Requirements Eng (2012) 17:187–201
DOI 10.1007/s00766-011-0135-y

ORIGINAL ARTICLE

Constructing high quality use case models: a systematic review of current practices

Mohamed El-Attar · James Miller

Received: 23 September 2009 / Accepted: 10 September 2011 / Published online: 30 September 2011

Abstract There is an increasing recognition for the need to develop high quality use case models from the professional and academic communities. Quality in use case models is of particular importance when they are utilized within a use case driven development process, whereby every aspect of development is driven by the models and influenced by their quality. Many practitioners and researchers have provided guidelines, suggestions and techniques to construct high quality use case models. This invaluable body of knowledge is disseminated across numerous literature resources. Without unifying this knowledge into one resource, it cannot be expected that a use case modeler would be fully aware of the entire body of knowledge and benefitting from it. This paper presents a systematic review that was conducted in order to identify and amalgamate this knowledge. The amalgamated knowledge is presented in a unified form, specifically as a set of 26 anti-patterns, which modelers can use to improve the quality of their models.

Keywords Use cases · Anti-patterns · UML · Systematic review

M. El-Attar (✉)
Department of Information and Computer Science,
King Fahd University of Petroleum and Minerals,
Al-Dhahran, Saudi Arabia
e-mail: melattar@kfupm.edu.sa

J. Miller
Department of Electrical and Computer Engineering,
University of Alberta, Edmonton, AB, Canada
e-mail: jm@ece.ualberta.ca

1 Introduction

UC modeling [34] has been gaining wide acceptance by analysts, designers, testers. UC modeling can be used to drive the design and testing phases [28] and can be utilized for managerial purposes such as effort estimation [4] and business modeling [27]. The success experienced by UC modeling is chiefly because it is very simple to use. Another attractive aspect of UC modeling is that it contains a small diagrammatic notational subset and a large degree of natural language. This allows all of the stakeholders within a project to understand the UC model—even those who are not technically equipped.

The syntactical rules for creating UC models are relatively simple; however, they can be misapplied, leading to the construction of low quality UC models [8]. Practitioners and researchers have repeatedly warned against developing low quality UC models reporting on their potential costly consequences. A comprehensive list of such consequences can be found in Anda et al. [3], and El-Attar and Miller [18, 19, 22]. Researchers and practitioners hence have published many contributions, dispersed across numerous literature resources, to assist with the development of high quality UC models. To combat the issue of developing low quality UC models, a systematic review of the literature was conducted in order to unify all information relative to improving quality in the form of 21 anti-patterns as recommended in El-Attar and Miller [18, 19, 22].

The remainder of this paper is structured as follows; Sect. 2 presents the systematic review process. Analysis of the collected data from executing the search process is presented in Sect. 3. Section 4 presents the amalgamated set of anti-patterns produced by the systematic review process. Section 5 discusses the threats to validity. Finally, Sect. 6 concludes and suggests future work.

2 A systematic review process for the development of UC modeling anti-patterns

As a prelude to obtaining information relative to improving the quality of UC modeling, it is required to identify the quality attributes that should exist in UC models. The literature has identified many quality attributes of UC models. The quality attributes of UC models can be divided into five major categories as shown in Table 1.

We followed Kitchenham's guidelines for systematic reviews as reported in Kitchenham [30]. A systematic review process consists of three main steps:

- The planning stage: research objectives are identified and a review protocol is created. The purpose of the review protocol is to specify the research questions or a research objective that needs to be satisfied, and the method by which the review process will be executed. The review protocol was developed by following the guidelines, procedures and policies of the University of York's Centre for Reviews and Dissemination's guidance for those carrying out or commissioning reviews [29].
- The execution phase: a broad spectrum of literature is selected, which is then subjected to preset inclusion and exclusion criterion. Before executing the review process, it is required to approve the review process protocol in order to determine its feasibility. Approval of the review process protocol can be obtained by conducting a trial execution of the protocol.
- The results analysis phase: the literature selected for consideration is analyzed and the information of interest is gathered and amalgamated to answer the original research questions and objectives.

The review process for this work is to use the following electronic databases: ACM Digital Library; Compendex; IEEE Xplore; ISI Web of Science; Kluwer Online; INSPEC; ScienceDirect—Elsevier; SpringerLink; Google Scholar; Wiley Inter Science Journal Finder. In addition, the official OMG UML specification [34] was reviewed. The following subsections will describe the review process protocol in more detail.

2.1 Data classification scheme and scope

The result of the literature review is to gather information regarding certain aspects of UC modeling, which are categorized according to the following data classification scheme:

- Information that explains UC modeling, its notation, syntactical rules and semantics.
- Information regarding how to properly apply UC modeling, such as best practices, recommendations, quality attributes, patterns and blueprints.
- Information regarding what not to do in UC modeling, such as mistakes, pitfalls, drawbacks and poor quality attributes.

To gather these types of information, a scope for the review process was set. The scope of the review only considered literature available in the form of books, scientific journals, conference and workshop proceedings, as well as the OMG UML specification [34].

2.2 Search strategy

The search for books was conducted using the Amazon database (http://www.Amazon.com). The search for scientific journals, conference and workshop proceedings was performed using the previously mentioned electronic databases. The OMG UML specification is available online at [34].

The accuracy of fit of the search results returned by search engines is dependent on the search terms used to execute the search. The online search process was conducted as follows:

Table 1 Quality attributes of a UC model

Quality attribute	Definition
Consistency	The UC descriptions must conform to the semantics of the UC diagrams and vice versa. Information and facts must be consistent across UC descriptions. If a UC model contains more than one UC diagram, diagrammatic elements across UC diagrams must also be consistent.
Correctness and completeness	The UC diagram and textual descriptions must correctly represent the underlying requirements. In addition, the entire set of requirements that are expected to be in the UC descriptions and diagram must be present.
Fault-free	The UC diagram and descriptions must not contain any incorrect information or facts.
Analytical	The model should be analytical. An analytical model is one that only describes what the system should do, including the exclusion of any design or implementation decisions, including interface details.
Understandability	The information and facts contained in the UC descriptions must be unambiguous and precise. The model in general must be presented in a readable form. The model should not contain repeated information. In general, all stakeholders must be able to share a common understanding of the underlying functional requirements.

1. Derive the most relevant terms from the research objective.
2. Derive the most relevant terms from literature already reviewed prior to this research work.
3. Identify any alternative synonyms and spellings for the set of terms derived.
4. Derive as many search term combinations as possible.

Based on this strategy, a large number of search terms were explored, specifically*: Use Case; Use Cases; Use Case AND* (*Actor OR Model OR Description OR Text OR Textual OR Diagram OR Authoring OR Practices OR Quality OR Inspection OR Heuristics OR Guidelines OR Anti-patterns OR OR Modeling OR Mistake OR Problem OR Drawback OR Recommendation OR Suggestion OR Warning OR Rule OR Syntax OR Metamodel OR Appropriate OR Pitfalls OR Proper OR Patterns OR Blueprints OR UML OR OMG*).

The above search terms for articles relating to UC modeling were combined into a single meta-search term using the disjunction operator. As a result, an article had to include any of the terms to be retrieved. The search excluded prefaces, interviews, news, reviews, poster sessions, panels, tutorial summaries, article summaries, comments and reader's letter.

