FAVE - Factor Analysis Based Approach for Detecting Product Line Variability from Change History

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

This paper describes a novel approach to detect variability in a software product line from its change history such that software changes are converted to vectors and a factor analysis is applied. To show the applicability of our approach, we conducted experimental applications using a software repository of automotive engine control software. As a result of the experiments, variability is detected from the change history of products.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Algorithms, Measurement, Design

Keywords

Software Product Lines, Variability, Factor Analysis, Change History

1. INTRODUCTION

Now, most control systems consist of embedded software because of its flexibility and cost efficiency. For example, an automotive control system has millions of lines of code for a high-class vehicle that includes hybrid power-train control, stability control, and GPS. Recently, developing control software has become increasingly difficult because the software size and number of variations are increasing dramatically. Software product line (SPL) engineering has been proposed as a development method for software that has variations. In the SPL engineering approach, software is developed for a specific product and for a product line that includes variations of the products. For reusing software across the product line, variability that are differences of requirement between product variations are analyzed. Then variants of the software component (e.g. a set of files) are implemented in order to satisfy specifications of the requirements. Benefits of SPL engineering have been extensively discussed in the literature and have been widely recognized by industry[1][2].

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 [3][4][5]. 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.

However, commonality and variability analysis of existing software is a block against migrating into SPL engineering. Related works proposed methods for analyzing commonality and variability from a requirement point of view and connecting that to the implementation [6][7]. On the other hand, some industrial case studies reported that commonality and variability analysis takes a long time[8][9]. Requirements and implementations of existing software are enormous. Moreover, such requirement analysis strongly depends on the expertise of the analysts and is time consuming. There is a need for analyzing the commonality and variability in an automatic way.

We have already developed a method for evaluating commonality between two current products in a quantitative way [10]. In the proposed method, the commonality between products was detected, but we were unable to detect the variability across the variations. The difference between products consists of variability and product-specific parts, and we could not separate them when analyzing only two products.

This paper proposes a novel method to detect variability across existing software by analyzing the change history. 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." 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."

We apply the factor analysis for changes between the existing products and detect the commonality of the changes. Of course, software product line is not a set of the existing products, but a

*Graduate School of Information Science and Technology, Osaka University

MSR'08, May 10-11, 2008, Leipzig, Germany.

set of variability of the product property, variation points in software architecture and variants of software components. From our viewpoint, it is difficult for detecting variability and mapping from variability to variation point that is a set of software components. Because an actual product has many internal and invisible variability, and the mappings from the requirements to the software components is not one-to-one, but complex. Before we develop the variants, we have to detect the variability and variation points.

Existing products contain their variability. If we could extract the variability from the existing products, it will be an useful information for migrating the software product line because most of the existing variability will be valid also in future product line. We focus on the change history of software components that compose the products. When the selections of variability were different between the products in the history, the implementation of the software components that relates the variability was also different. A major variability should occur several times and the relationship to the software components could be detected as a significant change pattern in the history. The factor analysis extracts the change pattern as commonality of changes and a set of software components that have changed with variability.

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. After that, the detected variability is refactored as a variation point and its variants.

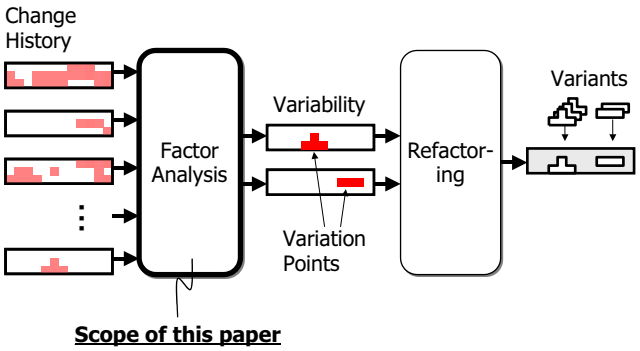

Figure 1: Overview of proposed method

One advantage of such a statistical approach is that we do not have to investigate specifications and source code files in detail for detecting variability that has occurred in the past. We do not analyze any variability from the requirement or implemental viewpoints, but just apply the factor analysis to the change histories. The statistical analysis does not depend on product domain and language but on its architecture and development style.

This paper is structured as follows: Section 2 provides an overview of the software product lines and variability. Section 3 describes the factor-analysis technique. Section 4 summarizes related work. Section 5 describes our FAVE method for detecting variability in existing products. The application of the proposed method to existing engine control software is the topic of Section 6. Section 7 describes our work in progress and a conclusion.

