A Method to Analyze Variability Based on Product Release History: Case Study of Automotive System

Kentaro Yoshimura, Fumio Narisawa, and Koji Hashimoto
Hitachi Research Laboratory, Hitachi, Ltd.
(MD#244) 7-1-1 Omika, Hitachi, Ibaraki 319-1292, Japan
kentaro.yoshimura.jr@hitachi.com

Tohru Kikuno
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
kikuno@ist.osaka-u.ac.jp

Abstract

Variability is a keystone for developing reusable software product line (SPL) assets. Top-down approaches like feature-oriented analysis are widely used for detecting variability. However, when we introduce the SPL approach into existing products, analyzing variabilities of requirements and comparing them to legacy assets depend on the expertise of the analysts and are time consuming.

We present an complemental approach to analyze the variability candidates from existing product release history. We apply factor analysis method to detect co-change patterns of the legacy artifacts across product releases, and to suggest the variability candidates. To examine the applicability of our approach, we conducted an experimental application using a software repository of automotive engine-control software. As a result of the experiment, four variabilities and their variation points are detected successfully.

1 Introduction

SPL approach has been proposed as a development method for software that has variations. For reusing software across a product line, variabilities that are differences in requirements between product variations are analyzed. For migrating from existing products to SPL, understanding variability of the existing products and variation points of the artifact is an important practice.

Research in SPL engineering has mostly focused on the construction of product line infrastructures and activities based on requirements of future products: scoping, domain analysis, architecture creation, and variability management [2][3][8]. On the other hand, existing products contain a lot of domain expertise and are reliable from an industry point of view. The commonality and variability analysis for existing software is one of the most important issues to define a future product line while reusing existing software.

Variability analysis of existing software is a block against migrating into SPL engineering. Related studies have proposed methods for analyzing commonality and variability from a requirement point of view and connecting that to the implementation [6][7]. However, requirements and implementations of existing software are enormous. Moreover, such requirement analysis strongly depends on the expertise of the analysts and is time consuming. Some industrial case studies reported that commonality and variability analysis takes a long time[11][12]. There is a need for analyzing the commonality and variability in an automatic way.

We focus on the release history of existing products. Existing products contain their variability in their change history. If we could extract the variability from the existing products, that will be useful information for migrating the software product line because most of the existing variability information will also be valid in future product lines. Specifically, we analyze the product release history. When the bindings of variability were different between products releases, the realization of the software components that relates to the variability was also different. A major variability should occur several times, and the relationship to the software components (variation point) could be detected as a significant change pattern in the history.

We developed a method to suggest variability candidates across existing software with analyzing the co-change

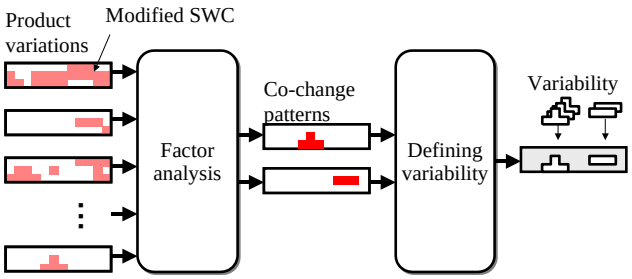

Figure 1. Overview of proposed method

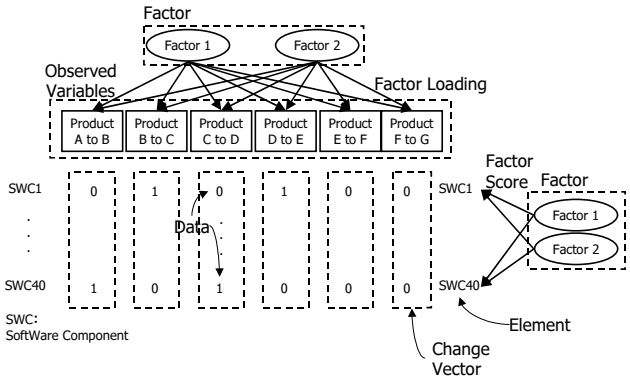

Figure 2. Overview of factor analysis

pattern[14]. The idea is inspired by factor analysis. Factor analysis is a multivariable analysis technique used to explain commonality among observed variables in terms of fewer unobserved variables called ”factors.” The observed variables are modeled as linear combinations of the factors plus ”specific variables.” The factor analysis extracts the co-change pattern as the commonality of changes between products and a set of software components that have changed with variability candidate. We thus tried to apply the factor-analysis technique to detect variability in existing products and call our approach ”FAVE: Factor Analysis based Variability Extraction”.