The execution of the search strategy resulted in 12,416 "hits" of which 1,636 were unduplicated citations—journals (278), conferences (962), workshops (231), books (165) and official syntax reference (1). Table 2 provides details about the distribution of "hits" per electronic database used:

2.3 Inclusion and exclusion criterion

A preset inclusion and exclusion criterion was used to filter the results returned by the search matches. The purpose of the inclusion criteria is to ensure that only literature that discusses UC modeling itself is included in the analysis. The purpose of the exclusion criteria is to avoid literature that only mentions UC modeling in a contextual manner. Articles were eligible for inclusion if:

- They discuss UC modeling, including its application, best practices, syntax, pitfalls, recommendations, quality improvement, notation, syntactical rules or semantics.
- Written in English.
- Published before 2010 (date of article submission)

Articles were excluded if they can be considered to only mention UC modeling terms or only provide a UC model without discussing the practical aspects of UC modeling or its notation**.**

For each book, the inclusion and exclusion criterion were applied by reading the title, preface and its short description if available. For books that satisfy the inclusion criteria while not satisfying the exclusion criteria, its table of contents was examined. Upon determining the relative chapters in the book, the inclusion and exclusion criteria are applied once again to exclude irrelevant chapters. For each journal, conference and workshop proceedings, the inclusion and exclusion criteria were applied by reading the title and abstract. For the OMG UML specification, the inclusion and exclusion criteria were applied by reviewing the table of contents and determining the chapters relevant to UC models. The search strategy was successfully piloted by consulting use case specialists and given the authors' prior in-depth knowledge of the domain, before executing the full search process. Citations were excluded based on their titles, prefaces, summaries and abstracts. Articles were excluded if their focus was not on UC modeling. Both authors manually went through the titles, prefaces, summaries and abstract to exclude citations that were clearly not about UC modeling. For a minority of articles, the abstracts were poorly written that they gave little indication as to what the article was about, or at least, if it is about UC modeling as a practice and not just an artifact. Only articles that were excluded by both authors (1,544 articles) would

Table 2 Hits distribution per electronic database

Database	Hits			
	Journals	Conferences	Workshops	Books
ACM Digital Library	221	827	191	142
Compendex	233	796	183	131
IEEE Xplore	248	868	204	133
ISI Web of Science	216	764	114	113
Kluwer Online	208	791	126	146
INSPEC	258	753	145	141
ScienceDirect—Elsevier	240	830	163	135
SpringerLink	236	822	182	126
Google Scholar	265	928	212	148
Wiley Inter Science Journal Finder	159	0	0	18

not proceed to the next stage, while articles excluded by one author only (26 articles) would proceed to the next stage. This strategy was applied since the ratio of such articles is much smaller than that excluded by both authors. At this stage, 92 articles remained for a full review and a detailed quality assessment.

3 Results analysis

We identified 92 articles from a variety for different information sources—journals (14), conferences (41), workshops (16), books (20) and official syntax reference (1)—that address our research question.

3.1 Quality assessment

The articles retrieved are assessed according to 6 quality criteria. Taken together, these six criteria provided a measure of the extent to which we could be confident that a particular article reported could make a valuable contribution to the field. Each of the six criteria was done on a dichotomous scale. Both authors independently reviewed the articles.

The quality of each of the articles was assessed using the following screening criterion:

c1. Would the application of a recommended practice lead to a syntactically correct UC model?
c2. Is the paper based upon research?
c3. Is the reported practice clearly identified and presented?
c4. Is the reported recommended practice validated in practice or merely a "lessons learned" report based on expert opinion?
c5. Has the recommended practice been empirically validated?
c6. Is the reported practice of value to future research?

The first two screening criterion represent the minimum quality threshold and is used to exclude articles and chapters. The remaining screening criterion is used to judge the validity and usefulness a recommended practice stated in an article or chapter. Due to space limitations, Table 3 provides a sample of the detailed quality assessment of a cross-section of references. Seven articles were rated "NO" on the first screening criteria and were subsequently excluded. A different article failed the second screening criterion and which was subsequently excluded. The majority of guidelines in books (17 out of 20) were not based on research. All guidelines presented in books, however, were validated in an industrial setting. More than half of the conference and workshop articles (27 out of 57) did not validate their findings using an industrial case study but rather mock examples. Six conference papers validated their findings empirically. Overall, none of the articles received a full score (of six) on the quality assessment—15 other articles received three to four positive answers.

A total of 61 unique "rules" or "heuristics" were compiled from our search. We developed a Table that gives an overview of the rules and heuristics retrieved according to their source. In general, we found that most guidelines in books were not based on research and were not empirically validated. Moreover, the majority of guidelines presented in books were repeated. Guidelines presented in books and workshops in general were more trivial than those presented in conference and journal proceedings. The heuristics and rules presented in conference and journal proceedings contained very little overlap or contradiction. Overall, guidelines presented in all articles were well described but their validation and reliability were not clearly addressed. Statistical analyses performed in empirical evaluations presented in conference and journal proceeding were not well presented. Empirical evaluations also did not clearly address the issues of bias in their threats of validity.

Table 3 Sample of the detailed quality assessment

Reference	Criteria					
	c1	c2	c3	c4	c5	c6
Bittner and Spence [8]	✔	×	✔	✔	✔	✔
Anda and Sjøberg [2]	✔	✔	✔	×	✔	✔
Cockburn [12]	✔	×	✔	✔	×	✔
El-Attar and Miller [22]	✔	✔	✔	×	×	✔
El-Attar and Miller [20]	✔	✔	✔	×	×	✔
Lilly [32]	✔	✔	✔	✔	×	✔
Overgaard and Palmkvist [35]	✔	×	✔	✔	✔	✔
Ren et al. [36]	✔	✔	✔	×	×	✔
Berenbach [7]	✔	×	✔	✔	✔	✔
Anda et al. [3]	✔	✔	✔	×	×	✔

Based on the assessment of the 92 articles, eight further articles were excluded. The information gathered from the remaining 84 articles was used to develop the set of anti-patterns presented in Sect. 4.3. Due to space limitations, a cross section of the developed anti-patterns is presented in Sect. 4.3. The heuristics and rules described in each anti-pattern are augmented with references to the literature sources from which it originated. Such information included in each anti-pattern can allow for data reduction techniques to be performed. Data reduction techniques can be used to determine which anti-patterns would have been identified if only journal and conference proceedings were considered, or if only books were considered…etc.

3.2 Analysis of the literature

After eliciting the desired information from the literature source, some interesting aspects of the information presented in the literature can be drawn. Most of the guidelines presented in the literature are practice-oriented, which were validated through case studies where their authors were involved. Such guidelines are available in books written by use case modeling practitioners rather than researchers or academic staff. Many books retrieved from this search were about better modeling using UML and not specific to use case models. The type of guidelines presented in such books is simple and at times trivial. This can be attributed to books usually presenting the basic "ground truth" about use case modeling, mostly concerned with the modeling notation and semantics based directly from the OMG standard [34]. Due to the basic nature of the guidelines provided in books and because they are presented in accordance to the OMG standard [34], such guidelines are therefore correct and valid. Therefore, the majority of information presented about use cases in such books contained a large amount of commonality. Guidelines provided in books are usually useful to avoid modeling mistakes rather than anti-patterns. We retrieved only six books that mainly discussed use case modeling. Such books provided more detailed (and usually more sophisticated) guidelines than those presented in general UML books, yet they contained a large degree of commonality. The guidelines presented in these six books do not contradict each other nor do they contradict the OMG standard [34]. However, the *advanced* guidelines presented in these six books contain very little overlap. For example, in Bittner and Spence [8], the guidelines heavily focused on advanced concepts of the modeling notation, while; in Overgaard and Palmkvist [35], the authors leverage the concept of design patterns and blueprints to develop better models. Finally, in Cockburn [12], the author presents guidelines to describe use cases and provides a number of templates that are nowadays informally considered as industry standard. Their respective authors validate the guidelines in these advanced books by applying them, with success, in their vast professional experiences and their evident utilization in industry.

