Experiences with Commonality Control Procedures to Develop Clinical Instrument System

[Industrial Experience Report]

Ryuichiro Kodama
Hitachi High-Technologies Corporation
882, Ichige, Hitachinaka-shi, Ibaraki-ken,
312-8504 Japan
kodama-ryuichiro@naka.hitachi-hitec.com

Jun Shimabukuro
Hitachi, Ltd.
Yoshida-cho 292, Totsuka-ku, Yokohama-shi, Kanagawa-ken,
244-0817 Japan
jun.shimabukuro.qw@hitachi.com

Yoshimitsu Takagi
Hitachi High-Technologies Corporation
24-14,Nishi-Shimbashi 1-chome, Minato-ku, Tokyo
105-8717,Japan
takagi-yoshimitsu@nst.hitachi-hitec.com

Shinobu Koizumi
Hitachi, Ltd.
18-13, Sotokanda 1-chome, Chiyoda-ku, Tokyo
101-8608 Japan
shinobu.koizumi.pd@hitachi.com

Shun'ichi Tano
Graduate School of Information Systems, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo
182-8585 Japan
tano@is.uec.ac.jp

ABSTRACT

This paper reports our experience with software development based on the Software Product Line (SPL) approach employed for Clinical Instrument Integration Management Software (CIIMS). CIIMS is the system software which systemizes heterogeneous clinical instruments. These instruments require their particular management so that various parts of CIIMS are forced to be changed. This makes it difficult to create development plans to connect new instruments to CIIMS. In this paper we summarize a new estimate method called the Architecture Domain Matrix (ADM) method which effectively solved this problem in our experience. In ADM each architectural element is further decomposed into clinical operation flow elements and core assets of software are extracted from these elements. This method estimates the CIIMS commonality with precision and finally enables to successfully connect new instruments. In addition this method provides a Work Breakdown Structure (WBS) and supports development team building. WBS is generated by collecting all the changes for each operational flow element. A development team suitable for change is organized by taking into consideration all the changes for each architecture element. We integrated three different instruments into CIIMS in 18 months after applying this method to a real project and achieved 2.5 times greater productivity with the embedded software than that with our previous non-SPL process.

Categories and Subject Descriptors

D.2.13 [Software Engineering]**: Reusable Software** – ***Domain engineering***; **D.2.11** [Software Engineering]**: Software Architectures** – ***Domain-specific architectures***; **D.2.9** [Software Engineering]**: Management** – ***Cost estimation***

General Terms

Measurement

Keywords

Software Product Lines, Domain Analysis, Architectures, Cost Estimation

SPLC '14, September 15 - 19 2014, Florence, Italy

http://dx.doi.org/10.1145/2648511.2648540

1. INTRODUCTION

Laboratory automation has contributed to improvements in the speed, costs, and quality of clinical inspection. A variety of automated instruments have been used in clinical laboratories to inspect medical specimens such as blood.

An example of advanced automation is the system shown in our previous work [1] to connect multiple automated instruments on conveyance roads. Clinical Instrument Integration Management Software (CIIMS) intensively controls instruments and conveyance roads in the system. The source codes of CIIMS are large and make it difficult to add new instruments. The source codes in vehicle control software and mobile phone software are more than millions of lines of code [2]. The scale of CIIMS is comparable to that of those software so that we have to deal with difficulties in developing CIIMS.

One of the solutions is to provide a common platform for CIIMS to connect instruments to reduce development costs. The software product line (SPL) [3] [4] is known as a methodology of development that can properly update and maintain such a common platform. SPL is expected to reduce the amount of application software contained in CIIMS by core assets and to improve productivity by managing the core assets.

A common platform is typically built by analyzing source codes contained in multiple products [5]. This analysis extracts commonality and variability from existing multiple software products. Although we can find reference source codes in legacy

CIIMS developed in the past, they are too old to be used for reference because it takes more than a year to develop one product in CIIMS. Therefore, we have to select products developed most recently as reference.