2. SOFTWARE PRODUCT LINES

Organizations often develop numerous products in a specific business area, or they may develop single, successful products that, over time, become new, separate products through enhancements and adaptations. Typically, these products are developed in separate, independent projects. The customer, the project, and organizational pressure make managing multiple development and maintenance threads very difficult.

In trying to manage their development and maintenance threads, organizations typically face other problems with their software development, such as high development and maintenance costs, schedule delays, and lower quality.

These are all problems that can be mitigated by taking a product-line approach to software development. A software product line is a set of similar software systems that are developed and maintained together. The basic idea that underlies product line engineering is to exploit the similarities of different systems and to reuse their common parts. A product line has been defined as "a family of products designed to take advantage of their common aspects and predicted variability" [5]. Another often-used definition was proposed by the Software Engineering Institute: "A software product line is a set of software intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [4].

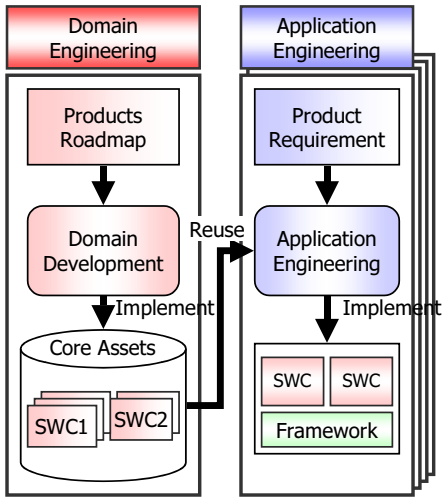

Figure 2: Overview of SPL

A generic SPL life cycle that is split into two phases: domain engineering and application engineering, is shown in Figure 2. After a scoping step, the products are defined so that the product line covers their characteristics. Then, domain engineering develops a product line infrastructure that contains core assets such as requirements, architecture, components, and tests. In application engineering, the core assets that have been created during domain engineering are used to build actual systems in the domain.

To achieve an SPL, architects focus on commonality and variability of the products in the product line. An example of an automotive control system, adaptive cruise control (ACC), is shown in Figure 3. ACC maintains a desired speed specified by the driver and adapts vehicle speed to changing traffic conditions by automatic acceleration and braking. ACC has a sensor that measures the distance of the preceding vehicle. Most of the basic functions of the ACC product line are the same in variations. However, there are many variability in the product line. For example, we can select a radar sensor, a laser sensor, and/or stereo camera for the distance sensor. A feature tree of the ACC that indicates commonality and variability in the product line is shown in Figure 4.

We can define variability as a common changing pattern across

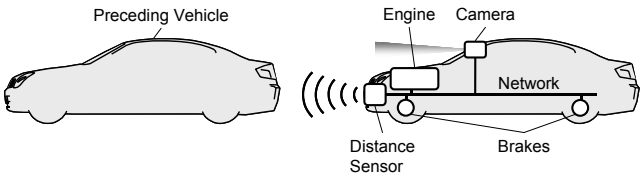

Figure 3: Overview of ACC system

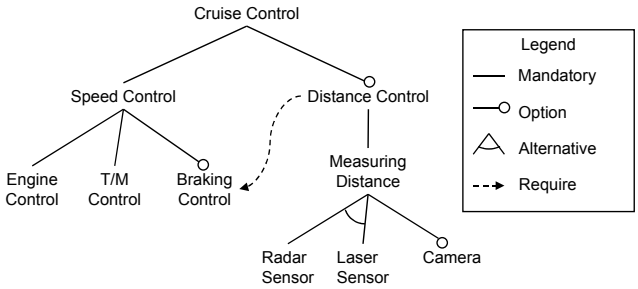

Figure 4: Feature tree of ACC

the product line. Architects design software architecture based on commonality and variability analysis. Then, parts of the software in the SPL are changed according to the variability so that the software satisfies the product requirement.

3. FACTOR ANALYSIS

An overview of the factor analysis is shown in Figure 5. Factor analysis is a multivariable analysis technique used to explain commonality among observed variables in terms of fewer unobserved variables called "factors". Mathmatically, eigenvector analysis is applied for the factor analysis and the factors correspond to eigenvectors.

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

Each factor correlates also with elements of the observed variables (i.e. vector). The degree of correlation between the element and the factor is called a factor score. With using the factor score, we can expect that the meaning of the elements is based on the meaning of the factor.