In this paper, we apply FAVE method to an industrial case study and discuss its results. The previous paper [14] describes FAVE method and applies it to an industrial, but small case study. In contrast, we apply the method to relatively large example in order to evaluate the method and to analysis the future works.

An overview of the proposed method and scope of this paper are shown in Figure 1. Inputs of FAVE are the change history of the products, and the output is variability in the product line. The factor analysis extracts the change pattern as commonality of changes and a set of software components that have changed with variability. After that, the detected variability is refactored as a variation point and its variants.

This paper describes a brief overview of the FAVE method and an experimental application to the software repository of automotive engine-control software. This paper is structured as follows: Section 3 summarizes related work. Section 2 describes a brief overview of the factor-analysis technique. Section 4 describes our FAVE method for detecting variability in existing products. The application of the proposed method to existing automotive control software is the topic of Section 5. Section 6 describes our work in progress and a conclusion.

2 Factor Analysis

An overview of the factor analysis is shown in Figure 2. Factor analysis is a well-known multivariable analysis technique used to explain commonality among observed variables in terms of fewer unobserved variables called ”factors”.

A factor loading is the degree of correlation between the observed variable and the factor. The factor has a strong relationship with the observed variable if its factor loading is significantly high.

Each factor also correlates with elements of the observed variables. The degree of correlation between the element and the factor is called a factor score. Using the factor score, we can expect that the meaning of the elements is based on the meaning of the factor.

3 Related Work

SPL engineering is an investment in future products, so various researchers have studied roadmap-oriented product line engineering[2][3][10]. There are also many studies about product line engineering for legacy software. Kang[7] analyzed the requirements of the legacy system and modeled them as a feature tree to migrate into SPL. John[6] analyzed documents of the legacy system to detect variability. Steger[11] analyzed electric and electronic architecture of the controller as variation points of the system. However, researchers have started mainly from current products to future products. In contrast, our approach focuses on the release history of the product line from the past to the present and analyzes how the product have varied. This result of the extraction helps developers understand the variability of the existing products.

We have already developed a method for evaluating commonality between two current products in a quantitative way [13]. In the proposed method, the commonality between products was detected, but we were unable to detect the variability across the variants. The difference between products consists of variability and product-specific parts. We could not separate them even when analyzing only two

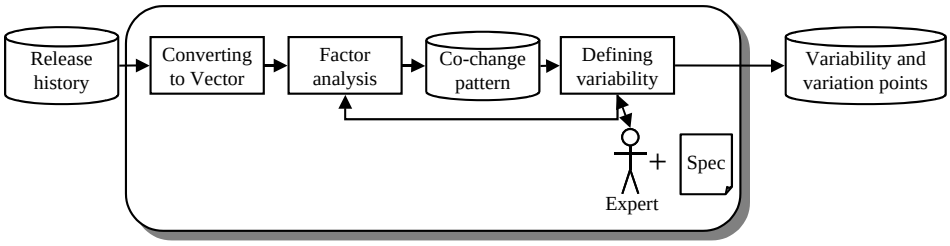

Figure 3. Overview of FAVE process

products. In contrast to our previous work, FAVE extracts the variability from the differences between multiple existing products.

In [4], Fischer proposed an approach to detect coupling between modules of multiple product variants by mining their code repositories. Their approach detects the degree of coupling between modules across the change history of the variants. In contrast to their work, FAVE detects the variability among the product release history in the orthogonal region and the variation points as correlation to the software components.

Loesch[9] presented a method to optimize variability provided in a product line. Their approach analyzes the usage of variable features in actual products derived from the product line. They applied a formal concept-analysis method for optimization. In contrast, our approach analyzes the release history of the existing products that have not been migrated into SPL yet.

Independently from SPL, Zimmermann[15] developed an approach to detect couplings of software components from the version history. However, their focus was on checking software components that were often modified at the same time. In contrast, our approach also clusters the change pattern of legacy software and extracts that as a variation point in the concept of SPL.

4 FAVE - Factor Analysis based Variability Extraction

4.1 Overview

The purpose of FAVE is to detect product-line variability that has already occurred in existing products. An overview of the FAVE process is shown in Figure 3. We have already developed the method in our previous work [14], so we describe the process briefly in this section.