Refactoring techniques can also be used to build a common platform from existing software [6]. The techniques modify source codes to improve variability and maintainability without changing external specifications. However, it is necessary to control how much should be modified in refactoring because we have to avoid too much refactoring cost.

This paper introduces a commonality estimate technique as a best practice learned from our experience. In the technique core assets are extracted from source codes of a single existing product and refactoring cost is controlled. The technique also supports the planning of development processes where both core assets and multiple products are simultaneously developed after estimates are made.

We solved three problems to develop new multiple products based on the analysis of a single product.

Problem 1: How to control the costs of developing core assets in an estimate phase.

Problem 2: How to improve the precision of estimates in the whole cost of development.

Problem 3: How to reduce the difficulty in simultaneously developing both core assets and multiple products.

First, we solved Problem 1 by confining our variability scope to multiple products planned on the road map. It may be possible to expand the variability scope to virtual future products which contain new imaginary variability. However, we abandoned this expansion because cost control may become difficult through unnecessary variability expansion.

We introduced a matrix for Problem 2 that consisted of architectural elements and domain elements as an analysis mesh for refactoring and classifying source codes. The classified source codes limited the range of commonality of core assets by examining what should be core assets as not only functions (architectural elements) but also demand (domain elements).

We assigned core asset engineers to an architectural element unit to cope with Problem 3 by taking into consideration the difficulty of core asset development from the previously mentioned matrix. We also derived a Work Breakdown Structure (WBS) from the matrix as a design review tool that was used by all engineers including core asset engineers. WBS describes what changes in the functions led to demand.

We simultaneously developed a common platform and multiple products after planning core assets from a single product by analyzing the matrix in an estimate phase. This paper contains a section for lessons learned which we hope will contribute to the development of common platforms with limited reference source codes.

The remainder of this paper is structured as follows. Section 2 describes related work and Section 3 explains problems with developing CIIMS. Section 4 introduces a method of analyzing the matrix. We present our application of matrix analysis in Section 5, describe lessons learned in Section 6, and finally conclude the paper in Section 7.

2. RELATED WORK

The successful practices of SPL have been summarized in a practical framework [7]. Domain understanding is one of these frameworks, and Feature Oriented Domain Analysis (FODA) has been introduced as an associated technique [9]. This extracts a common set of features from products, models their combinations, and develops an individual product that logically combines features inside core assets. If too many features are extracted to be combined, the cost of validating the feature settings increases. Therefore, a technique of rationalizing logical combinations of features has been proposed [11]. However, it proposed to reduce the combinations of features and did not address the reduction of features themselves. Core assets that may be excessive are not controlled in advance because known approaches in past papers sought the maximum variability through features inside core assets.

In addition, Schmid [12] suggested a technique of choosing features in a feature design. This compared the risks and benefits caused by using features and determined whether a feature would be adopted or not. However, it did not enable the cost of developing application programs to be evaluated specific to a product. The general economy in SPL has been explained with a break-even point [13]. In other words, a cost in developing a core asset in SPL is regarded as a fixed cost that will be recovered by using it in as many products that can compensate for the cost. Thus, the balance in development costs between core assets and application programs is not evaluated or controlled.

Stoermer and O'Brien [14] suggested mining architectures for product lines to build a product line from existing source codes. They extracted information about the architecture and evaluated a product line. However, domain knowledge was not included in their evaluation.

Knodel et al. [15] built core assets with reverse-engineering information acquired from specification sheets, instruction manuals, or experts. Although architectural elements were examined, further grain size was not analyzed. Koziolek et al. [16] reported examining re-availability from both the architecture and the domain. They described a highly abstract architecture drawn from interviews with architects and used the architecture for rebuilding. Neither research group utilized both the architecture and the domain to analyze source codes.

Jones and Bergey [17] nominated the generation of WBS for a future problem in core asset development. The difficulties in SPL projects have not been overcome through WBS, which was generated in the estimate phase of core assets development in our study.