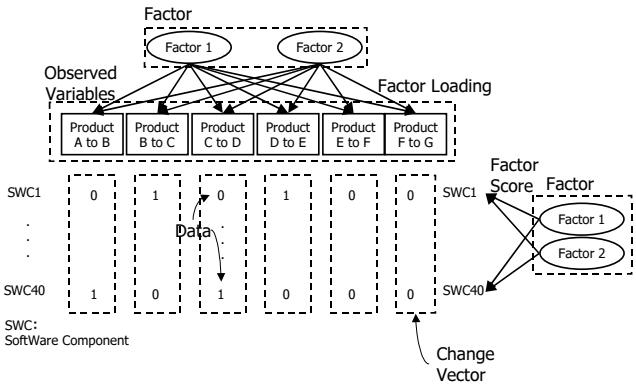

Figure 5: Overview of factor analysis

4. RELATED WORK

SPL engineering is an investment in future products, so various researchers have studied roadmap-oriented product line engineering [3][4][11]. There are also many studies about product line engineering for legacy software by industrial request. However, researchers have started mainly from current products to future products. Kang [6] analyzed the requiments of legacy system and modeled as a feature tree to migrate into SPL. John [7] analyzed documents of legacy system to detect variability. Steger [8] analyzed electric and electronic architecture of the controller as variation points of the system.

In contrast, our approach focuses on the change history of the product line from past to current and analyzes its change patterns automatically. This result of the extraction helps developers understand the variability of the product line.

In [12], Fischer proposed an approach to detect coupling between components of multiple product variants by mining their code repositories. Their approach detects the degree of coupling between components across the change history of the variants. In contrast to their work, FAVE detects the variability among the change history in orthogonal and the variation points as correlation to the software components.

Independently from SPL, Zimmermann [13] developed an approach to detect couplings of software components from 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 variation point in the concept of SPL. Moreover, FAVE detects the significant variability and variation points from the history while using the factor analysis.

5. FAVE - FACTOR ANALYSIS BASED VARIABILITY EXTRACTION

5.1 Overview

The purpose of FAVE is to detect product-line variability that has already occurred in existing products. We apply factor analysis to changes between existing products and detect the commonality of the changes, which signifies the variability of the product line. An overview of FAVE process is shown in Figure 6.

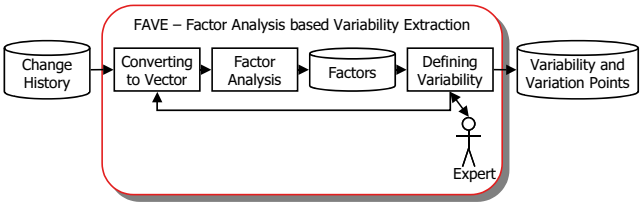

Figure 6: Overview of FAVE process

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.

In FAVE, we convert change histories so that we could analyze them numerically and consider the changes as an observed variable for the factor analysis. Then, we apply the factor analysis to the changed data and mine the variability of the products. After detecting the variability, we define the means of the variability from the viewpoint of requirements and specifications.

In the following subsections, we explain how FAVE, as shown in figure 6, is applied to existing products.

5.2 Converting Change Histories

Change histories of software components are not numerical data, so we cannot apply the factor analysis directly. For analyzing variability based on factor analysis, we convert change histories of software components into numerical data.

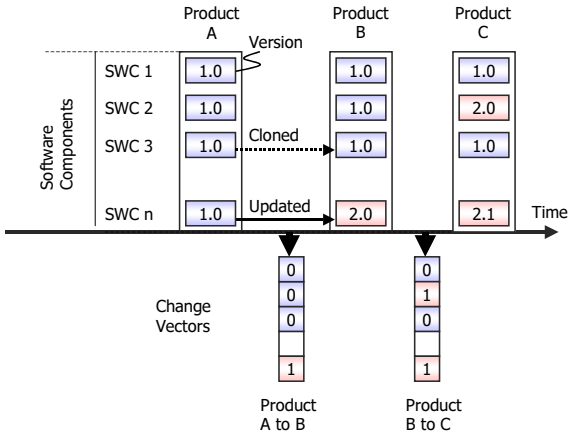

Figure 7: Converting Change Histories

Change histories are converted to numerical data that we call change vectors, as shown in Figure 7. Note that we assume that each product consists of software components, and a product is developed from similar products by updating some software components. The updating means changing the specification and source code files and so that the component satisfies product specific requirements.

