Are Themes and Use Cases the Same?*

Valentino Vranić
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology
vranic@fiit.stuba.sk

Pavol Michalco
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology
pavol.michalco@gmail.com

Abstract

Theme/Doc and use cases are two prominent approaches to aspect-oriented analysis. They have been developed independently and up to now, there have been no attempts to analyze how they are connected to each other. This paper explores the extent to what themes and use cases can be considered the same by developing a process of the transformation of themes into use cases and the reverse one. Extensive similarities have been revealed between themes and use cases with respect to expressing aspect-oriented decomposition, relationship to functional decomposition, and generalization. Main differences lie in the way themes and use cases are described, naming convention, lack of actors in themes, and lower level character of some themes. Despite the differences, most of the themes can be transformed directly into use cases and vice versa with a quite straightforward derivation of the relationships among them.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Theme, use case, aspect-oriented analysis, feature

1. Introduction

Theme [1] is a comprehensive approach to aspect-oriented analysis and design with its analytical part–Theme/Doc–being a kind of conceptual modeling. Themes, the basic notion of the Theme approach, seem to be close to use cases, a technique of choice in contemporary software analysis. Ivar Jacobson and Pan-Wei Ng showed that use cases not only can be used in aspect-oriented development, but that they are intrinsically aspect-oriented though their application in aspect-oriented development may benefit from some aspect-oriented extensions of the use case modeling notation [4].

A way of transforming a Theme/Doc model into a use case model and vice versa is proposed in this paper showing such a process is feasible and pointing out detailed differences and commonalities between use cases and themes. It demonstrates that despite Theme/Doc lacks the elements that would correspond to actors, flows, or extension points, most themes can be transformed directly into use cases and vice versa with a quite straightforward derivation of the relationships among them. This shows that these two independently developed techniques have common grounds giving more credibility to both of them for their use in aspect-oriented analysis. In practice, however, this means that use cases as a widely accepted analysis technique can be used instead of Theme/Doc even when it is to be followed by Theme/UML, the design part of the Theme approach, with or without an explicit transformation into themes.

The rest of the paper is structured as follows. Section 2 introduces the Theme/Doc approach by a brief example. Section 3 introduces the process of transforming themes into use cases, while Sect. 4 introduces the reverse transformation. Section 5 summarizes equivalence of Theme/Doc and use case modeling mechanisms and discusses potential benefits of each approach. Section 6 discusses related work, and Sect. 7 concludes the paper.

2. Theme/Doc: An Example

Theme/Doc is concerned with the identification and modeling of themes where a theme is a modularization construct that encapsulates a concern [8]. Let us present the example Theme/Doc model used in our study.

2.1 Theme Identification

Assume we are developing a retail support system. The application will embrace store management, price management, and cash desk functionality provided simultaneously on several computers in the store. The specification of requirements is presented in Fig. 1.

The corresponding themes are displayed in Fig. 2. The semantics of a theme is expressed by the requirements it is connected to. Each requirement may involve many

*An earlier version of this paper was presented at the Early Aspects 2010 workshop at the 9th International Conference on Aspect-Oriented Software Development. The paper is recommended by Ruzanna Chitchyan and Steffen Zschaler.

Vraniæ, V., and Michalco, P. Are Themes and Use Cases the Same? Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Early Aspects, R.Chitchyan, S. Zsachaler (Eds.), Vol. 2, No. 1 (2010) 66-71

1. The application will record and maintain the product quantity in the stock in the central database.
2. The storekeeper can remove products from the database.
3. The storekeeper can add products into the database.
4. The storekeeper can change the product quantity in the database.
5. The cashier can bill the item by manually entering the bar code or with a bar code reader.
6. Only the products recorded in the database can be billed.
7. The billed items can be removed from the bill until it has been closed.
8. The billed item removal must be approved by a store manager by entering his authentication data.
9. The billed items will be printed on the cash desk bill as they are entered. The bill will consist of the store name, billed items, information on removed billed items, the total amount of money to be paid, and date and time.
10. The product price can be entered or modified only by a properly authenticated store manager.