There have been studies that have taken into consideration either architectural or domain knowledge, but they have not used all the knowledge to extract core assets from a single product. Our study utilized two kinds of knowledge to analyze the source codes of a single product and extract core assets. A development plan could be derived even from a single product by exploiting these two kinds of knowledge.

3. AUTOMATED INSTRUMENTS IN CLINICAL LABORATORIES

3.1 CIIMS

Many clinical laboratories adopt an automation system that connects multiple instruments to conveyance roads and automatically analyzes one specimen such as blood with these multiple instruments. The test tubes containing specimens are conveyed to the instrument automatically via an appropriate conveyance path. A clinical test item is acquired through a bar code at the conveyed point and an analysis process is carried out with the instruments. After all the test results output by multiple instruments are confirmed for the specimen, and the validity of the data is confirmed by a laboratory technician, a report to a doctor is printed for the specimens. CIIMS plays a role in supporting laboratories in practice and automating combinations of conveyance roads and instruments.

CIIMS is remodeled whenever we need to connect a new instrument. Whenever a new instrument needs to be connected to CIIMS, this remodeling occurs, and CIIMS evolves each time. We also have to connect the evolved CIIMS to instruments that were connected in the past for upward compatibility. A platform that can survive evolution is therefore required.

3.2 Remodeling of CIIMS

The architecture of CIIMS is composed of (1) a user interface (UI) such as a screen and printing forms, (2) a database (DB), (3) a specimen handler (SH), (4) instrument communication (ICOM), (5) instrument control (ICNT), and (6) external communication (ECOM). The SH (3) is a component that tracks the specimen in the system until all the test results are reported after the inspection items of the specimen are ordered. ECOM (6) is a component that communicates with a host computer that manages the accounts of medical businesses.

It will be ideal to connect a new instrument to CIIMS by changing only a part or none of architecture. However, we were practically forced to change all the architecture from (1) to (6) to connect it. SH and ICOM need to be changed to communicate with a new instrument. In addition, it is necessary to remodel UI and DB to monitor the state of the new instrument mechanics and to manage the new reagent. We are also forced to change ECOM for the host computer to manipulate new instrument functions.

Our main goal was not only to establish measures for estimates but also to control the entire development cost in environments where such change chains occur.

4. ADM METHOD

4.1 Application of SPL

SPL engineering is defined by three activities of domain engineering (or core asset development), application engineering (or product development), and management [7].

Figure 1 outlines the flow of development based on these activities. Domain engineering outputs core assets according to the business plan for future new products, which should be maintained in a specific domain. Products that can maximize customer satisfaction while effectively applying core assets are developed in application engineering. This paper suggests an approach to supporting both domain and application engineering by an intensive estimate technique as shown in Figure 1.

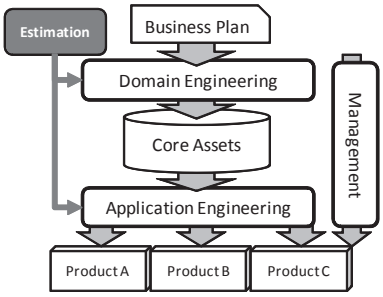

Figure 1 Development processes in SPL.

There are two approaches to develop core assets, i.e., proactive and reactive [8]. While core assets are developed in advance before products are developed in the proactive approach, enhancement and application of core assets run in parallel in the reactive approach. We adopted a reactive approach because there were very few reference products for core assets in our cases. It takes more than a year to develop CIIMS to adapt a new instrument due to compliance with reliable regulated development processes. Source codes other than the most recent CIIMS are too old to serve as a reference. Therefore, we decided to analyze core assets in an estimate phase by referring to the most recent CIIMS and the new instrument specifications written in a business plan.