We define the change vector as a set of binary data that indicates about changes in software components between products. For example, in Figure 7, component 3 is not updated from product A to B. Therefore, the corresponding element in the change vector of product A to B is "0". On the other hand, component n is updated from version 1.0 to 2.0, so the corresponding element is "1."

The change history is converted into change vectors, and the vectors are used as observed variables for the factor analysis in the FAVE process.

5.3 Factor Analysis

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

- **Factor**
  A factor is an unobserved variable that is a commonality of observed variables.

  In FAVE, a factor indicates a commonality of the changes between existing software. We can extract the factor as variability in the product line because commonality of the changes is a pattern of how the existing software changes across the history and meets the definition of the variability of a product line.

- Factor Loading
  Factor loading indicates how the factor affects observed variables.

  In FAVE, factor loading indicates which change history is correlated with the variability. When the factor loading is high, the variability is related to some specification that changes in the change history.

- Factor Score
  Factor score indicates how the factor correlates with the element of the observed variable.

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

5.4 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 8.

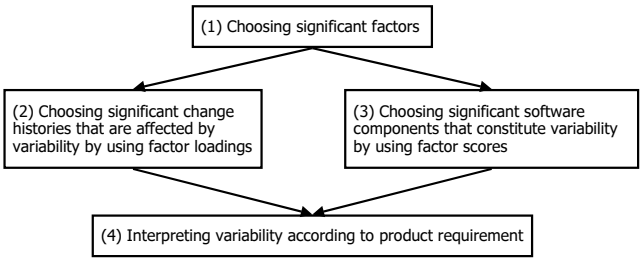

Figure 8: Process for defining variability

Firstly, factors are calculated from the change vectors and examined their significancy. The details of the calculation process depend on the analysis method of factors, e.g. the varimax method and promax method. When the varimax method is applied, the number of the significant factor is decided with using the chi-squared test and cumulating ratio of the factors.

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

Then, we select the set of software components that means the variation point. If the factor score is significantly high, the software component has been modified when the variability occurred in the change histories that has 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 the meaning of the variability with referring the combination of the software components functionality.

As the last step, the factor is interpreted as a variability of the SPL. For example, we guess the meanings of the factor as variability from the set of the software components that have high factor scores. Then we check whether the variability has occurred in the change histories that have 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.

6. EXPERIMENTAL APPLICATION

6.1 Overview

An overview of an engine control system is shown in Figure 9. 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 is a number of variations that correlate the mechanical structure, which satisfy the product specification, e.g. number of cylinders, transmission type, fuel type, and fuel injection, for example.

We apply FAVE to part of the engine control application layer. Generally, an engine control system consists of basic software layer and application software layer. As defined by an industrial standard for automotive control software [14], most of software components in basic software layer correspond the electronic architecture of control system, such as microprocessors, LSIs and network protocols. So the variability of basic software is mainly visible and the relation between the variability and variation point is relatively simple in this example. On the other hand, a feature of the application layer calculates phenomenon of engine, i.e. intake airflow, fuel combustion and exhausting gas. These phenomenon correlate many kinds of physical parts such as engine size, number of cylinders, valves, fuel injectors, fuel type and so on. Moreover there are some non-functional requirement that cause variability such as exhaust gas regulation by different countries and drivability. So the feature in the application layer has much visible and invisible variability, and the correlations between the functions and the variability are complex. We apply FAVE in order to extract the variability of a feature in the application layer and the mappings to software components of the feature.

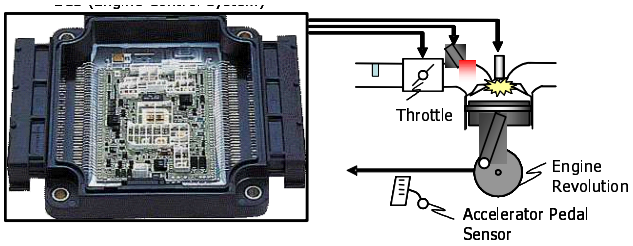

Figure 9: Engine control system

An overview of the experimental application is shown in Table 1. The repository has data about seven products. We extracted one control subsystem that consists of forty software components. Each software component calculates the state variables of the control system. To compute the factors, we apply R [15] and configured the parameter, as shown in Table 2.

Table 1: Experimental example

Application	Engine control software
# of products	7
# of software components	40

Table 2: Experimental environment

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

6.2 Analyzing Variability

We analyzed the variability by following the process in Figure 6.