Guidelines presented in conference and journal papers usually build upon the guidelines already presented and validated in books. For example, authors of Anda and Sjøberg [2] provide an inspection technique for use case models that is based on guidelines presented in Armour and Miller [5], Kulak and Guiney [31] and Schneider and Winters [38]. Conference and journal papers have very little in common with respect to their contribution toward use case modeling as a technique. A large number of conference and journal papers researched contained guidelines to improve use case models with a goal to improve other aspects of the software development process, such as developing a higher quality design or creating more effective unit tests. Only conference and journal papers proposed suggestions to extend the UML use case modeling notation to improve quality in use case models. In general, very little contradiction was found between literature resources. This is attributable to the fact that conference and journal papers provide different approaches to improve quality in use case models that are not exclusive. A use case modeler hence can use a multitude of approaches and guidelines presented in conference and journal papers to improve their use case models. For example, with respect to the use case textual descriptions, a set of use case descriptions can be inspected [2], as well use a template [12], and adhere to styling and content guidelines [3]. Moreover, guidelines presented in conference and journal papers at times build upon guidelines presented in other conference and journal papers.

The guidelines can be generally categorized as to improve one of the following aspects:

1. The clarity and design of the UC diagram;
2. The correct utilization of the UC diagrammatic notation to represent the functional requirements;
3. The contents of each UC description;
4. The style of documenting UC descriptions;
5. Social, political and psychological aspects of UC model construction; and
6. The impact of UC modeling on subsequent activities.

Examples of guidelines for each of the abovementioned categories are outlined in Table 4.

3.3 Anti-patterns: a brief introduction

Prior to presenting the set of anti-patterns in Sect. 4, a brief introduction to anti-patterns is provided. Learning from previous experiences and mistakes is the main concept behind using anti-patterns. An anti-pattern is "*a literary*

Table 4 Examples of guidelines

Category	Guideline examples
1	The system boundary should be defined or consistent (Lilly [32])
	Do not have too many use cases (Bittner and Spence [8]; Lilly [32])
	Use various blueprints and patterns (Overgaard and Palmkvist [35])
	Avoid the early use of packages until some significant amount of use case modeling has been completed (Berenbach [7])
2	Make parent use cases abstract unless they are complete (Bittner and Spence [8])
	Do not use the *include* and *extend* relationships to create functionally decomposed use cases (Bittner and Spence [8])
3	Do not include nonfunctional requirements (Bittner and Spence [8]; Firesmith [26]; Jacobson et al. [27]; Overgaard and Palmkvist [35])
	Use an inspection checklist to detect defects in the use case descriptions (Anda and Sjøberg [2])
4	Use a template to describe use cases because the predefined structure of a template forces the developer to identify and include important elements in each use case (Kulak and Guiney [31], McCoy [33])
	Using the tool RUT to identify risky descriptions such as incomplete or weak phrases (McCoy [33])
5	Do not spend too much time perfecting use cases (Ambler [1]; Cockburn [12])
	Do not begin writing use cases when you are tired (Cockburn [12])
	Use case models should be developed by staff that are technically trained to perform use case modeling (Chandrasekaran [10])
6	Avoid use case driven design (Firesmith [26])
	Use an iterative incremental, parallel development cycle (Firesmith [26])
	Utilize use cases to estimate software development effort (Anda et al. [4])

form that describes a commonly occurring solution to a problem that generates decidedly negative consequences" [9]. An anti-pattern explains a repeated pattern of action, process or structure that may initially appear beneficial but ultimately may cause deficiencies in a UC model. An anti-pattern will explain why such pattern seemed appropriate in the first place. A detection mechanism in included in each anti-patterns description to guide modelers to areas in the UC model where an anti-pattern may exist, be it in the UC diagram, the descriptions, or both. Finally, an anti-pattern provides suggestions upon improving and refactoring the current structure to avoid potential harmful consequences. All information included in an anti-pattern should be from actual practice.

El-Attar and Miller [18, 19, 22] proposed the concept of utilizing anti-patterns in order to improve quality in UC models. In El-Attar and Miller [22], a template to define anti-patterns was provided, which is shown in Table 5 and is used in this paper. The template presented here is a lightweight adaptation of the "*Full Anti-Pattern Template*" presented in Brown et al. [9]. The purpose of each field is described within Table 5. For further information regarding the UC modeling anti-pattern template and how UC modeling anti-patterns can be used to improve quality in UC models, see El-Attar and Miller [18, 19, 22].

4 UC modeling anti-patterns

A summary of the anti-patterns presented in this Section are listed in Table 6. A detailed description of all anti-patterns follows Table 4. Due to space limitations, only 14 anti-patterns are shown in this paper. We encourage our interested readers to view a detailed description of the remaining anti-patterns available at [17]. A number of anti-patterns were presented in earlier work, which can be located in El-Attar and Miller [22]. Table 6 also indicates the literature sources which were used to construct each anti-pattern. Recall that an anti-pattern may exist in the UC diagram, the descriptions, or both. Therefore, to detect an

Table 5 An UC modeling anti-pattern template

Anti-pattern name: The title of the anti-pattern.

Description: A description of the faulty decisions or techniques.

Rationale: A list of the deceptive or seductive reasons as to why the fallacious solution seemed to be appropriate.

Consequences: A list of the harmful consequences that could be sustained from applying the fallacious solution.

Detection:

Where: A guide to the areas where the anti-pattern can exist.

How: Instructions that are used to positively identify a match for the anti-pattern.

Improvement: A list of actions that can be performed to convert a fallacious solution into a superior solution or avoid the fallacious solution.

Table 6 Anti-patterns presented in this section

	Anti-pattern name	Literature sources
Anti-patterns that require examination of the UC diagram		
a1.	A UC that is used as an extension and inclusion UC.	Armour and Miller [5]; Bittner and Spence [8]; Kulak and Guiney [31]; Object Management Group [34]; El-Attar and Miller [22]
a2.	Functional decomposition of UCs: using the *include* relationship.	Armour and Miller [5]; Bittner and Spence [8]; Ben Achour et al. [6]; Overgaard and Palmkvist [35]; Object Management Group [34]
a3.	Functional decomposition by using an *extension* UC to *extend* multiple UCs	Armour and Miller [5]; Bittner and Spence [8]; Ben Achour et al. [6]; Overgaard and Palmkvist [35]; Object Management Group [34]
a4.	Using *extension/inclusion* UCs to implement an *abstract* UC.	Berenbach [7]; Bittner and Spence [8]; Overgaard and Palmkvist [35]; Object Management Group [34]
a5.	Multiple generalizations of a UC.	Bittner and Spence [8]; El-Attar and Miller [22]; Object Management Group [34]
a6.	An actor associated with an unimplemented *abstract* UC.	Berenbach [7]; El-Attar and Miller [22]; Ren et al. [36]; Rui and Butler [37]; Schneider and Winters [38]; Xu et al. [39]; Object Management Group [34]
Anti-patterns that require examination of the textual descriptions		
a7.	Using instances for actors instead of roles.	Bittner and Spence [8]; Ben Achour et al. [6]; Cockburn [11, 12]; Object Management Group [34]; Fabbrini et al. [24]; Fantechi et al. [25]; Firesmith [26]
Anti-patterns that require examination of both the UC diagram and textual descriptions		
a8.	Too many UCs.	Bittner and Spence [8]; Lilly [32]; Overgaard and Palmkvist [35]; Cockburn [12]

anti-pattern match, it may be required to examine the (a) UC diagram, (b) text or (c) both.