Figure 1: The retail support application requirements.

themes, and several requirements may concern the same theme.

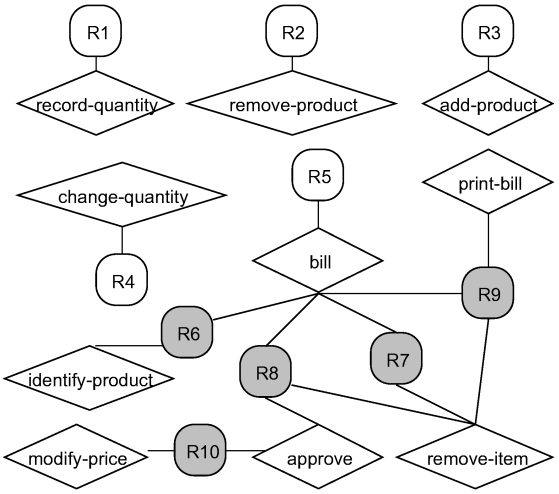

Figure 2: Themes in the retail support application.

2.2 Aspect Separation

The goal of Theme/Doc is to separate the exact semantics of each theme by rewriting the requirements in such a way that each requirement remains associated with only one theme. By this, a set of requirements exclusively associated with a theme virtually become its (only) description. The themes that represent crosscutting concerns cannot be separated this way unless the crosscutting is made explicit by transferring it directly to the affected themes.

There are five shared requirements in our example: R6, R7, R8, R9, and R10. They are marked gray in Fig. 2. R8 is about removing the billed item, while the theme *bill* is about billing in the sense of adding items to the bill, which is covered by R7, so the connection between R8 and the *bill* theme can be omitted. The removed item printing can be separated from R9 into a new requirement R11, so the connection between R9 and the *remove-item* theme can be omitted (see Fig. 3).

9. The billed items will be printed on the cash desk bill as they are entered. The bill will consist of the store name, billed items, the total amount of money to be paid, and date and time.
11. The removed items will be printed on the cash desk bill as they are entered.

Figure 3: The rewritten requirements.

While we managed to remove two connections, we have not fully resolved any shared requirements. If no further requirement rewriting can help to isolate requirements, the remaining requirement sharing is due to the crosscutting nature of some themes. Such themes are identified as being dominant with respect to some requirements they share with other themes. A crosscutting theme remains the only theme associated with the requirements it dominates.

In our example, *print-bill* dominates requirements R9 and R11, while the *approve* dominates requirements R8 and R10. This is displayed in the so-called crosscutting view. Figure 4 presents the crosscutting view in our example (the themes with no shared requirements have been omitted). The crosscutting is denoted by arrows (in the original notation rendered as thick shaded arrows).

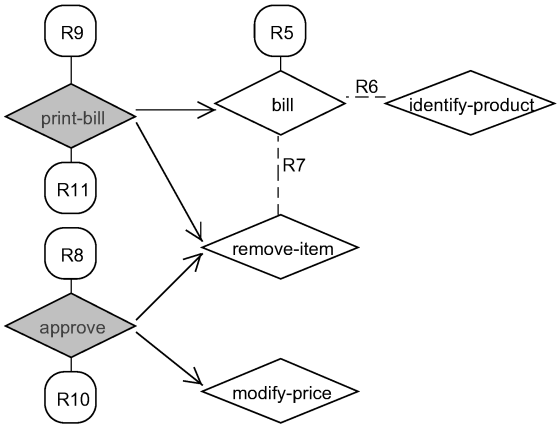

Figure 4: The crosscutting theme view.

There are cases of requirements sharing in which it is not possible to determine the dominant theme. Such cases are indicated by dotted edges between themes marked by respective requirement number and postponed to design. In our example, there are two such requirements: R6 and R7.

3. Transforming Themes into Use Cases

As we saw, both themes and use cases represent functionality. Although use case names are usually quite self-explanatory, while theme names are hard to understand without referring to associated requirements,[1] they are expressed in the same active form.