Variability analysis is the first step for migrating existing products into software product lines. In this step, we analyze the existing product artifacts and classify them into common parts, variable parts, and product-specific parts. The aim of FAVE is the variability mining of existing products.

As shown in Figure 3, we convert product release histories to change vectors so that we could analyze them numerically. Then, we apply the factor analysis to the change vectors and identify co-change patterns as the variability candidates of the product line. After that, we define the means of the variability from the viewpoint of requirements and specifications.

In the following subsections, we explain briefly how FAVE is applied to existing products.

4.2 From Product Release History to Change Vectors

Product release history is not numerical data, so we cannot apply the factor analysis directly. For analyzing variability based on factor analysis, we convert the release history to change vectors.

An overview of the product release history is shown in Figure 4. Typically, an organization develop new product from the released product that has similiar requirements. Based on the difference of the requirements, e.g. different mechanical parts or new functionality, some software components (SWCs) are altered, added or deleted.

We define a change vector as a set of binary data that indicates difference in software components between product releases. For example, in Figure 4, software component 1, 2, 3 and 5 are changed from product A to B. Therefore, the change vector of product A to B is $(1, 1, 1, 0, 1, ...)^T$.

Product release history often branches because of the pararell development of product variations. In that case, change vector is generated from the difference between the last and the first product release over the branch point. For example, in Figure 4, software component 1, 4 and 5 are changed from product B to D. Therefore, the change vector of product B to D is $(1, 0, 0, 1, 1, ...)^T$.

After change vectors are generated based on changes between all serial product release, the vectors are used as observed variables for the factor analysis in the next step.

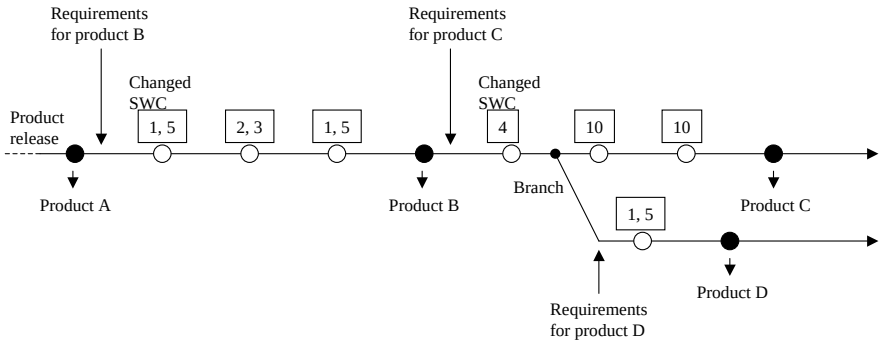

Figure 4. Product release hisotry

4.3 Factor Analysis

After converting the release history, we apply the factor analysis. As a result of the factor analysis, we obtain the following data.

- **Factor**
  A factor indicates a commonality of the changes between existing products. We can extract the factor as variability candidate in the product line because commonality of the change vector is a co-change pattern of how the existing software changes across the release history.

- **Factor Loading**
  A factor loading indicates which change vector is correlated with the variability candidate. When the factor loading is high, the variability candidate may be related to some specifications that changes in the change vectors.

- **Factor Score**
  A factor score indicates which software components are correlated with the variability candidate. Components that have high factor scores were modified together, and the timing is related to the variability candidate. This suggests that a cluster that consists of software components is a variation point of the variability candidate.

4.4 Defining Variability

A process of how domain experts define the means of the extracted variability based on the result of the factor analysis is shown in Figure 5.

First, significant factors are selected for defining the variability. The details of the selection process depend on the analysis method of the factor analysis such as the promax method and varimax method. For example, the number of significant factors is decided by using the chi-squared test and cumulating ratio of the factors.

Next, we analyze the product release histories that correlate the factors. We select the change vectors that have high factor loadings. The factor may be a variability that occurred in the selected change vectors between the existing products. In this step, we do not go into the details of the release, but define which change vectors are related to the variability.

Then, we select the set of software components that indicates the variation point. If the factor score is significantly high, the software component has been modified when a variability occurred in release histories that has a high factor loading. In the context of SPL, a set of the software components means the variation point of the variability. In the next step, we also speculate on the meaning of the variability candidate by referring to the combination of the software components' features.