4.1 Anti-patterns that require examination of the UC diagram

Anti-pattern name

4.1.1 a1. A UC that is used as an extension and inclusion UC

- Description

The reuse of a preexisting UC is achieved by making it both an *extension* UC and an *inclusion* UC. For example, in a car dealership system (see Fig. 1), when a new car arrives at the dealership, it is recorded into the database of the dealership using the Add New Car UC. A precondition to adding the new car to the dealership's database is that the car must not already exist at the dealership. Therefore, UC Add New Car *includes* UC Car Not Found to check for that precondition. UC Update Car's Information is responsible for updating the information related to a particular car, such as, its current mileage or where it is located (assuming several branches). In order to update a particular car's information, this car must exist in the dealership's database. An error is generated if the given car does not exist in the database. Therefore, UC Update Car's Information is *extended* by UC Car Not Found to handle this error generated.

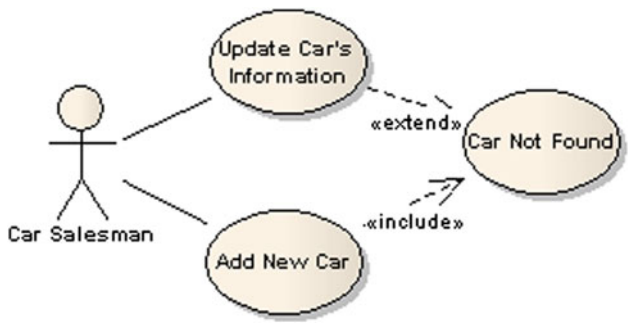

Fig. 1 UC Car Not Found was incorrectly used for the purposes of containing common functionality and exception-handling behavior

This eventually leads to the discouraged construct where a UC (Car Not Found), is an *extension* and *inclusion* UC.

- Rationale

Object-oriented modeling and design strongly promotes the concept of reuse [5, 31]. When modelers are constructing their UC models, they are keen to reuse much of the functionality contained preexisting in UCs. UC modeling offers mechanisms through its *extend, include* and *generalization* relationship to allow this reuse [34]. Reusing UCs also prevents the cluttering of the UC with many redundant UCs. However, when applying the concept of reuse, the *include* and the *extend* relationships can be misused leading to the creation of UCs containing both common and exception-handling behavior [8, 22];

- Consequences

The shared UC currently contains common and exceptional behavior required by the two *base* UCs. Therefore, when either of the *base* UCs initiate the shared UC, additional undesired functionality is performed. To further elaborate, during the operation of the *including base* UC

Add New Car, UC Car Not Found is initiated to check that the given car does not exist in the database. However, UC Car Not Found will unnecessarily also perform the procedure of trying to update a car's information that does not exist. On the other hand, if the UC Update Car's Information is performed to update the information of a given car that does not exist in the database, UC Car Not Found is initiated to handle the generated error. In this situation, the UC Car Not Found will unnecessarily check if the given car does not exist in the system.

- Detection

Where—Search for any *included* UCs in the UC diagram. **How**—If an *inclusion* UC is found, check if this *inclusion* UC is *extending* other UCs.

- Improvement

Check if the shared UC contains functionality suitable for only one of the *base* UCs. This can be achieved by examining the contents of the shared UC.

1. If the shared UC contains functionality suitable for only the *base* UC that *includes* it, the *extend* relationship should be removed. A new *extension* UC should be created to handle the exceptional situation generated by the other *base* UC. The resulting model is illustrated in Fig. 2.
2. If the shared UC contains functionality suitable only for the *base* UC that it *extends*, the *include* relationship should be removed. A new UC should be created and *included* by the other *base* UC, again resulting in the model shown in Fig. 2.

In both cases (1) and (2), the UCs should be renamed to be more indicative of their respective purposes.

3. In the case that the shared UC does indeed contain both common and exception-handling behavior. The shared UC should be split into two separate UCs. Each of the newly created UCs should only contain functionality appropriate to the *base* UC. Once again, resulting is the same model (see Fig. 2).

Anti-pattern name

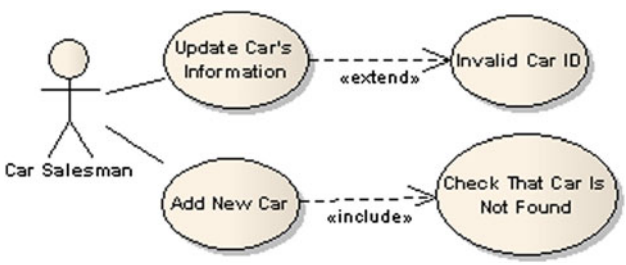

Fig. 2 The shared UC is broken into two separate UCs, each serving a different purpose

4.1.2 *a2. Functional decomposition of UCs: using the include relationship*

- Description

The *inclusion* UCs represent functions in a program, or menu options instead of a complete service that is offered the system. Such *inclusion* UCs are not used by any other UC and are not associated with any actors. For example, in the espresso machine system shown below (see Fig. 3), the *inclusion* UCs together are used to prepare a cup of coffee.

- Rationale

Dissecting analytical UCs into functions yields a set of "smaller" UCs that are naturally easier to implement. Overall, this will lead to a speedier implementation of the system. Creating "smaller" UCs is particularly attractive to modelers since they are easier to understand and code. Consequently, in later development phases, the "smaller" UCs will easier to test and maintain. Functional decomposition can be used to embody design decisions that analysts would like to enforce throughout the development of a system [6, 8, 35].

- Consequences

UCs should represent services that a system offers to its actors. Being able to abstract the actual service offered by these numerous functions by examining many decomposed UCs is a very difficult task. One can at best guess what service these UCs will offer when performed together. For complex systems, it is more likely that this "guess" will be incorrect. Moreover, functional decomposition of UCs may lead to complex descriptions of the interactions between the actors [8, 34]. Functional decomposition embodies premature design decisions which severely limits the creativity of designers and enforces them to abide to these decisions.

- Detection

Where—Look for an *inclusion* UC inside the UC diagram. **How**—Upon finding an *inclusion* UC, count the number of UCs that *include* it. If the *inclusion* UC is

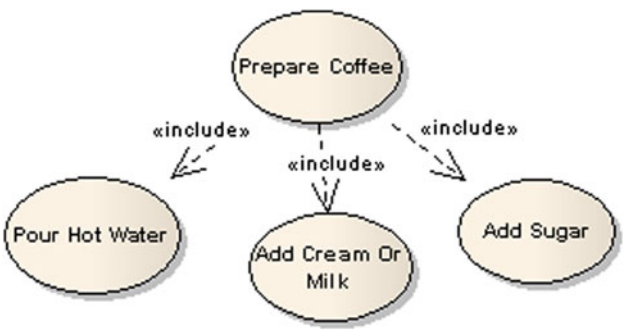

Fig. 3 Functional decomposition of the Prepare Coffee UC

included once, then the anti-pattern is matched. It is important to note that in order to match this anti-pattern; the *inclusion* UC must not be associated with any actors or UCs in the UC model.