[1]Newer literature on Theme employs the use case naming scheme [8].

Use cases are not just any functionality, and it may seem themes are, but this holds only in initial phases of automated theme identification. Real themes represent only a fraction of the initial set of themes. The process of use case identification is similar to theme identification. Use cases are even acknowledged as one of the sources of themes [1].

Possible relationships between themes are similar to those between use cases. This is most obvious with the crosscutting relationship between themes and extend relationship between use cases, but also so with grouped and unified themes.

Grouped themes resemble include relationship between use cases. Grouped themes—denoted as *subthemes*—are supposed to be "called" by their *grouping theme*.

Unified themes look as a rudimentary form of generalization. Actually, they may be viewed as specializations of a *unifying theme* created to represent each one of them. For that, unified themes have to be close enough to each other. If they are the same, then the unification is just a synonym substitution.

One significant difference between themes and use cases is that themes lack a direct description whereas use cases are primarily textual. Also, there are no actors in theme models.

With respect to our initial observations, themes can be transformed into use cases according to the following guidelines:[2]

1. Create a use case for each theme. Identify actors in requirements.

2. Create an extend relationship for each crosscut relationship found in the crosscutting view preserving its direction.

3. Consider splitting themes. Identify grouped themes in individual theme views (both the existing ones and those obtained in step 1). Consider transforming each theme–subtheme relationship into an include relationship or into a generalization relationship if the theme and subtheme conceptually represent the same theme. Deciding not to transform the subtheme means deciding its functionality will be an integral part of the existing use case possibly as a separate flow.

4. Consider unifying themes. Identify unified themes in the history of the operations performed upon the theme model if it is available. Consider transforming unified themes into generalizations.

5. Consider the granularity of the obtained use cases and restructure them as necessary by including too low level use cases as flows of regular ones.

6. If not resolved by previous steps, resolve the postponed relationships as include, extend, generalization, general relationship, or dismiss them.

[2]An initial version of this process has been proposed in our earlier work [7].

3.1 The Initial Use Case Model

Figure 5 shows the use case model of our example application obtained by applying the first two steps of the transformation. Although it is not necessary to do so, names of the use cases obtained by transformation can be adjusted. In our example, the correspondence between the theme and use case names is quite straightforward.

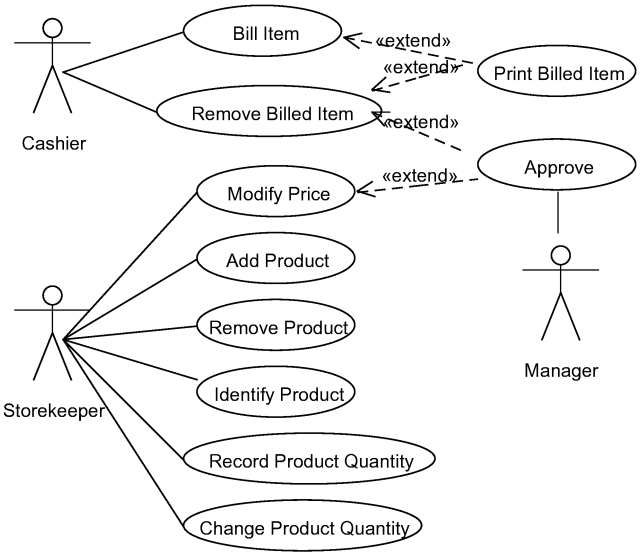

Figure 5: Identified use cases and extend relationships.

We omit determining extension points since we do not have the actual flows in use cases. However, there is a sufficient understanding of each extend relationship to provide a Cockburn style [2, 9] descriptive reference to extension points in extending use cases. For example, we could say that *Print Billed Item* is activated each time an item is added to a bill or removed from it.

Base themes (that do not appear as subthemes) in the initial process correspond to peer use cases (use cases with no direct relationships between them [4]), so no special effort is needed in this direction.

3.2 Subthemes

There are no grouped themes (step 3 of the transformation) in our example, but examining themes for splitting resulted in three subthemes displayed in Fig. 6. We actually separated (implicit) product identification in the context of each of these three themes. For simplicity, we ignore the data entities themes operate on. If present in the model, they should be referred to in the use case flows.