4.2 Model of Idealized Core Assets

When N products are developed, the main purpose of SPL is to provide dominant development techniques in terms of cost, quality, and speed that are better than when the N products are individually developed. Core assets play a role in reducing the burden of developing these N products. Then, what kinds of factors influence idealized core assets?

Let us assume that we will independently develop three products of A, B, and C without core assets. Because these three development projects are independent in this case, each project has to redundantly create a tool, a design, and the source codes that should be shared. We call the cost to develop a product individually in this way an individual development cost. However, let us assume that we can form core assets that can be used in the three products. Then, we can decrease duplication by using core assets. If we increase core assets, we can finally acquire one large accumulation of core assets in an extreme case. The source codes of the three products become common, and the three products may emerge by managing the configuration of feature parameters [9]. If there are large differences in the specifications of the three products, the parameters referred to in each architectural layer increase and their logical combination becomes too costly to be managed. We call the cost of developing core assets costs in this way a core asset cost.

We previously discussed two extreme cases. The actual development will take an intermediate form that falls between them. The relationship between an individual development cost and a core asset cost is outlined in Figure 2. While the commonality ratio, which is the ratio of common source codes to three product source codes, varies from zero to one, the individual

development cost decreases; however, the core asset cost increases. The total development cost will be the sum of a core asset cost and an individual development cost and will be minimized somewhere with a commonality ratio between zero and one. That is our idealized scale of core assets.

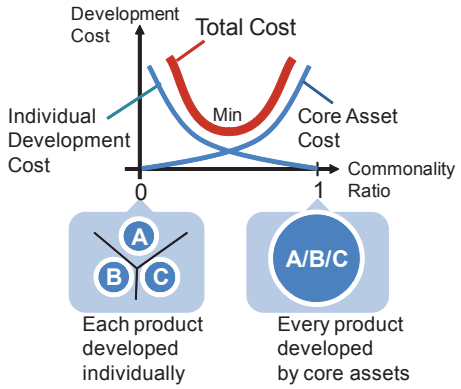

Figure 2 Relationship between individual development cost and core asset cost.

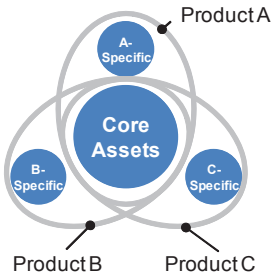

Figure 3 Relationship between core assets and products.

The idealized relationship between resources of products A, B, and C is depicted in Figure 3. The core assets to minimize the total development cost are combined with the application programs of products A, B, and C. A technique to control such an ideal development mix is required in an estimate phase.

4.3 ADM Method

This paper presents the architecture domain matrix (ADM) method as a technique of extracting core assets. ADM is a matrix consisting of two elements, i.e., domain knowledge elements (simply called domain elements in the following) and architectural components (Figure 4).

The general way to generate core assets will be to analyze each architectural element and extract common source codes from it. The ADM method enables to further decompose each architectural element into domain elements so that CIIMS commonality can be estimated with precision even from a single product.

The source codes in the ADM method are assigned to a cell of ADM and analyzed in each matrix cell to determine whether they can be used as core assets. Then, the cost to make the change is estimated. The range of variability is defined because a pair of a domain element and an architectural element is regarded as the relationship between requirements and achieved functions.

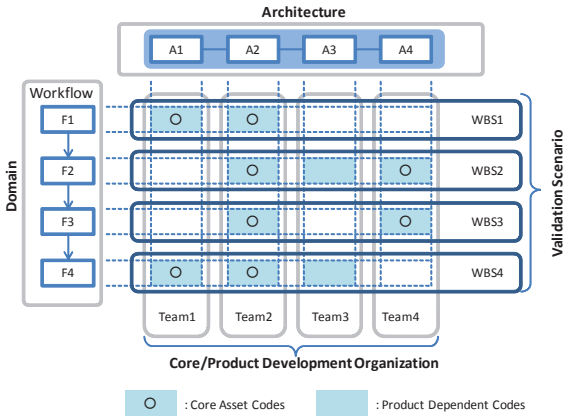

Figure 4 Architecture Domain Matrix.