- Improvement

The behavior described in *inclusion* UCs must be combined into UCs that individually offer a complete and meaningful service to a system's user [5].

For the espresso machine system shown in Fig. 3, the behavior described by UCs Pour Hot Water, Add Cream Or Milk and Add Sugar should be merged into the UC Prepare Coffee. It can be deduced that the user will not benefit from pouring hot water only, or having a cup with only sugar in it. The real value offered to the user is the preparation of a cup of coffee. Hence, from a conceptual point of view, a single UC called Prepare Coffee (which already exists) should be individually responsible for preparing a cup of coffee.

Anti-pattern name

4.1.3 a3. Functional decomposition by using an extension UC to extend multiple UCs

- Description

Another form of functional decomposition is the improper use of the *extend* relationship whereby an *extension* UC *extends* multiple UCs. If an *extension* UC contains general behavior that would be useful to more than one UC, this would be a strong indication that the *extension* UC has degraded into a function since an *extension* UC inserts additional behavior to a *base* UC that is very specific to the respective *base* UC. For example, in the following racquet sports store system (see Fig. 4), the UC Equipment Damaged *extends* both the Sell Racquet and Sell Ball UCs.

It is the employee's responsibility to ensure that any merchandise being sold is not damaged. Whenever the employee encounters faulty merchandise, the *extension* UC Equipment Damaged is initiated. In the case of a damaged ball, the defective ball is discarded and a new ball is handed to the customer. Meanwhile, an in-store technician can fix defective racquets; however, if the racquet is severely damaged, it is send back to the manufacturer for an exchange. Hence, it can be deduced that there are two different procedures for handling defective balls and racquets, yet the structure shown in Fig. 9 would indicate a single procedure for handling any type of damaged merchandise.

- Rationale

Similar to what is described in the "Rationale" section of the "Functional decomposition of UCs using the include relationship" anti-pattern. Moreover, there might be times where the *extension* UC is used to provide general functionality that is specialized by the UCs it *extends* [8]; Overgaard and Palmkvist [34, 35].

- Consequences

Similar to what is described in the "Consequences" section of the "Functional decomposition of UCs using the include relationship" anti-pattern. Moreover, when functional decomposition is applied using the *extend* relationship; it is often the case that the *extending* UC does not properly handle the exceptional situations caused by the *base* UCs. This situation can be easily detected in the racquet sports store system, since the procedure of handling a damaged racquet differs significantly from the procedure of the handling a damaged ball. Therefore, it can be easily deduced that more than one *extending* UC is required to handle the different exceptional situations occurring [6, 8]; Overgaard and Palmkvist [35].

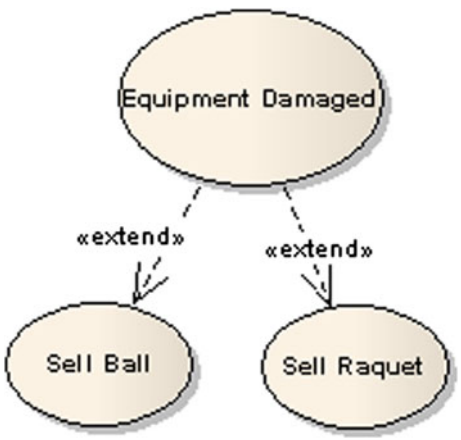

Fig. 4 Improper use of the *extend* relationship to promote functional decomposition

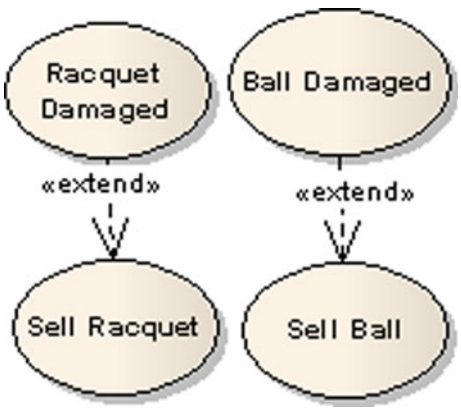

Fig. 5 *Extending* UCs disjointed to properly handle different exceptional situations

- Detection

Where—Search for *extension* UCs in the UC diagram. **How**—Upon finding an *extension* UC, count the number of UCs that the *extension* UC *extends*. The anti-pattern is matched if the *extension* UC *extends* more than one UC. It is then required that the analyst examines the behavior described by the *extension* UC to check if it is too generic. If the *extension* UC was found to contain specific behavior, it is then required by the analyst to ensure that this specific behavior is in fact suitable for all the *extended* UCs.

- Improvement

The behavior described in the *extension* UCs must be combined into UCs that individually offer a complete service to a system's user.

For the racquet sports store system illustrated in Fig. 4, the *extension* UC Equipment Damaged should be divided into two separate *extension* UCs (see Fig. 5). Each of the newly created *extension* UCs will be specifically designed to more appropriately handle the exceptional situations arising at their respective *base* UCs.

For the case when *extended base* UCs are used to specialize general behavior described by their respective *extension* UC¸ a *generalization* relationship would be more appropriate than an *extend* relationship to describe such a relationship.

Anti-pattern name

4.1.4 a4. Using extension/inclusion UCs to implement an abstract UC

- Description

An actor is directly associated with an *abstract* UC that is not implemented through a *specializing* UC. The implementation of the *abstract* UC is done through *extension* or *inclusion* UCs [7, 8]; Overgaard and Palmkvist [35].

- Rationale

The scenario described above may occur for different reasons:

1. Modelers find that the *inclusion* UCs contain subroutine behavior. On the other hand, the *extension* UCs contain exceptional or optional behavior. Therefore, the *inclusion* or *extension* UCs do not contain specialized behavior with regard to the *abstract* UC and thus should not be modeled using the *generalization* relationship. Fig. 6 illustrates an example of this scenario.
2. *Extension* or *inclusion* UCs represent specialized behavior with respect to an *abstract* UC. For example, in Fig. 7, the *abstract* UC Make a Trade can be implemented in the context of making a bonds trade, using the *inclusion* UC Make a Bonds Trade, or a stocks trade, using the UC Make a Stocks Trade.
3. The model is so far incomplete. At a later point, *specializing* UCs will be added to implement the *abstract* UC.

- Consequences

In first two scenarios described above, the *extension/inclusion* UCs are used to directly implement the *abstract* UC. However, *extension/inclusion* UCs contain behavior different from the behavior specified in the *abstract* UC. To elaborate, the behavior contained in the *extension/inclusion* UCs does not realize the behavior described in the *abstract* UC. Therefore, when the actor initiates a service request, the behavior specified by the *abstract* UC will never be performed since it is never realized by any UCs. Only *specializing* UCs may implement *abstract* UCs [34].