As the last step, the candidate is interpreted as a variability of the SPL. For example, we guess the meanings of the candidate as indicating the abstracted feature from the set of the software components that have high factor scores. Then, we check whether the variability has occurred in the change vectors that have a high factor loading. Although only this step is an iterative step and depends on the expertise of an analyst, the result of the factor analysis will be helpful information for the analyst.

5 Case Study

5.1 Overview

An overview of an engine-control system is shown in Figure 6. The system monitors engine status and driver requests, and controls the engine by regulating the amount of fuel injection, ignition timing, and quantities of intake air, for example. There are a number of variations that correlate to the mechanical structure, which satisfy the product spec-

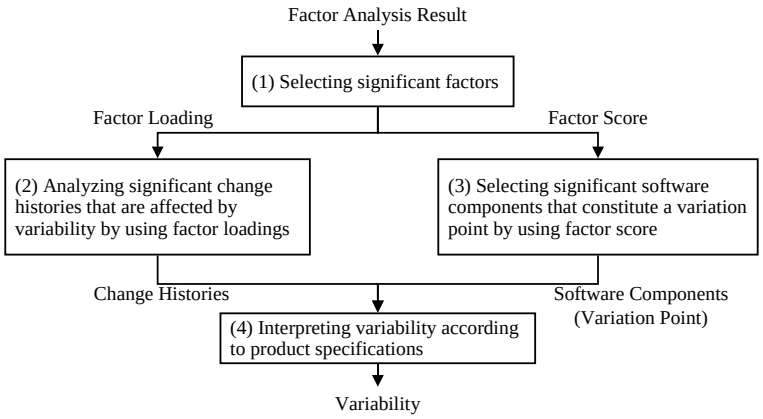

Figure 5. Process for defining variability

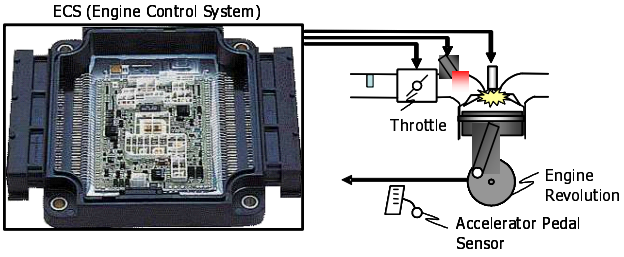

Figure 6. Engine-control system

Table 1. Case study example

Application	Engine-control software
# of products	16
# of software components	49

Table 2. Case study environment

Environment	R ver 2.6.1
Method	Maximum likelihood estimation
Rotation	Orthogonal (varimax)
Scores	Regression

ification, e.g., number of cylinders, transmission type, fuel type, and fuel injection.

We apply FAVE to part of the engine-control application layer. Generally, an engine-control system consists of a basic software layer and application software layer. As defined by an industrial standard for automotive control software [1], most software components in the basic software layer correspond to the electronic architecture of the control system, such as microprocessors, LSIs, and network protocols. The variability of basic software is easily visible and the relationship between the variability and variation point is simple in this example. On the other hand, a feature of the application layer calculates engine phenomena, i.e. intake airflow, fuel combustion, and exhaust gas. These phenomena are correlated to many kinds of physical components such as engine size, number of cylinders, valves, fuel injectors, and fuel type. Moreover, there are some nonfunctional requirements that cause variability such as exhaust gas regulation by different countries and drivability. Therefore, the features in the application layer have much invisible variability, and the correlations between the feature and the variability are complex. Thus, we apply FAVE to extract the variability of a feature in the application layer and the mappings to software components of the feature.

An overview of the case study is shown in Table 1. The repository has data about sixteen products that were released for different vehicles. We extracted one control subsystem that consists of forty-nine software components. Each software component is responsible for a fine-grained feature of the engine or the control system itself. To compute the factors, we applied R [5] and configured the parameter, as shown in Table 2.

5.2 Analyzing Variability

We analyzed the variability by following the process in Fig. 3.

5.2.1 Converting to Change Vector

First, we converted the release histories to the change vectors. We obtained fifteen change vectors from sixteen releases of existing products from the software repository. The release of software corresponds to different vehicles in the market.

Due to a company confidentiality issue, we are unable to disclose the organization of the software repository and change vectors.