The source codes of the existing product (called a base product) are analyzed through the ADM method together with the specifications of planned multiple new products. If the source codes assigned to the matrix cell should be shared by multiple products, they will fully or partially be candidates of core assets. After the cost estimates of all cells have finished, the development cost estimated in each matrix cell is summed up. If the total development cost does not fit the plan, the number of core assets to be developed is adjusted. Reduction in core assets may be chosen when multiple specifications are too different to be implemented in core assets. Increase in core assets may be chosen in reverse when the difference in specifications should be implemented with less cost by more abstraction. The design of core assets in either case can be controlled by gaining a broad perspective in the ADM. These assessments are conducted through cooperative consultation by domain experts and architects.

The grain size of architectural elements in the ADM is a range to be developed by one leader and engineers. Because all changes to an ADM architectural element can be seen when a vertical group of cells is viewed in Figure 4, the necessary skills for that element can be recognized and appropriate engineers can be assigned to the vertical group of cells. If an architectural element includes core assets, engineers of core assets are assigned to that element.

The grain size of domain elements in ADM is a range to be validated in a system test. A series of changes to an architectural element can be identified as being necessary to achieve a domain element when a horizontal group of cells is viewed in Figure 4. WBS is created by collecting changes in each domain element that is achieved. WBS supports design reviews during development.

Thus, the support to allocation of engineers and promotion of design reviews can reduce the difficulty of developing programs including those in the development of core assets.

We used clinical laboratory workflow elements as domain knowledge in this study. The workflow, for example, includes the preparation of instruments before inspections are practiced, practiced inspections, and maintenance of instruments. The laboratory workflow was also handled as domain knowledge about medical devices in a report [10]. Because clinical

instruments should be devices to automate strict medical practices, the workflow will be stable domain knowledge for analysis of source codes.

In addition, the workflow was comprised of an independent process of duties. The architectural element was also assumed to build an independent layer structure surrounded by clear input and output. If the architecture had the structure of independent layers, we could secure independence in ADM cells, and ADM would become a stable tool to analyze source codes. We assumed the cells of ADM were independent and the ADM framework would continue to be used in the long term.

4.4 Procedures in ADM Method

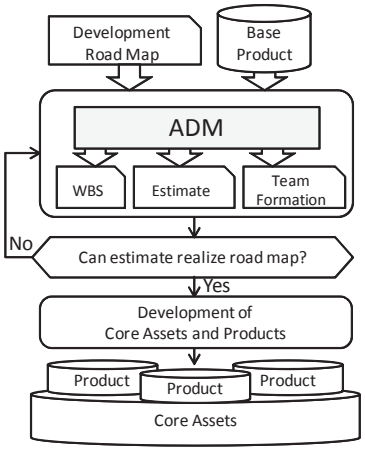

Figure 5 Development workflow using ADM method.

The development workflow for the ADM method is explained in Figure 5. The input for the method is the legacy CIIMS (the base product described in the figure) and the development road map. The map indicates the shipping plan for the multiple, say, N instrument models to be connected to CIIMS in the future.

The source codes of the base product are assigned to the ADM cells and are examined to see how much of the code can be core assets to be shared among the N models. The development cost of core assets and application program can be estimated depending on each model.

The number of core assets to be developed is adjusted so that the total development cost satisfies the shipping plan. Harmony between individual development costs and core asset costs is planned, as seen in Figure 2. The remodeling cost, $C_{i,j}$, for the source codes of the domain element, i, and the architectural element, j, is calculated as:

$$C_{i,j} = \min\left(CC_{i,j}, CD_{i,j}, CA_{i,j}\right),$$

where $CC_{i,j}$, $CD_{i,j}$, and $CA_{i,j}$ are the development costs for core assets, separation of core assets and application programs, and pure application programs that are dependent on each model. The cost is estimated through various factors. It depends on how deeply source codes can be analyzed, how well-structured the source codes are, and how valuable information can be acquired by domain experts and architects. Finally, the total development cost is calculated by summing all $CC_{i,j}$. Because ADM cells are assumed to be independent as previously described, tuning each $CC_{i,j}$ leads to the total development cost being adjusted.