6.2.1 Converting to Change Vector

First, we converted the change histories to the change vectors. We obtained six change vectors from seven change histories of existing products from the software repository.

First, we converted the change histories to the change vectors. We obtained six change vectors from seven major release of existing products from the software repository. The major release of software corresponds to the different vehicle in the market. We ignored the minor revision history of each major release so that the variability between products would be extracted. Due to a company confidential issue, we are unable to disclose the organization of the repository and the change vectors.

6.2.2 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. Unfortunately, p-values of the chi-squared test are too high to determine the number of the factors. Therefore, we defined the numbers as "2" using the cumulating ratio because the cumulating ratio is 40.5%, which is sufficient for representing 40 elements of the change history.

Table 3: Examination Result for # of factors

# of factors	1	2
X squared	9.45	4.24
Degree of freedom	9	4
p-value	0.397	0.374
Cumulating	0.225	0.405

- Factor loading

  The factor loading of the experimental application is shown in Figure 10.

  Change history No. 6 must correlate with factor 1 and histories No. 1 and 5 might correlate with factor 1. The experimental result suggests that variability relating to factor 1 has been occured in the change history No. 6.

  Change history No. 2 must correlate with factor 2. In other words, some changes that occurred in change history No. 2 define the meaning of factor 2. This means that variability relating to factor 2 has been occured in the change history No. 2.

- Factor score

  The score of factor 1 is shown in Figure 11. Only four in forty software components have significantly high factor score > 2.5, and the other components have low factor score. The experimental result suggests that software components No. 8, 10, 17, and 26 are correlated with each other. This means that there is a set of software components has been changed together in the change history and FAVE extracted the set automatically.

  The score of factor 2 is shown in Figure 12. Only three in forty software components have significantly high factor

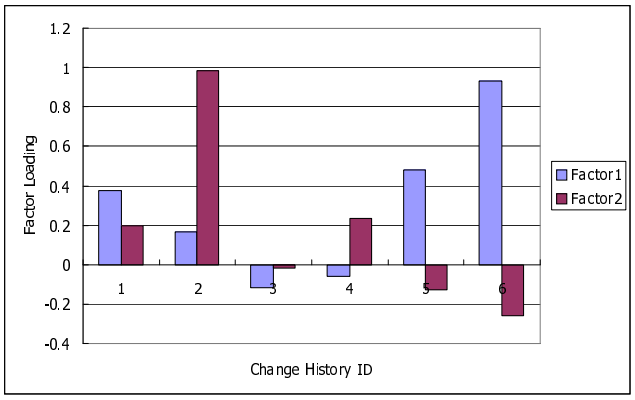

Figure 10: Factor loading

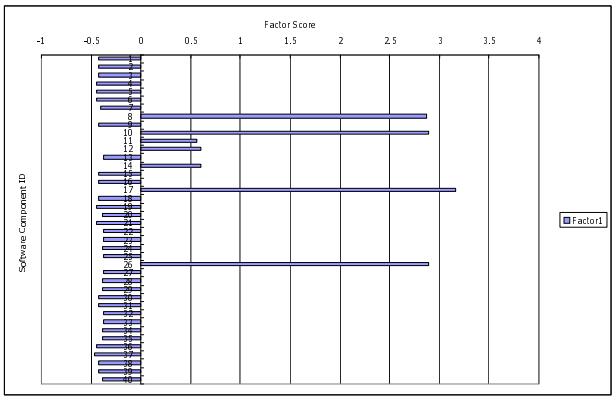

Figure 11: Factor score (Factor 1)

score > 3.0, and the other components have low factor score. The experimental result indicates that software components No. 11, 12, and 14 are correlated with each other, i.e. these software components were modified together in the change history. Again, this means that there are sets of software components have been changed together in the change history and FAVE extracted the set automatically.

Moreover, one can see that only 7 in 40 (14.3%) components account for 40.5% of the change history. This means that the frequencies of the software components modification are not equivalent and FAVE integrates the change pattern as the factor scores.

6.2.3 Defining variability

Finally, we defined the meaning of the factors as the variability.