Table 3. Examination result for # of factors

# of factors	1	2	3	4
p-value	1.43E-24	5.74E-16	5.39E-8	0.0105
X squared	301.1	220.2	141.7	77.15
Cumulating	0.229	0.359	0.501	0.594

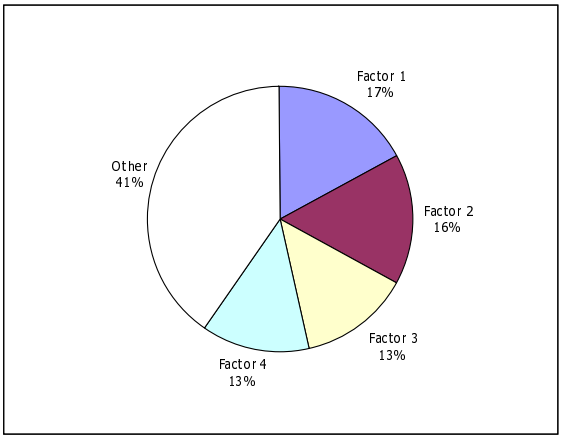

Figure 7. Factor proportion

5.2.2 Factor Analysis

- Number of factors

  Next, we examined the number of significant factors of the product line. We applied the chi-squared test and checked the cumulative variable of the factors. The result of the examination is shown in Table 3. When the number of factors is four, p-values of the chi-squared test are larger than 0.01 and the cumulating value is 0.594. Therefore, we defined the numbers as ”4”.

  The percentages of factors are shown in Fig. 7. The other than the factors indicates specific changes for each change history. From the viewpoint of the SPL, this portion corresponds to product-specific modifications.

- Factor 1

  Next, we define the variabilities for each factor based on the factor loadings and scores. Because of the page limit, we explain about factor 1 and 2 case only.

  The loading of factor 1 is shown in Figure 8. Change vectors F, H, and J correlate with factor 1. The experimental result suggests that variability relating to factor 1 has occurred in change vectors F, H, and J.

  The score of factor 1 is shown in Figure 9. Some software components have a factor score significantly higher than 1.4, and the other components have a low factor score. The experimental result suggests that

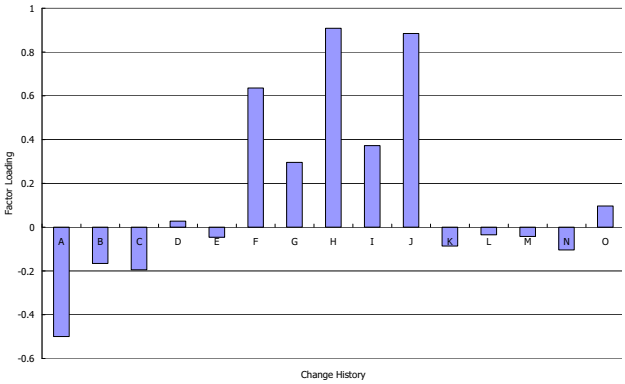

Figure 8. Factor loading (factor 1)

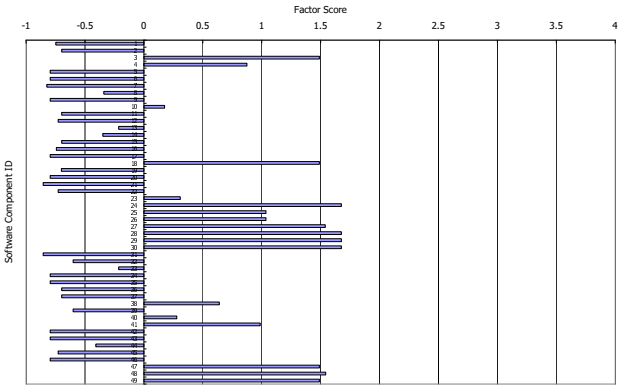

Figure 9. Factor score (factor 1)

software components No. 3, 18, 24, 27, 28, 29, 30, 47, 48, and 49 are correlated with each other. This means that there is a set of software components that have been changed together in the change vectors and FAVE extracted the set automatically.

Finally, we define the variability of factor 1. We select software components No. 3, 18, 24, 27, 28, 29, 30, 47, 48, and 49 as the variation point of the variability based on the factor score. Then, we check the features of the components and suppose what kind of variability is related to an abstracted feature. From the control specification point of view, software components No. 24, 27, 28, 29, and 30 are strongly related to the variable valve timing control feature and the others are related to the phenomenon of intake airflow. Therefore, we checked how the products are modified in the change histories F, H, and J, and then found that the valve system was modified from the fixed type to the variable type and from the variable type to the fixed type. Therefore, we defined factor 1 as the variability of ”Variable Valve Timing Control” and the selected software components as variation points.