- Detection

Where—Search for any *abstract* UCs. **How**—(a) the *abstract* UC is associated with an actor, and (b) the

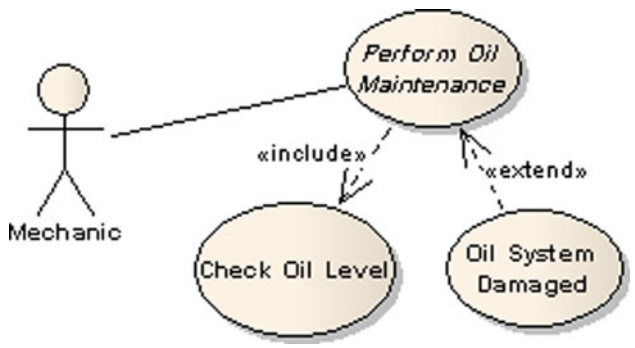

Fig. 6 An *abstract* UC *including* subroutine behavior and being *extended* by a UC containing exceptional or optional behavior

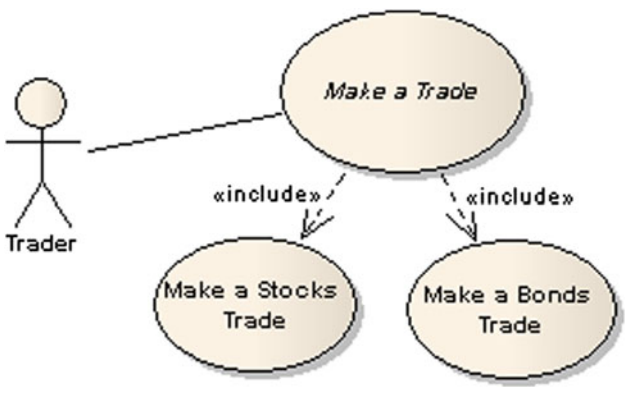

Fig. 7 An *abstract* UC *including* UCs that contain specialized behavior

abstract UC is *extended* by the other UCs or *including* other UCs, and (c) the *abstract* UC does not have child UCs.

- Improvement

1. In the scenario illustrated in Fig. 6, the type of relationships between the *abstract* UC and the other UCs (Oil System Damaged and Check Oil Level) are appropriate; and hence, should remain unchanged. Unless the model is incomplete, the *abstract* UC Perform Oil Maintenance should be set as *concrete* as shown in Fig. 8.
2. In the scenario illustrated in Fig. 7, the usage of the *include* and *extend* relationships is incorrect because the *extending/included* UCs Make a Bonds Trade and Make a Stocks Trade represent specialized behavior with respect to the *abstract* UC Make a Trade. In this case, the *specialization* relationship is considered to be the appropriate relationship between the UCs, and therefore this model can be fixed as shown in Fig. 9.

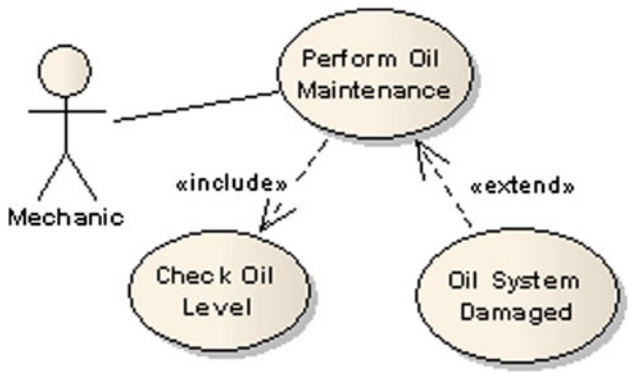

Fig. 8 The *abstract* UC now set to be *concrete*

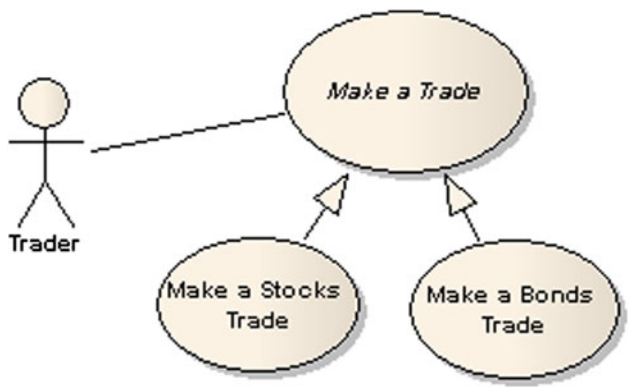

Fig. 9 The *abstract* UC is associated with its *specializing* UCs using the *generalization* relationship

3. The modelers should review and consider adding the missing *specializing* UC(s) whenever possible.

Anti-pattern name

4.1.5 a5. Multiple generalizations of a UC

- Description

Setting a single UC to *specialize* two or more UCs.

- Rationale

Modelers extract common behavior between two or more UCs and create a new *specializing* UC that will contain this common behavior. For example, preparing either a cargo or a passenger aircraft for a trip requires the cleaning of the aircraft. As shown in Fig. 10, the common behavior is contained in the UC Clean Aircraft¸ which *specializes* the UCs Prepare Passenger Aircraft For Trip and Prepare Cargo Aircraft For Trip.

- Consequences

The behavioral semantics of the model is violated. The UC Clean Aircraft is not a specialized version of the Prepare Passenger Aircraft For Trip and the Prepare Cargo Aircraft For Trip UCs, which may lead to an incorrect implementation of the system [8, 22, 34].

- Detection

Where—Search for a *specializing* UC. **How**—If that UC is *specializing* more than one UC.

- Improvement

The shared UC Clean Aircraft contains subroutine behavior required by the two other UCs. Therefore, the *specialization* relationship should be replaced with an *include* relationship. The *include* relationship is considered more appropriate since the shared UC contains common behavior not specializing behavior [22]. This solution is illustrated below in Fig. 11.

Fig. 10 Multiple *generalizations* of one UC

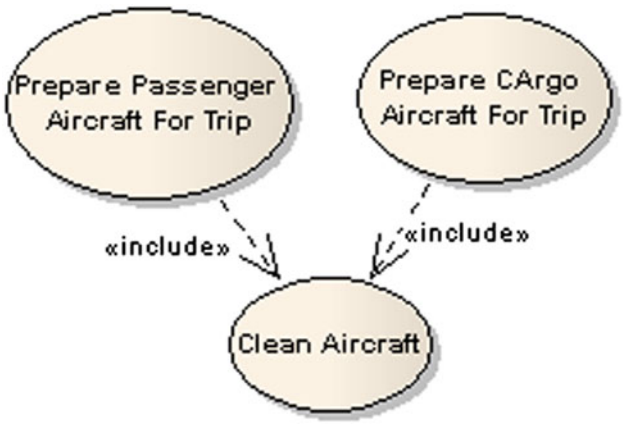

Fig. 11 The *generalized* UC should be *included* by the other UCs that need it

Anti-pattern name

4.1.6 a6. An actor associated with an unimplemented abstract UC

- Description

An actor is directly associated with an *abstract* UC that is not implemented by *specializing* UC(s) (see Fig. 12).

- Rationale

1. This situation is most likely to occur when the model is incomplete. The *abstract* UC will be implemented by *specializing* UCs in a later phase [22].
2. Modelers may incorrectly assume that an *abstract* UC may be initiated to offer a service to the initiating actor.

- Consequences

1. This is an acceptable modeling practice as long as modelers eventually insert the missing *specializing* UC(s) that will implement the *abstract* UC [7, 22, 34, 36–38, 39].
2. *Abstract* UCs cannot be initiated and hence no behavior will be performed [7, 22, 34].