In case of the factor 1, we selected the software components No. 8, 10, 17 and 26 as the variation point of the variability based on the factor score. Then we checked the functionalities of the components and suppose what kind of variability is related to the functionalities. From the control specification point of view, one of the components is strongly related to the throttle and 2 of them are related to the phenomenon of intake airflow. Correlation between other one and product specification was not clear. Therefore we checked how the products are modified in the change histories No. 5 and 6 then found that the throttle type was modified from the electric control type to the wired type. So we defined the factor 1 as the variability of "Throttle type" and the software components No. 8, 10, 17 and 26 as the variation points.

In the case of factor 2, we checked the requirements across software components No. 11, 12, and 14 and compared them with the changes that occurred in change history No. 2. The software components calculate the variables that relate the fuel amount of each cylinder. When change history No. 2, the product specification, was modified from "3 cylinders" to "4 cylinders," we defined factor 2 as the variability of "Number of Cylinders."

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

Table 4: Variability of experimental application

Factor ID	1	2
Variability	Throttle type	# of cylinders
Variation points (Software Component ID)	8, 10, 17, 26	11, 12, 14
Proportion	0.215	0.191

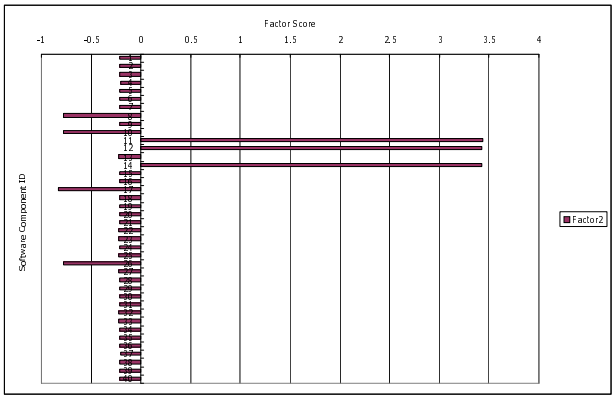

Figure 12: Factor score (Factor 2)

6.3 Evaluation

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

The detected variability is the actual variability of the product line and has occured in the history that we analyzed. The engine control system may have many variability other than the detected variability, FAVE detects the variability included in the history. This is a limit but also a benefit of FAVE. Although we can suppose hundreds of variability of engine control system, FAVE detects the variability that causes modifications for the software components. This means that we can focus into only the detected variability and ommit the rest possible variability at least for the existing products. Note that the sample data is very important and the census servey is desirable.

The detected variation points that relate to software components are also relevant. Even the expert, it is very difficult that an engineer defines an exact set of software components that are modified by a variability, because of the complexity and non-linearity of the engine system's physical phenomenon. Surprisingly, FAVE detects software components that matches the experience of the expert. This means that FAVE extracts tacit knowledge implemented in the existing products and we could use them as exiplicit 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.

In the experimental application, only the defining the number significant factors (6.2.2) and the defining variability (6.2.3) was done by human and the others were done automatically. The application of factor analysis to variability analysis was really helpful to reduce the overhead for introducing SPL because existing software repository is changing day by day. If we need one year for analyzing variability, there must be a gap between the result of analysis and the real software repository.

7. CONCLUSIONS

In this paper, we proposed an approach to detect variability across existing software by analyzing the change history. We applied a factor analysis technique to detect 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 commonality of the changes that indicate the variability of the product line.

In the case study of the automotive engine control system, we detected two clear variabilities from the change history. The detected variability corresponds to the variability from the requirement viewpoint. Although the change history appears random, FAVE detected the variability as the change pattern when the factor analysis was applied.

In this paper, we used a relatively small example for the experimental application. We believe that FAVE can be used for a large software history, and there are many technical issues. For example, we will need to reduce some data in order to calculate the factors. In the future, we will analyze a larger example 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 into future products, we need to integrate our approach and the requirement oriented approach based on product roadmap.

In any case, before making brute force requirement analysis into actual software, the change history must be analyzed carefully. The history contains information about variability that occurred in the past.

8. REFERENCES

[1] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action*, pages 111–274. Springer, 2007.

[2] Software Engineering Institute. Product line hall of fame. *http://www.sei.cmu.edu/productlines/*. visited on Feb. 18, 2008.

[3] 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.

[4] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] Klaus Pohl, Gunter Bockle, and Frank Van Der Linden. *Software Product Line Engineering: Foundations, Principles And Techniques*. Springer-Verlag New York Inc, 2005.

[12] Michael Fischer 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.

[13] Thomas Zimmermann 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.

[14] AUTOSAR. Automotive open system architecture. *http://www.autosar.org/*. visited on Jan. 18, 2008.

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