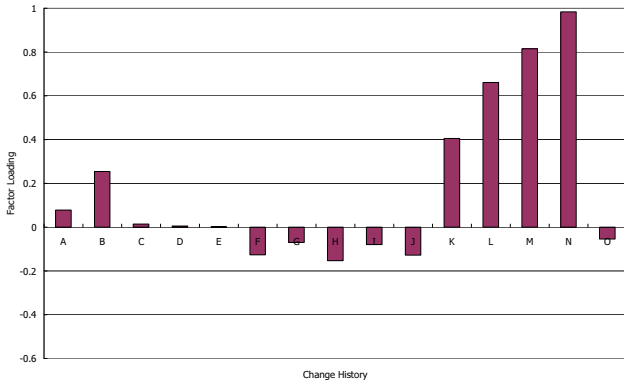

Figure 10. Factor loading (factor 2)

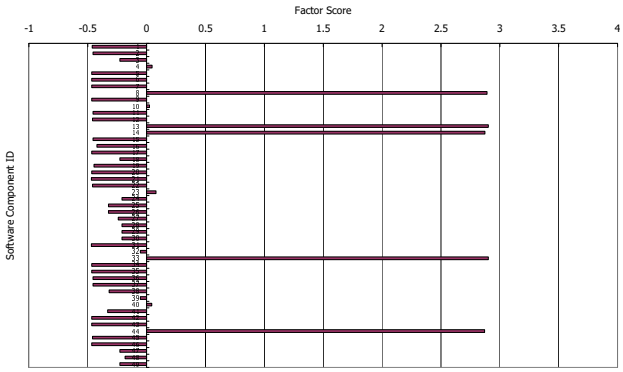

Figure 11. Factor score (factor 2)

- Factor 2

  The loading of factor 2 is shown in Figure 10. Change vectors L, M, and N correlate with factor 2. The experimental result suggests that variability relating to factor 1 has occurred in the change vectors L, M, and N.

  The score of factor 2 is shown in Figure 11. Software components No. 8, 13, 14, 33, and 44 have a factor score significantly higher than 2.8, and the other components have a low factor score. The experimental result suggests that software components No. 8, 13, 14, 33, and 44 are correlated with each other. Again, this means that there is a set of software components that has been changed together in the change vectors and FAVE extracted it correctly.

  We select software components No. 8, 13, 14, 33, and 44 as the variation point. All of them are strongly related to the ignition timing control. Therefore, we checked how the products are modified in the change histories L, M, and N, and then found that the basic functionality of the ignition timing control was updated in the histories. Therefore, we defined factor 2 as the variability of "Ignition Timing" and the selected software components as variation points.

5.2.3 Result

The detected variability in the case study is shown in Table 4.

5.3 Evaluation

We confirmed the result of the analysis with senior engineers in the business division to evaluate the practicality of FAVE.

The detected variability is the actual variability of the product line that has occurred in the product release history that we analyzed. FAVE detects the variability that causes modifications for the software components. This means that we can focus on only the detected variability and omit the remaining possible variability at least for the existing products. Moreover, FAVE detects variability that the engineers have not expected. For example, they thought that modifications related to the fuel type (factor 3) had a very minor effect on the software. As FAVE detected, the modification caused a significant effect on some of the software components and its proportion ratio was 13%. Note that the sample data is very important and the census survey is desirable.

The detected variation points that relate to software components are also relevant. Even for the expert, defining an exact set of software components that are modified by a variability is very difficult for an engineer because of the complexity and non-linearity of the physical phenomenon of the engine system. FAVE detects software components that match the experience of the expert successfully. This means that FAVE has potential to extract tacit knowledge implemented in the existing products, and we could use that as explicit knowledge. Moreover, FAVE detects some software components that we did not expect as variation points. Some software components suggested by the experimental result looked irrelevant to the variability. However, we understood that the component should be included to the variation point, after we analyzed the specification of software components. This means that FAVE may extract variation points that we have not noticed yet.

Variability 3 (Fuel type) and 4 (Intake valve) contain software component No. 4 as one of their variation points. This means that the variability in the legacy software was not orthogonal to each other. To improve modularity of the variability, software component No. 4 should be refactored. For example, we can divide the software component into two components in which one is related to Fuel type and the other is related to the Intake valve. FAVE detects the variability itself and overlap of the variability that should be refactored in the future.