- Detection

Where—Search for an *abstract* UC in the "UC Diagram". **How**—If that *abstract* UC (1) is associated with an actor, and (2) is not *specialized* by at least one UC. If the *abstracted* UC is *including* other UCs or being *extended* by other UCs then review the "Using *extension/inclusion* UCs to implement an *abstract* UC" anti-pattern.

- Improvement

1. Modelers should review and consider adding the missing *specializing* UC(s) whenever possible.
2. The *abstract* UC should be set as *concrete*. This will enable that UC to be initiated by the actor [22, 36, 37].

Fig. 12 An actor directly association with an unimplemented *abstract* UC

4.2 Anti-patterns that require examination of the textual descriptions

Anti-pattern name

4.2.1 a7. Using instances for actors instead of roles

- Description

Actors represent instances of a particular role rather than the underlying role itself.

- Rationale

1. The modelers incorrectly depict instances of the system's users instead a class of the system's users. This situation is illustrated in Fig. 13; actors Adam, Jane and Mary are all students who would like to enroll into the university.

- Consequences

The model illustrated in Fig. 13 violates the true semantics of an actor. This will yield to similar consequences as described above in (1). Moreover, the model will need to be changed frequently as instances of a type of the system's users are frequently created and removed [11, 12, 24–26, 34].

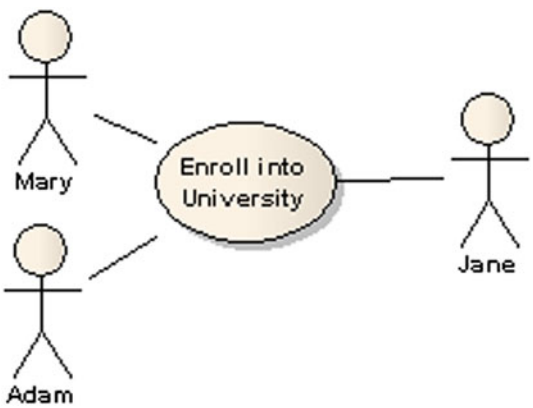

Fig. 13 A model representing instances of an actor

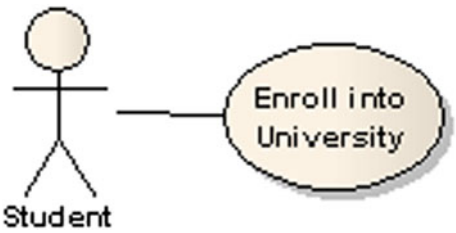

Fig. 14 The model should represent the role of a class of users not instances of the underling roles

- Detection

Where—Search for any UCs associated with actors in the UC diagram. **How**—If the UC is associated with more than one actor.

- Improvement

1. In the scenario shown in Fig. 13, actors Adam, Jane and Mary represent the role of a student. Therefore, an actor called Student should be created that will represent all instances of a student. This solution is illustrated below in Fig. 14.

4.3 Anti-patterns that require examination of both the UC diagram and textual descriptions

Anti-pattern name

4.3.1 a8. Too many UCs

- Description

The UC model contains numerous UCs. Detecting this anti-pattern is dependent on the problem domain. Therefore, identifying how many UCs is too many requires domain expertise and examination of UC models of similar systems. This knowledge will yield an appropriate range for the number of UCs expected. Therefore, this anti-pattern is matched only if the existing number of UCs far exceeds the appropriate range [8, 32]; Overgaard and Palmkvist [12, 35].

- Rationale

1. The system provides new functionalities and services which incorporates new technologies that were not available in older similar systems. Moreover, the system by nature is extremely complex, providing numerous services to many actors.
2. The UCs are designed to be simple and contain very simple behavior for easier implementation. Such UCs usually contain very short flows, and often represents GUI menu commands.

- Consequences

1. This situation is acceptable since systems evolve and become more complex, offering far more services than before.
2. This is another form of functional decomposition. The UCs offer no meaning individually and contain very little substance. The UCs are only useful when combined and sequenced with other UCs.

- Detection

Where—The UC diagram. **How**—If the number of UCs exceeds the expected range.

- Improvement

1. No corrective actions are required.
2. UCs that contain very little substance should be reformed into uses that offer a complete meaningful service to a system's user. UCs that contain implementation details, such as GUI menu commands should be removed and replaced with analytical UCs that describe what the system needs to do rather than how it does it.

5 Threats to validity

As is the case in any systematic literature review, the main limitations of the review are the incompleteness of the literature domain that was searched, the inaccuracy in the extracted data and the inaccuracy of the synthesis process.

The completeness of the literature domain is mainly affected by; (a) the electronic databases used, (b) the keywords used to conduct the search, and (c) the inclusion and exclusion criteria. There are a constantly increasing number of electronic databases that index scientific publications. It would be rather infeasible to search each electronic database available. In this study, we search ten databases that can be argued to be most well-known databases in the fields of computer science and software engineering. Moreover, it is evident that the high quality publications that are published by revered authors and publications in reputable journals and conferences are available in databases used in this study. Therefore, although some publications may have been missed due to limiting the search to the ten databases, it can be argued that the publications missed would likely not have passed the subsequent quality assessment. Another issue is the set of keywords used to execute searches in databases and the risk of omitting relevant publications. Naturally, the larger the set of keywords used becomes, the greater the number of publications retrieved. The set of keywords used can potentially be expanded to include thousands of keywords. If such an extreme measure was undertaken, then using

thousands of keywords may result in the retrieval of many irrelative publications. It is also not known whether all electronic databases would accept to perform searches using such a large number of keywords. Therefore, the set of keywords used must be limited to just a handful of keywords as was the case in this study. The keywords used in this study are ones that are deemed to be most relevant to the field of use case modeling. Confidence in selecting the set of keywords used in this study is enhanced due to the authors' expertise in the subject area (El-Attar and Millller [18–23]. However, it is important to recognize the keywords in the software engineering field are not standardized and they can be both language and discipline specific. This issue was experienced in this study since the search was conducted in the field of use case modeling, whereby the name of the field consists of the words "use" and "case". Finally, there is an issue with respect to the effectiveness of the inclusion and exclusion criteria used. An improperly designed inclusion and exclusion criteria can result in many relevant studies being omitted. Moreover, an improperly designed inclusion and exclusion criteria can result in some of the relevant data in the literature retrieved being overlooked. In this case study, confidence in the design of the inclusion and exclusion criteria is enhanced due to the authors' expertise in the subject area. The effectiveness of the inclusion and exclusion criteria is also validated during the execution of the search as there were only a very small number of disagreements between the searchers. In general, while a number of validation checkpoints and criteria exist within a systematic review process (such as the validation of the search strategy and the definition of an inclusion and exclusion criterion), the ultimate evaluation of the systematic review process lays with the effectiveness of the resulting information gathered. This can be validated on two fronts: (1) by interviewing UC modeling experts and eliciting their opinion with regard to the resulting set of UC modeling anti-patterns presented; and (3) inquiring whether the UC modeling experts know of additional relative information. This validation approach can also be applied to evaluate the results of future systematic reviews that will be conducted to obtain and develop anti-patterns for other types of models. For this study, a preliminary validation of the results was provided by using the developed anti-patterns in four case studies to improve the quality of the use case models of four distinct real-world systems. Detailed analyses of the four case studies are presented in [13–16]. The results of the case studies indicate that the anti-patterns developed in this study can effectively improve the quality of each of the respective use case models.

The synthesis process in systematic literature reviews is inherently a subjective activity. The synthesis process is greatly affected by the expertise of the searchers in the subject area. In this study, the synthesis process was conducted by the authors' who possessed a great level of expertise in the subject area prior to the search being conducted.