We could create a separate use case for each subtheme, but a closer look at them reveals they represent the same functionality named differently merely due to Theme/-Doc not allowing multiple themes with equal names. This means that the subthemes we have just separated out have to be unified and given a common name. A logical choice is *identify-product*, which points us lexically to the equally named theme already present in our model. We acknowledge these three themes are one and only theme and recognize it corresponds to the *Identify Product* use case, which has been already present in the model. We also dropped the association of this use case to the actor because we revealed it cannot be activated separately

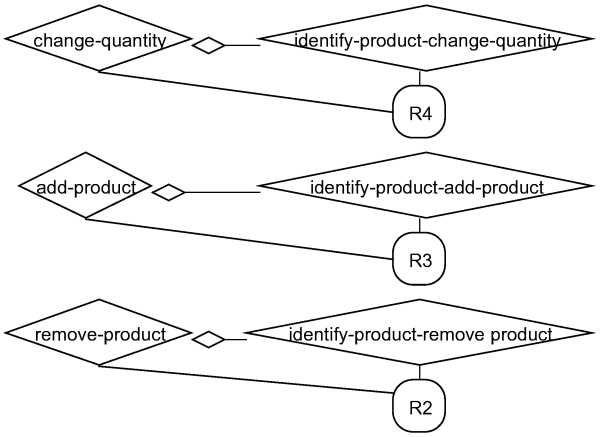

Figure 6: Splitting the themes.

(Fig. 7).

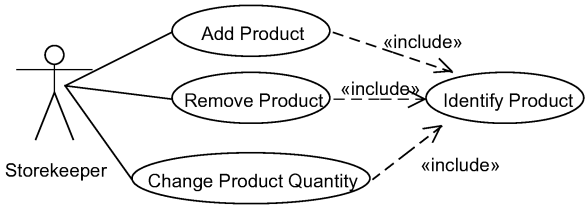

Figure 7: Inclusion.

3.3 Unified Themes

There are no unified themes in our model (step 4 of the transformation), but there is one opportunity for unifying themes. Both *bill* and *remove-item* themes are about billing an item: both of them add a record to the bill, and both of them add a value to the total sum of the bill, though a negative one in the *remove-item* theme. We separated the common behavior into an abstract use case named *Bill Item* and derived two concrete use cases from it that correspond to the two kinds of billing operations as shown in Fig. 8.

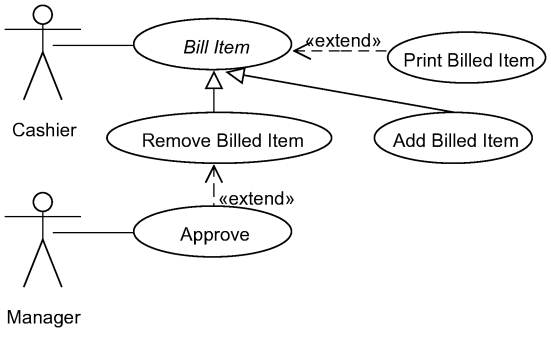

Figure 8: Generalization of use cases.

3.4 Granularity

Themes tend to be lower level than use cases, so it is advisable to check whether all the use cases obtained so far are at an appropriate level (step 5 of the transformation). Taking the *Bill Item* use case as an example, we may note that it would be more natural to have billing a whole purchase as a use case. Here we have two possibilities: to replace all three billing use cases with one, integral use case called *Bill Purchase* or to introduce the *Bill Purchase* use case and keep the existing use cases as inclusion-only use cases.

Formerly separate use cases may persist as basic flows in the integral use case. This is not an uncommon practice in use case modeling often applied to develop so-called CRUD use cases that describe typical create-read-update-delete operations by a separate base flow [10]. Both solutions are correct and may be viewed as a matter of style.

