low fidelity interactive screen mockup. Similar to DUSM, the tool allows to precisely describe the used terminology in a glossary that can be referred in use cases steps. UeXceler[8] is a requirement gathering extension for Visual Paradigm that takes place in agile development. The user has the possibility to enter use cases statements (in the well-known *As a/I want/so that* template), to generate associated use cases that will include some user stories deducted from the meeting with the stakeholders. User stories can be detailed by adding interaction steps and GUI elements leading to wireframes design. Each wireframe can be attached to a specific interaction step and the whole flow of wireframes can be executed to simulate the user interaction. Those commercial tools subsume methods close to DUSM, since they adopt a sort of glossary to explain the terminology and link GUI representations to steps. However, they do not provide a systematic way to create and link screen mockups to use cases. Furthermore, they miss a complete set of constraints over use cases, screen mockups, and their mutual relationships, to guarantee a good quality of the resulting specification, as DUSM does.

There is also a category of work pointing to (manually, semi-automatically or automatically) generate screen mockups from use case specifications.

In his book, Maciaszek[24] offered some principles and guidelines to define interface images of desktop and web applications, starting from use cases specifications. The book also provides examples of user experience (UX) storyboards, where UX elements (GUI components) are identified and, consequently, UML diagrams are derived, in order to represent the use cases scenarios through UX elements interactions (using sequence diagrams) and the content of the attached screen windows (using a class diagram). Different from DUSM, the book does not explicit any precise constraint that a GUI has to follow according to related use cases steps. Olek *et al.*[25] proposed a screen specifications tool, ScreenSpec, which is based on a language to textually describe screen specifications during web applications re-

[6] http://power-story.com/, July 2018.

[7] http://casecomplete.com, July 2018.

[8] https://www.visual-paradigm.com/training/agile-development-with-uexceler/, July 2018.

quirements elicitation. The textual representation of a screen is basically a screen identifier followed by a list of web elements names and types (e.g., a button or a link) adherent to a grammar. The textual description is then translated into interactive HTML low fidelity visual representations and connected to use cases steps through the external aid of UC Workbench tool[21-22]. Here, the main difference with DUSM concerns the fact that screen mockups are described by means of an adhoc language and then generated, while in DUSM screen mockups are produced using a GUI prototyping tool. In addition, the part related to use case steps linkage and constraints checking is lacking.

Jiménez *et al.*[26] presented a model-driven development approach to extend UML use case, activity and class diagrams for describing and implementing generic system interfaces. Similarly to DUSM, the basis for system interface definition are the use cases, from which the system's main GUI windows are derived. The users' interactions are extracted from use cases and represented through activity diagrams, which are then connected to the relative use cases. Actions and transitions are marked with stereotypes that represent GUI components and interactions (e.g., clicking on a button). From these extended diagrams, a GUI-class diagram is modelled and used to generate interfaces in Java code (i.e., a use case is a Java class). The main difference with respect to DUSM is that this approach translates a whole use case as a unique GUI window, thereby the connection between system interfaces and use cases is not at step level.

7.2 Constraints Definition and Quality Evaluation of the Produced Requirements Specification

Ciemniewska *et al.*[27] proposed a mechanism for supporting use cases reviews, based on natural language processing techniques. Its aim is finding automatically easy-to-detect defects, including, for instance, use cases duplications, inconsistent style in use cases naming, and too complex sentence structure. In this way, reviewers can focus on detecting more complex defects. This approach is very different from DUSM, since the authors tried to detect simple bad smells by automatically analyzing the textual content of the use cases, while we propose a way to present disciplined use case satisfying quite complex well-formedness constraints.

Kamalrudin and Grundy[28] presented an extension to their extraction and modelling tool, MaramaAI, which captures requirements as essential use cases (semi-formal models validated against some patterns) from a natural language specification and maps them into abstract (and later concrete) form-based interfaces. Similar to DUSM, one of the goals is the reduction of misleading information that may originate from an unstructured natural language. However, the goal is completely different: generating simple mockups starting from textual information versus proposing a method for creating (or restructuring existing) high-quality use cases linked with screen mockups.

In the context of agile methodologies, requirements are usually expressed in sketched and informal formats, resulting in possible issues concerning, among all, ambiguities, incompleteness and inconsistencies. Starting from both the traditional and agile methodologies quality frameworks and from the criteria found in literature, Heck and Zaidman[29] proposed a quality framework for agile requirements, instantiated on features requests in open source projects and on user stories. Different from their proposal, our well-formedness constraints work at a lower level perspective (i.e., interactions described in use cases) and involve all the produced artifacts composing a requirements specification and how they are made mutually consistent. Our method is also based on iterations between the analysts and the stakeholders, in order to produce a requirements specification complete, consistent, and free from any ambiguity, thanks to the introduction of screen mockups to enrich the textual descriptions.