The five practical steps in the procedures for the ADM method are below.

Step 1. Set up the development program

The first step is to choose what we call a base product that is a proven product used by customers. In addition, the N instrument models planned in a development road map are chosen in this step. Further, similarities between the base product and N models should be taken into account.

Step 2. Analyze the base product

An ADM table equivalent to that in Figure 4 should be created for the base product. First, the source codes of the base product are classified and assigned to each ADM cell. Next, the source codes for the base product assigned to the cells are analyzed to classify them into three categories:

c: Codes to be commonly used as core assets

d: Codes to be separated into core assets and application programs

a: Codes to be independently developed as application programs that are dependent on each model

These are encoded with **c** (core), **d** (dual), and **a** (application), which are written in the ADM cells.

Step 3. Estimate the N models

An ADM table equivalent to that in Figure 4 for N models should be created. The contents in the table of the N models are the same as those of the base product, but a dash (') is attached to the three codes **c**, **d**, and **a** in the cell where remodeling is needed. The meanings of these dashed codes are as follows.

c': Modify core assets to fit application programs.

d': Modify the interface inside the source codes.

a': Modify application programs to fit core assets.

The table of the base product may acquire a dash, which means that a change occurred when the core assets in the base product absorbed N model instruments. After coding in the ADM table is completed the total man-hours spent in developing the core assets and N models are estimated. Codes other than **c** mean remodeling work. If there are too few core assets of **c**, the effect of reducing costs through commonality is slight, and the total man-hours for development increase. When there are too many **c'**, **d'**, and **a'**, on the other hand, the number of man-hours will increase due to the remodeling workload in **c'**, **d'**, and **a'**. We return to Step 2 and plan core assets again as long as the total development cost can be reduced by increasing **c**, or decreasing **c'**, **d'**, and **a'**. If the increase and decrease in core assets does not contribute to reducing the total man-hours spent in development, the total man-hours in development are accepted as final estimates.

Step 4. Form development teams

Development teams are formed to correspond to each architectural element. Development man-hours are calculated for

each team by vertically viewing the cells in the ADM table. Talented engineers are allocated to the teams depending on the number of changes and degree of difficulty. If the cells contain core assets, core asset engineers are added to the team.

Step 5. Design WBS

When cells in the ADM table are viewed horizontally, what changes in architectural elements have affected the implementation of the workflow can be inspected. These are summarized in WBS, where the practice of the workflow and its related changes in architectural elements are documented. The role of WBS is to guide design reviews by teams involved in implementing workflow practices.

5. APPLICATION OF ADM METHOD

We applied the ADM method to a real development project according to the steps described in Section 4. We selected the most recently developed CIIMS as a base product.

5.1 Step 1: Set up the development program

Table 1 Schedule for case project.

	First Year									Second Year							
	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9
Base Product																		
Connection A																		
Connection B																		
Connection C																		

The case project was required to connect three new instrument models (A, B, and C) to CIIMS in 18 months (1.5 years). The development schedule on a monthly basis is summarized in Table 1. The colored rectangles in Table 1 mean terms in the development process. The base product was expected to be completed by March in the first year.

The development process for instruments A and B began with a design for each and ended after quality assurance (QA) was completed after being implemented and tested. Instrument C was the same as A and B except that the process did not include the QA process because the purpose of Connection C was to internally test the performance of instrument C.

In general after the QA process is finished, we continue exhaustive tests to validate the system, which is also carried out in an actual clinical environment. Such validation tests are out of the scope of this paper because we focus on the embedded software.

Before Connections A, B, and C were developed together with core assets, we applied the ADM method to control the design of core assets in the estimate phase.