6 Conclusions and future work

A systematic review of the literature on producing high quality UC models is presented. Upon defining the scope and classifying the desired information to be gathered, a search strategy was proposed, executed and the gathered results were subjected to inclusion and exclusion criteria and quality assessment.

The information retrieved from the literature fell into three categories: information that explains UC modeling, its notation, syntactical rules and semantics; information regarding how to properly apply UC modeling, such as best practices, recommendations, quality attributes, patterns and blueprints; and information regarding what not to do in UC modeling, such as mistakes, pitfalls, drawbacks and poor quality attributes.

A total of 61 unique "rules" or "heuristics" were compiled from our search. The strength of evidence for each rule and heuristic retrieved was evaluated by subjecting the source literature to a quality assessment consisting of six screening criterion. The retrieved results were synthesized and compiled into a set of 21 anti-patterns.

Acknowledgments The authors would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum and Minerals (KFUPM) for funding this work.

References

1. Ambler S (2002) Agile modeling: effective practices for extreme programming and the unified process. Wiley, London
2. Anda B, Sjøberg D (2002) Towards an inspection technique for use case models, In: Proceedings of the 14th international conference on software engineering and knowledge engineering
3. Anda B, Sjøberg D, Jørgensen M (2001) Quality and understandability in use case models, In: Proceedings of 15th European conference object-oriented programming, Knudsen JL (ed) pp 402–428
4. Anda B, Dreiem H, Sjøberg D et al. (2001) Estimating software development effort based on use cases—experiences from industry, fourth international conference on the unified modeling language
5. Armour F, Miller G (2000) Advanced use case modeling. Addison-Wesley, Reading
6. Ben Achour C, Rolland C, Maiden N et al. (1999) Guiding use case authoring: results of an empirical study, In: Proceedings of IEEE symposium on requirements engineering
7. Berenbach B (2004) The evaluation of large, complex UML analysis and design models, In: Proceedings of 26th international conference on software engineering, pp. 232–241

8. Bittner K, Spence I (2002) Use case modeling. Addison-Wesley, Reading
9. Brown W, Malveau R, McCormick H et al (1998) Antipatterns, refactoring software, architectures and project crisis. Wiley, London
10. Chandrasekaran P (1997) How use case modeling policies have affected the success of various projects (or how to improve use case modeling), addendum to the conference on object-oriented programming, systems, languages, and applications
11. Cockburn A (1995) Structuring use cases with goals, Tech. Rep. Human and Tech., 7691 Dell Rd, Salt Lake City, UT 84121, HaT.TR.95.1, http://members.aol.com/acockburn/papers/usecaes.htm
12. Cockburn A (2000) Writing effective use cases. Addison-Wesley, Reading
13. El-Attar M (2011) Improving the quality of a biodiversity database system use case model using antipatterns—a case study. Available [Online] at http://www.steam.ualberta.ca/main/research_areas/MAPSTEDI%20Analysis.htm. Last accessed May 2011
14. El-Attar M (2011) Improving the code crawler system use case model using antipatterns—a case study. Available [Online] at http://www.steam.ualberta.ca/main/research_areas/CodeCrawler%20Analysis.htm. Last accessed May 2011
15. El-Attar M (2011) Improving the FAIN system use case model using antipatterns—a case study. Available [Online] at http://www.steam.ualberta.ca/main/research_areas/FAIN%20Analysis.htm. Last accessed May 2011
16. El-Attar M (2011) Improving the quality of a supply chain management system use case model using antipatterns—a case study. Available [Online] at http://www.steam.ualberta.ca/main/research_areas/SCM%20Analysis.htm. Last accessed May 2011
17. El-Attar M (2011) Examples of use case antipatterns (Online). Available at http://faculty.kfupm.edu.sa/ICS/melattar/UCAntipatterns.htm. Last updated Feb 2011
18. El-Attar M, Miller J (2006) Matching antipatterns to improve the quality of use case models, 14th IEEE international requirements engineering conference
19. El-Attar M, Miller J (2006) AGADUC: towards a more precise presentation of functional requirement in use case models, 4th ACIS international conference on software engineering, research, management and applications
20. El-Attar M, Miller J (2008) Producing robust use case diagrams via reverse engineering of use case descriptions. Softw Sys Model 7(1):67–83
21. El-Attar M, Miller J (2009) A subject-based empirical evaluation of SSUCD's performance in reducing inconsistencies in use case models. Empir Softw Eng 14(5):477–512
22. El-Attar M, Miller J (2010) Improving the quality of use case models using antipatterns. Softw Sys Model 9(2):141–160
23. El-Attar M, Miller J (2010) Developing comprehensive acceptance tests from use cases and robustness diagrams. Requir Eng 15(3):285–306
24. Fabbrini F, Fusani M, Gnesi S et al. (2001) The linguistic approach to the natural language requirements quality: benefits of the use of an automatic tool," In: Proceedings of the 26th annual NASA Goddard software workshop, pp 97–105
25. Fantechi A, Gnesi S, Lami G et al. (2002) Application of linguistic techniques for use case analysis, In: Proceedings of IEEE joint international conference on requirements engineering, pp 157–164
26. Firesmith DG (1999) Use case modeling guidelines, In: Proceedings of technology of object-oriented languages and systems
27. Jacobson I, Ericsson M, Jacobson A (1995) The object advantage. ACM Press, New york
28. Kaner C, Bach J, Pettichord B (2003) Lessons learned in software testing. Wiley, New york
29. Khan KS, Ter Riet G, Glanville J et al. (2001) Undertaking systematic review of research on effectiveness, CRD's guidance for those carrying out or commissioning reviews, CRD report number 4, second edn., NHS Centre for Reviews and Dissemination, University of York
30. Kitchenham B (2004) Procedures for performing systematic reviews, Tech. Rep. TR/SE0401, Keele University and Tech. Rep. 0400011T.1, National ICT Australia Ltd
31. Kulak A, Guiney E (2000) Use cases: requirements in context. Addison-Wesley, Reading
32. Lilly S (1999) Use case pitfalls: top 10 problems from real projects using use cases, In: Proceedings of technology of object-oriented languages and systems
33. McCoy J (2003) Requirements use case tool (RUT). Companion of the 18th annual ACM SIGPLAN Conf. on object-oriented programming, systems, languages, and applications, pp 104–105
34. Object Management Group (2005) UML superstructure specification, Object management group, Dec 2005, ver. 2.0 formal/05-07-04, 2005. (Online). Available: http://www.omg.org/docs/formal/05-07-04.pdf. Accessed Dec 2008
35. Overgaard F, Palmkvist K (2005) Use cases patterns and blueprints. Addison-Wesley, Reading
36. Ren S, Rui K, Butler G (2003) Refactoring the scenario specification: a message sequence chart approach, In: Proceedings of 9th object-oriented information systems, pp 294–298
37. Rui K, Butler G (2003) Refactoring use case models: The metamodel, In: Proceedings of 25th Computer Science Conference, Oudshoorn M (ed), pp 4–7
38. Schneider G, Winters J (1998) Applying use cases–a practical guide. Addison-Wesley, Reading
39. Xu J, Yu W, Rui K, et al. (2004) Use case refactoring: a tool and a case study, In: Proceedings of the 11th Asia Pacific Software Engineering Conference, pp 484–491