4. The Reverse Transformation

Since a use case model corresponds to the combination of the crosscutting and individual theme views, its transformation into a theme model will result in these two views. Subsequently, the theme–relationship view will be derived from them. The transformation steps are as follows:

1. Identify themes by transforming each use case not involved in a generalization into a theme and transforming each generalization among use cases into unified themes. Optionally rename themes by shortening the corresponding use case names. Drop actors.
2. Create the crosscutting view by transforming each extend relationship between use cases into a crosscutting relationship between the corresponding themes preserving its direction.
3. Create the individual view by transforming each include relationship between use cases into a theme–subtheme relationship preserving its direction. Derive the data entities the theme operates on from the use case flows and attach them to the corresponding themes.
4. Transform all requirements use cases refer to into requirements in the theme model. Transform each use case to requirement relationship into a relationship between the corresponding theme and requirement.
5. Derive the theme–relationship view by including all the themes in the crosscutting view and identifying shared requirements. Transform each unspecified dependency between use cases into a postponed relationship between the corresponding themes preserving its direction.

4.1 Generalizations

The transformation of generalizations among use cases relies on the inversion of the two schemes applied in the themes to use cases transformation (see Sect. 3.3). In one, the common behavior among two or more themes is separated into an abstract use case with derived use cases corresponding to themes, while in the other one use cases are derived one from another corresponding to the generality of themes.

4.2 Extend and Include Relationships

Theme/Doc is capable of representing chained extend relationships with the chained theme crosscutting. However, chained extend relationships are rare in use case models.

On the other hand, chained include relationships represent a problem since Theme/Doc supports only one level

Table 1: Equivalence of Theme/Doc and use case modeling mechanisms.

Theme/Doc	Use Case Modeling
base theme	peer use case
requirement	brief description/flow
crosscutting theme	extending use case
grouping theme	including use case
grouped theme	included use case
unifying theme	general use case
unified theme	special use case
subtheme	inclusion use case
crosscutting relationship	extend relationship
theme–subtheme rel.	include relationship
theme–requirement rel.	use case to requirement link
postponed relationships	any/no relationship
n/a	actor

of subthemes. Although chained include relationships indicate functional decomposition, which should be corrected directly in the use case model, there are two ways to deal with them during a transformation: to keep the subtheme connected to the theme and have it include all other subthemes or to keep the leaf subthems and have them include all other subthemes.

4.3 Requirements

The themes derived from use cases have no description and as such are hardly understandable to the parties not involved in the transformation. There are several possibilities to derive theme descriptions during the transformation of use cases into themes.

Deriving Theme/Doc style requirements from use cases and requirements attached to them would obviously require a lot of effort. If use cases are linked to respective requirements, the initial set of requirements can be obtained by attaching these requirements to themes.

However, it is questionable whether the requirements in the Theme/Doc style are needed at the stage at which there is a Theme/Doc model with completely separated crosscutting. A regularly developed theme model at this stage encompasses highly restructured requirements that reflect this separated crosscutting and it may be very hard to arrive at desirable formulations of requirements.

Describing each theme according to the understanding of its meaning still requires some effort, but not as big as in mimicking the exact Theme/Doc requirement style. Attaching use cases or just their brief description instead of requirements to the corresponding themes may be considered as the quickest yet viable solution.

5. Discussion and Comparison

Table 1 summarizes equivalence of Theme/Doc and use case modeling mechanisms. A use case model as such corresponds to the combination of the crosscutting and individual theme view.

Main similarities between themes and use cases include the way they express aspect-oriented decomposition, their relationship to functional decomposition, and generalization. With respect to aspect-oriented decomposition, it is remarkable how both themes and use cases accommodate both asymmetric and symmetric aspect-oriented decomposition with equivalent mechanisms (base themes/peer use cases and crosscutting relationship/extend relationship).

One of the main differences between themes and use cases is in the way they are described. While themes are described indirectly by restructured requirements, use cases are described by flows of events. Further differences include naming convention, lack of actors in themes, and lower level character of some themes. While non-functional requirements can be captured by so-called infrastructure use cases [4], the Theme/Doc approach does not seem to deal explicitly with them, which means they have to be postponed to design.