5.2 Steps 2 & 3: Analysis and Estimates

Table 2 is an ADM table merged with the tables for the base product P and the three instrument models A, B, and C in Step 1. We classified the workflow elements vertically and the architectural elements horizontally. The divisions of P, A, B, and C are written in the last line of the table. We classified the source codes in the table as explained in Step 3 of Section 4.

We reconsidered two cases that led us to return from Step 3 to Step 2. The first was where we reduced core assets in the architectural element called ICOM because the communication protocols were too different to be covered by core assets. The second was where we increased core assets in the architectural element called SH because we expected costs would be reduced by enhancing the abstraction level of programs and absorbing the

Table 2 ADM table for P, A, B, and C products.

	Domain Category		Architecture																							
	Major	Minor	UI				DB				SH				ICOM				ICNT				ECOM			
1	Start Up (SU)		c'	c	c	c	d	d	d	d'	c'	c	c	c	d	d	d	d'	a	a	a	a				
2	General Mngt (GM)	GM1	d	d	d	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
3		GM2	d'	d'	d'	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
4		GM3	d'				d	d	d	d	a'	a	a	a	d	d	d	d	a	a	a	a				
5		GM4	d	d	d	d																				
6	Reagent Prep (RP)	RP1	d	d'	d'	d'	d	d	d'	d'	a	a	a'	a	d	d	d	d	a	a	a	a				
7		RP2	d	d	d	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
8		RP3	d	d	d	d'	d'	d	d	d'	a'	a	a	a'	d	d	d	d'	a'	a	a	a'				
9		RP4	d	d	d	d	d	d	d'	d	a	a	a'	a	d	d	d	d	a	a	a	a				
10	Measurement (M)	M1					d	d'	d'	d'	c	c'	c'	c'	d	d'	d'	d'	a	a'	a'	a'				
11		M2					d	d'	d'	d'	c	c'	c'	c'	d	d'	d'	d'	a	a'	a'	a'				
12		M3					d'	d'	d'	d'	c'	c'	c'	c'	d	d'	d'	d'	a'	a'	a'	a'				
13		M4	d	d'	d'	d'	d	d	d'	d'																
14	Calibration (C)	C1	d	d	d	d'	d	d	d	d'	c	c	c	c'									c	c	c	c
15		C2	d'	d	d'	d'	d'	d	d'	d	c'	c	c'	c'									c	c	c	c
16	Quality Control (QC)	QC1	d	d	d	d'	d	d	d	d'	c	c	c	c'									c	c	c	c
17		QC2	d	d	d'	d					c	c	c'	c'									c	c	c'	c'
18		QC3	d	d	d	d																	c	c	c	c
19	Inspection (I)	I1	d'	d	d	d	d'	d	d	d													c	c	c	c
20		I2	d	d	d	d	d	d	d	d													c	c	c	c
21		I3	d'	d	d	d'	d	d	d	d'	c	c	c'	c'									c	c	c	c
22		I4	d'	d'	d	d'	d	d	d'	d	c	c	c'	c									c	c	c	c
23		I5	d	d	d	d'					c	c	c	c	d	d	d	d	a	a	a	a'				
24		I6									c	c	c	c	d	d	d	d	a	a	a	a				
25	SHUTDOWN (SD)		c	c	c	c	c	c	c	c	c	c	c	c	d	d	d	d	a	a	a	a				
		Instruments	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C

UI: User Interface, DB: Database, SH: Specimen Handler, ICOM: Instrument Communication, ICNT: Instrument Control
ECOM: External Communication

differences between instruments.

We finally determined there were no changes in six out of 25 domain elements (e.g., line numbers 2 and 5 of the domain elements) and there were changes in the rest of the 19 domain elements. We were able to control the remodeling costs with the ADM method having been given the specifications for the three instrument models in the business plan.

5.3 Step 3: Final estimates

The final estimates are listed in Table 3.