8 Conclusions

In this paper we proposed DUSM, a method for specifying the requirements of a generic software system. The main novelty is given by the precise linkage between screen mockups and use cases and by the large amount of well-formedness constraints to detect problems among the produced artifacts. The disciplined use cases are in fact consistently enriched by screen mockups fully integrated in the development process. In this way, requirements specifications produced with DUSM are:

• easier to comprehend (see RQ2 of the ACME case study, Section 5), thanks to the screen mockups[19-20],

• less prone to inconsistencies, ambiguities, and incompleteness (see RQ1 of the ACME case study, Section 5), thanks to the glossary and the large amount of well-formedness constraints,

• in general of "good quality" (see RQ1 of the ACME case study, Section 5), since checking the many well-formedness constraints associated with our template results in a deep and strongly structured inspection, and

• not burdensome in its application (indeed the screen mockup generation, one of the most onerous activities, has been considered not too heavy[15]).

Moreover, DUSM has been successfully applied in an industrial project and used for many years in software engineering students' projects at the University of Genova, which improves the confidence in favour of its applicability.

Differently from other work based on formal languages or modelling notations, our specification may be read, understood, and produced also by non-experts, with the extra (but not burdensome) effort to make use cases disciplined.

As future work, we intend to implement a tool guiding the user in the application of DUSM to produce requirements specifications based on disciplined use cases and screen mockups. The tool will also automatically validate the produced artifacts with respect to the set of well-formedness constraints and bad smells we described in the paper, concerning glossary, use case diagram, use cases descriptions, and use cases scenarios, that will be implemented in the tool in a more formal way (e.g., OCL) in order to make the automated validation possible. The validation of screen mockups with respect to the associated well-formedness constraints and bad smells will be made automated after having abstractly represented screen mockups with a proper metamodel. We also intend to apply our method on further business domains and case studies, involving both students and industrial partners to gather any useful feedback to be used for future improvements. Finally, we will investigate existing user stories quality assurance proposals in order to understand whether some principles could be also adopted to improve our method.

References

[1] Cockburn A. Writing Effective Use Cases (1st edition). Addison-Wesley Professional, 2000.

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

[3] O'Docherty M. Object-Oriented Analysis and Design: Understanding System Development with UML 2 (1st edition). Wiley, 2005.

[4] Ferreira J, Noble J, Biddle R. Agile development iterations and UI design. In *Proc. Agile 2017*, Aug. 2007, pp.50-58.

[5] Astesiano E, Reggio G. Knowledge structuring and representation in requirement specification. In *Proc. the 14th SEKE*, Jul. 2002, pp.143-150.

[6] Choppy C, Reggio G. Improving use case based requirements using formally grounded specifications. In *Proc. the 7th International Conference on Fundamental Approaches to Software Engineering*, Mar. 2004, pp.244-260.

[7] Reggio G, Leotta M, Ricca F *et al.* Business process modelling: Five styles and a method to choose the most suitable one. In *Proc. the 2nd Int. Workshop. Experiences and Empirical Studies in Software Modelling*, Oct. 2012, Article No. 8.

[8] Leotta M, Reggio G, Ricca F, Astesiano E. Towards a lightweight model driven method for developing SOA systems using existing assets. In *Proc. the 14th Int. Symp. Web Systems Evolution*, Sept. 2012, pp.51-60.

[9] Rivero J M, Grigera J, Rossi G, Luna E R, Montero F, Gaedke M. Mockup-driven development: Providing agile support for model-driven web engineering. *Information and Software Technology*, 2014, 56(6): 670-687.

[10] Zhang J, Chang C, Chung J Y. Mockup-driven fast-prototyping methodology for web requirements engineering. In *Proc. the 27th Annual International Computer Software and Applications Conference*, Nov. 2003, pp.263-268.

[11] Reggio G, Ricca F, Leotta M. Improving the quality and the comprehension of requirements: Disciplined use cases and mockups. In *Proc. the 40th Euromicro Conf. Software Engineering and Advanced Applications*, Aug. 2014, pp.262-266.

[12] Reggio G, Leotta M, Ricca F. A method for requirements capture and specification based on disciplined use cases and screen mockups. In *Proc. the 16th Int. Conf. Product-Focused Software Process Improvement*, Dec. 2015, pp.105-113.

[13] Reggio G, Clerissi D. ACME: Complete requirements specification: A case study. http://sepl.dibris.unige.it/TR/ACME-Complete.pdf, Aug. 2018.

[14] Landay J A, Myers B A. Interactive sketching for the early stages of user interface design. In *Proc. the SIGCHI Conference on Human Factors in Computing Systems*, May 1995, pp.43-50.

[15] Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E. On the effort of augmenting use cases with screen mockups: Results from a preliminary empirical study. In *Proc. the 2010 ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement*, Sept. 2010, Article No. 40.

[16] Scanniello G, Ricca F, Torchiano M, Gravino C, Reggio G. Estimating the effort to develop screen mockups. In *Proc. the 39th Euromicro Conference on Software Engineering and Advanced Applications*, Sept. 2013, pp.341-348.