Theme/Doc might be closer to natural language specification, while use cases tend to enforce some design/implementation mechanisms such as generalization (object-oriented) or extension (aspect-oriented). Although these mechanisms receive quite a lot of attention, use cases are primarily textual and can be succesfully expressed without employing them [2].

Theme/Doc may seem attractive as it promises automated theme model generation from requirements [1]. However, tool support for Theme does not seem to be available at all,[3] and even if was, generated models that were reported were huge and required a lot of manual work [1] heavily compromising the potential benefit from the automatic generation.

6. Related Work

Baniassad and Clarke introduced use cases as one of the sources of themes and gave a hint that each use case or action (perhaps a use case step or flow) can become a theme [1]. Apart from this, we encountered no attempt to bring use cases to a direct connection with themes, nor to formulate the transformation between them.

Themes are declared to be close to features in the sense of feature modeling (having its roots in FODA [5]) or even referred to as features [1]. Griss et al. [3] view features as more general than use cases as they represent not only the user visible behavior as use cases tend to, but also system internal structure and realization (observed also by Baniassad and Clarke). Lopez-Herrejon and Batory see a theme as a collection of structures that represent a feature [6]. The connection of use cases to features and features to themes indirectly confirms relatedness of use cases to themes.

7. Conclusions and Further Work

The transformation process of the Theme/Doc model into the use case model and vice versa presented in this paper confirmed that themes are very close to use cases, which has been anticipated by an initial comparison of these two notions as well as by other literature, but haven't been studied deeper so far.

While Theme/Doc may still be interesting for being potentially less design dependent, lack of tool support and general acceptance of use case modeling in analysis pushes the choice between the two significantly towards use cases.

[3] `http://www.thethemeapproach.com/downloads.html`

As a further work, we consider extending the transformation towards Theme/UML to explore the relationship of themes in Theme/UML to use case slices [4]. The excess information in use cases could provide a part of the information needed for modeling Theme/UML themes, while they can provide flows to use cases. It would also be interesting to explore to what extent features (in feature modeling) cover the properties of themes in which they differ from use cases.

8. Acknowledgments

This work was supported by the Scientific Grant Agency of Slovak Republic (VEGA), grant No. VG 1/0508/09 and it is a partial result of the Research & Development Operational Program for the project Support of Center of Excellence for Smart Technologies, Systems and Services II, ITMS 25240120029, co-funded by ERDF.

I would like to thank Ruzanna Chitchyan and Steffen Zschaler for their valuable suggestions.

References

[1] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005.

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

[3] M. L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In P. Devanbu and J. Poulin, editors, *Proc. of 5th International Conference on Software Reuse*, pages 76–85, Victoria, B.C., Canada, 1998. IEEE Computer Society Press.

[4] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA): A feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, Nov. 1990.

[6] R. E. Lopez-Herrejon and D. Batory. Modeling features in aspect-based product lines with use case slices: An exploratory case study. In *Proc. Models in Software Engineering: Workshops and Symposia at MoDELS 2006*, Genoa, Italy, Oct. 2006.

[7] P. Michalco. Transforming aspect-oriented analysis in Theme/Doc into use cases. In *Proc. of Informatics and Information Technologies Student Research Conference, IIT.SRC 2009*, Bratislava, Slovakia, Apr. 2009. `http://fiit.stuba.sk/~vranic/proj/dp/Michalco/dp.pdf`.

[8] P. Sánchez, L. Fuentes, A. Jackson, and S. Clarke. Aspects at the right time. *Transactions on Aspect-Oriented Software Development*, IV:54–113, 2007.

[9] Ľ. Zelinka and V. Vranić. A configurable UML based use case modeling metamodel. In M. Weske and P. Liggsmeyer, editors, *Proc. of 1st IEEE Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2009*, pages 122–137, Novi Sad, Serbia, Sept. 2009. IEEE Computer Society.

[10] G. Övergaard and K. Palmkvist. *Use Cases: Patterns and Blueprints*. Addison-Wesley, 2004.