In the experimental application, people performed only defining the number of significant factors and defining the

Table 4. Variability of experimental application

Factor ID	1	2	3	4
Variability	Variable valve timing	Ignition control	Fuel type	Intake valve
Variation points (SWC ID)	3, 18, 24, 27, 28, 29, 30, 47, 48, 49	8, 13, 14, 33, 44	4, 10, 23, 32, 39, 40	4, 25, 26, 41
Proportion	0.17	0.16	0.13	0.13

variability. FAVE will reduce the overhead for introducing SPL approach.

6 Conclusion

In this paper, we proposed an approach to suggest variability candidates across existing software products by analyzing the change history. We applied a factor analysis technique to analyze variability of the existing products and call our approach "FAVE: Factor Analysis based Variability Extraction." We apply the factor analysis to changes between existing products and detect the co-change patterns that may indicate the variability of the product line.

In the case study of the automotive engine-control system, we detected four variabilities and their variation points from the change history. The detected variability corresponds to the variability from the requirement viewpoint.

Clearly, this work is in its initial stage. We are currently exploring several extensions to it.

We used a part of a software product as an example for the experimental application. We hope that FAVE can be used for a whole software product, and there are many technical issues. For example, we will need to visualize the results to be understandable for domain experts. In the future, we will analyze a larger data set and study appropriate solutions.

Variability analysis is an important step, but not the goal. We need to refactor the analyzed software components so that the components can be reused as core assets of the product line. The refactoring of the detected variability is the key challenge for migrating the software product line.

FAVE only detects the existing variability. To introduce variability from a roadmap of future products, we need to integrate our approach and requirement-oriented variability analysis.

The products release history contains information about variability that occurred in the past. The history may help us to understand its variability.

References

[1] AUTOSAR. AUTomotive Open System ARchitecture. *http://www.autosar.org/*. visited on Jan. 18, 2008.

[2] J. Bayer and et al. PuLSE: A methodology to develop software product line. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, pages 122–131, May 1999.

[3] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[4] M. Fischer and et al. Mining evolution data of a product family. In *Proceedings of 2nd International Workshop on Mining Software Repositories (MSR'05)*, pages 1–5, 2005.

[5] R. Foundation. The R project for statistical computing. *http://www.r-project.org/*. visited on Jan. 18, 2008.

[6] I. John. *Software Product Lines*, chapter Capturing Product Line Information from Legacy User Documentation, pages 127–160. Springer, 2006.

[7] K. C. Kang, M. Kim, J. Lee, and B. Kim. Feature-oriented re-engineering of legacy systems into product line assets - a case study. In *Proceedings of Software Product Lines: 9th International Conference (SPLC 2005)*, pages 45–56, 2005.

[8] C. Lai and D. Weiss. *Software Product Line Engineering*. Addison Wesley, 1999.

[9] F. Loesch and E. Ploedereder. Optimization of variability in software product lines. In *Proceedings of Software Product Line Conference 2007 (SPLC2007)*, pages 151–160, 2007.

[10] K. Pohl, G. Bockle, and F. V. D. Linden. *Software Product Line Engineering: Foundations, Principles And Techniques*. Springer-Verlag New York Inc, 2005.

[11] M. Steger and et al. Introducing pla at bosch gasoline systems : Experiences and practices. In *Proceedings of Software Product Lines: International Conference (SPLC 2004)*, pages 34–50, 2004.

[12] C. Tischer, A. Mueller, M. Letterer, and L. Geyer. Why does it take that long? Establishing product lines in the automotive domain. In *Proceedings of Software Product Line Conference 2007 (SPLC2007)*, pages 269–274, 2007.

[13] K. Yoshimura, D. Ganesan, and D. Muthig. Defining a strategy to introduce software product line using the existing embedded systems. In *Proceedings of 6th ACM & IEEE Conference on Embedded Software (EMSOFT06)*, 2006.

[14] K. Yoshimura, F. Narisawa, K. Hashimoto, and T. Kikuno. FAVE - Factor analysis based approach for detecting product line variability from change history. In *Proc. of 5th Workshop on Mining Software Repository (MSR2008)*, 2008. (to appear).

[15] T. Zimmermann and et al. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 429–445, 2004.