Recommending Alternative Scenarios Using Use Case Patterns

Deokyoon Ko1, Sooyong Park1, Suntae Kim2*, Malrey Lee3, Younghwa Cho4

1 Dept. of Computer Science & Engineering, Sogang University,
Seoul, South Korea
{maniara,sypark}@sogang.ac.kr

2 Dept. of Software Engineering, Cheonbuk National University,
Jeonju Si, Jeollabuk Do, South Korea (Corresponding Author)
{skim}@jbnu.ac.kr

3 561-756, RCIT, School of Electronics & Information Engineering,
Chon Buk National University, 664-14, 1Ga, Deokjin-Dong, Jeonju, Chon Buk, Korea,
{mrlee}@jbnu.ac.kr

3 Collage of Information and Communication Engineering,
SungKyunKwan University, 2066 Seobu-Ro, Jangan-Gu, Suwon, Gyeonggi-Do, Korea
{shoyh2285}@skku.edu

Abstract. An alternative scenario in use case specification indicates abnormal or less frequent behaviors in a software system. It has an influence on not only customer’s satisfaction but also software quality. However, alternative scenarios are frequently missed because of the author’s mistake or lack of experiences. In order to tackle this issue, this paper proposes a use case pattern based approach to recommending alternative scenarios. We adopt six essential use case patterns and extend them to identify the pattern type from ordinary natural language based scenario. We propose four-step approach using the word similarity technique to find out the use case patterns of the basic scenarios.

1 Introduction

Completeness is one of the key properties for a successful requirements specification [5]. To develop complete software, the requirement must contain all necessary information for the developer or the reader to comprehend it [3]. Complete software requirements can be also said that most of or all user’s requirements are specified in software requirements specification. For specifying requirements, a scenario based requirements specification is a general approach, composing of a basic scenario that describes user-system interactions for the frequent and successful use of the system, and alternative scenarios mentioning interactions for diverse abnormal and less-frequent use of the system. However, in many cases alternative scenarios are omitted, which impedes achieving complete requirements specification.

There has been some work on recommending alternative scenarios of software requirements at the earlier phase of software development [8][6][1]. One research de-
fines a template as an alternative scenarios in associated with a user and action types in advance. Depending on the user and actions in a basic scenario, the templates are produced. However, as all possible associations are previously defined, it has a limitation on the flexibility. Another research tried to author the domain knowledge in detail in advance, and then diverse scenarios are generated from the knowledge. This approach, however, cannot produce appropriate scenarios for a project, as it is impossible to define all possible domain knowledge in advance. I. Alexander stressed the importance of examples regarding the current project in identifying alternative scenarios. According to his experimental result, identifying alternative scenarios with consulting other system’s alternative scenarios is drastically effective.

To address above issues, this paper proposes a use case pattern based approach to generating alternative scenarios from a basic scenario. A use case pattern defines six patterns of recurring essential user-system interactions discovered in diverse requirements specifications (e.g., alarming, requesting or monitoring) [2]. For automatic generation of alternative scenarios, our approach obtains the basic scenario of scenarios from users to be validated. It extracts a main verb of each scenario at first, identifies candidates of patterns through comparing occurrence patterns of agents in each scenario, and analyzes the most appropriate pattern combination by measuring verb-distances. Alternative scenarios are automatically generated from pattern combination.

2 Background

A scenario is a narrative description of interactions between users and a system by using a natural language [7][6]. As a scenario is described in a natural language, an unclear agent in a basic scenario and an alternative scenario causes one to misunderstand a subject of the system or user’s behavior. It may make software developers to develop a wrong system due to the inverted subjects [4]. To ensure the agent of behaviors, Constantine et al. suggested the essential use case that clarifies a use case specification by separating a scenario into user intentions and system responses. Also, it summarizes a use case specification and extracts essential interactions between a user and system from the use case specification [4].

R. Biddle et al. suggested six essential use case patterns based on the sequence of the essential use case interactions and its intentions [2]. It enables one to write better essential use cases and validate it effectively for the sake of correctness and completeness. The following summarizes the six patterns:

– Request: When a user requests information to a system, the system provides requested information.
– Monitor: A system displays its information.
– Alarm: A system warns to a user.
– Command: When a user requests to modify information, system performs the request.
– Prompting: A system offers a prompt for a user to enter his/her decisions.
– Confirming Step: When information is updated, a system and a user confirms the updates.
3 Recommending Alternative Scenarios

The scenario author’s mistake or insufficient experience of the author causes one to have incomplete requirements specification. In order to address the issue, this paper proposes a use case pattern based approach for automatically recommending alternative scenarios from the basic scenarios that an author wrote. Figure 1 shows the overview of the suggested approach. Each steps of the approach has responsibilities as bellows:

1) Main Verb Extractor: It aims at identifying the main verb of basic scenario. The main verb can represent the sentence because it determines the action type. For this step, the natural language parser is applied.

2) Candidate Patterns Matcher: This step is to identify the candidate use case patterns from the basic scenario. This step identifies the candidate patterns based on the occurrence pattern of the agent (user or system) in the basic scenario. One basic scenario can include multiple patterns. This step finds possible pattern combination.

3) Verb Distance Calculator: To find the one pattern combination among candidates, this step calculates the similarity between verbs of candidate patterns and the main verb of the basic scenario. If the two verbs are same or a synonym, the similarity score is each 2 or 1.5, otherwise the score is calculated by using word distance [9].

4) Optimal Pattern Extractor: To find the applied pattern among the candidates, we have to find which pattern combination has the maximum similarity score. In this step, the modified shortest-path algorithm is used.
After carrying out these steps, alternative scenarios for each pattern are attached to the original basic scenario.

4 Conclusion

In this paper, we suggested an approach to automatically recommending alternative scenarios from the basic scenario in order to improve the completeness of software requirements specification. The approach of this paper is validated four experts and the generated scenarios are obtained high significance score (3.1/5.0) on average. We are planning to automatically/statistically extract scenario patterns from the large volume of scenarios in an automatic way by applying machine-learning based methods as a future work.

Acknowledgments. This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030503).

References