[17] Astesiano E, Cerioli M, Reggio G, Ricca F. A phased highly-interactive approach to teaching UML-based software development. In *Proc. the 3rd Educators Symposium of the 10th ACM/IEEE Int. Conf. Model Driven Engineering Languages and Systems*, Sept. 2007, pp.9-18.

[18] Alexander A. ACME phase 2(c) requirements specification enabling external user functionality (version 6. 1). http://alvinalexander.com/java/misc/ReEnableExternalUser/, July 2018.

[19] Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E. On the effectiveness of screen mockups in requirements engineering: Results from an internal replication. In *Proc. the 2010 ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement*, Sept. 2010, Article No. 17.

[20] Ricca F, Scanniello G, Torchiano M *et al.* Assessing the effect of screen mockups on the comprehension of functional requirements. *ACM Trans. Software Engineering and Methodology*, 2014, 24(1): Article No. 1.

[21] Nawrocki J, Olek Ł. Use-cases engineering with UC Workbench. In *Software Engineering: Evolution and Emerging Technologies*, Zielinski K, Szmuc T (eds.), IOS Press, 2005, pp.319-329.

[22] Nawrocki J R, Olek Ł. UC Workbench — A tool for writing use cases and generating mockups. In *Proc. the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*, June 2005, pp.230-234.

[23] Adolph S, Bramble P, Cockburn A, Pols A. Patterns for Effective Use Cases (The Agile Software Development Series) (1st edition). Addison-Wesley Professional, 2002.

[24] Maciaszek L A. Requirements Analysis and System Design (3rd edition). Pearson Education Canada, 2007.

[25] Olek Ł, Ochodek M, Nawrocki J. Enhancing use cases with screen designs. A comparison of two approaches. *Computing and Informatics*, 2010, 29: 3-25.

[26] Almendros-Jimenez J M, Iribarne L. Designing GUI components for UML use cases. In *Proc. the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Apr. 2005, pp.210-217.

[27] Ciemniewska A, Jurkiewicz J, Olek Ł, Nawrocki J. Supporting use-case reviews. In *Proc. the 10th Int. Conf. Business Information Systems*, Apr. 2007, pp.424-437.

[28] Kamalrudin M, Grundy J. Generating essential user interface prototypes to validate requirements. In *Proc. the 26th ASE*, Nov. 2011, pp. 564-567.

[29] Heck P, Zaidman A. A quality framework for agile requirements: A practitioner's perspective. arXiv: 1406. 4692, 2014. http://arxiv.org/abs/1406. 4692, July 2018.

Gianna Reggio has been an associate professor at University of Genova, Genova, since 1992. Previously she was an assistant professor at the same university. She received her Ph.D. degree in informatics in 1988. She took part in several research projects, and co-organized several international conferences and workshops, the most relevant being ETAPS 2001 and MODELS 2006. She has been a member of the program committee of several international conferences and in particular MODELS. Her research activity started in the field of formal methods (algebraic specifications, specification languages for concurrent systems and semantics of programming languages). Later her interests moved to the field of software engineering, proposing modelling methods based on UML for requirement and design specifications validated empirically, and recently she has started working on development methods for service-based and IoT systems. She is an author of several national and international conferences and journals papers and books.

Maurizio Leotta is a postdoctoral researcher at the University of Genova, Genova. He received his Ph.D. degree in computer science from the same university, in 2015, with the thesis "Automated Web Testing: Analysis and Maintenance Effort Reduction". He is the author or co-author of more than 50 research papers published in international journals and conferences. His current research interests are in software engineering, with a particular focus on the following themes: web/mobile/IoT application testing, functional testing automation, business process modelling, empirical software engineering, and model-driven software engineering.

Filippo Ricca is an associate professor at the University of Genova, Genova. He received his Ph.D. degree in computer science from the same university, in 2003, with the thesis "Analysis, Testing and Re-structuring of Web Applications". In 2011 he was awarded the ICSE 2001 MIP (Most Influential Paper) Award, for his paper: "Analysis and Testing of Web Applications". He is an author or co-author of more than 100 research papers published in international journals and conferences/workshops. Filippo Ricca was the Program Chair of CSMR/WCRE 2014, CSMR 2013, ICPC 2011 and WSE 2008. Among the others, he served in the program committees of the following conferences: ICSM, ICST, SCAM, CSMR, WCRE and ESEM. From 1999 to 2006, he worked with the Software Engineering group at ITC-irst (now FBK-irst), Trento, Italy. During that time he was part of the team that worked on reverse engineering, re-engineering and software testing. His current research interests include: software modeling, reverse engineering, empirical studies in software engineering, Web applications and software testing. The research is mainly conducted through empirical methods such as case studies, controlled experiments, and surveys.

Diego Clerissi is a Ph.D. student in computer science at the University of Genova, Genova. In 2015, he received his Master degree from the Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS) of the University of Genova, with the thesis "Test Cases Generation for Web Applications from Requirements Specification: Preliminary Results". His research interests include systems modeling, requirements-based testing, web applications and IoT systems testing.