Table 3 Development cost estimates.

Base Product	Case Project				
	Core Assets	Connection A	Connection B	Connection C	Total
100	11.3	18.9	34.0	52.8	117.0

The table summarizes by relative scale the costs in man-hours required for each development of core assets, i.e., the instruments for connections A, B, and C. These values are relative to the man-hour costs that were originally required to totally develop the base product where this was assumed to be 100. It was a significant result that we were able to estimate the three products to be developed with 117% effort for one base product. The cost for Connection C was relatively higher than that for the others because a great deal of remodeling was required inside instrument C for evaluation purposes.

Project risk was greatly lowered because the subdivided source codes were analyzed in the estimate phase.

5.4 Step 4: Form development teams

The results for designing the formation of teams are summarized in Table 4. The development teams were formed for each architectural element. We assigned core asset engineers to teams where checks were indicated in the second row of Table 4. If checks are indicated in the last row of Table 4, this means that teams should be divided into sub-teams by each instrument.

Table 4 Summary of design of development teams.

	Architecture Elements					
	UI	DB	SH	ICOM	ICNT	ECOM
Core Assets Engineers	✓	✓	✓	✓		✓
Instrument Engineers	✓		✓		✓	

We only located core asset engineers in the teams for the architectural elements of DB, ICOM, and ECOM. This is because these architectural elements required relatively little instrument based development as can be seen from the ADM table. Because the architectural elements of ICNT did not contain any core assets for total development, the team was formed based on instruments.

We could harmonize engineering skills with a degree of technical difficulty by arranging organization placements into a matrix in this way.

5.5 Step 5: Design WBS

Figure 6 has a simplified example of a WBS specification sheet that we created during this project.

No.	Requirement	Description
	Maintenance tool for instrument C	To perform maintenance program for instrument C

Project			
	A	B	C
			✓

Architecture Element	Minor Category	Remodeling	Cost
ECOM			
UI	System	To add a screen to set up parameters	
	Overview	To add a screen to monitor progress	
DB		To register parameters	
SH	Control	To plan test tubes to run for maintenance	
	Status	To monitor state of tubes	
ICOM		To execute maintenance program	
ICNT	Control	To control mechanics	

Figure 6 WBS specification sheet initiated by ADM.

The sheet was filled with information about workflow elements and related changes to individual architectural elements. The workflow elements in this example involved practicing the use of maintenance tools, which were implemented due to changes to UI, DB, SH, ICOM, and ICNT. The engineer in charge of architectural elements wrote specifications by referring to the WBS sheet and conducted design reviews together with engineers for the other architectural elements written on the WBS sheet. After all the related architectural elements had been implemented, we tested the maintenance tool in practice.

5.6 Results

We started with a project to develop core assets and CIIMS connections with instruments A, B, and C after specifying a development plan using the ADM method described in Subsection 5.5. The outcomes for the project are summarized in Table 5.

The start of this project was actually delayed for two months due to the delayed development of the base product. However, we were able to complete the project for A three months ahead of schedule and two months ahead of schedule for B. Although the project for C was delayed for two months because additional functions were required in instrument C, as a whole, we completed the development of the three instrument models in eighteen months (1.5 years).

Table 5 Actual progress in case project.

	First Year									Second Year									
	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11
Base Product																				
Connection A																				
Connection B																				
Connection C																				

Our experience in the past revealed that it could take 2.5 years to complete the development of two instrument models where SPL was not employed. The average development scale of the two past models was approximately equal to that of instruments A, B, and C discussed in this paper. Therefore, as the number of development models per year was improved with 2.0 (models per year) from 0.8 (models per year), we achieved 2.5 times greater productivity. These results demonstrate that core assets were

effectively controlled by the ADM method so that the period for development went according to schedule.

There are two reasons behind the achievement of our goal: high estimate precision and effective commonality extraction.

The estimated effort to complete the three product project was 1.17 times greater than that of the base product project as shown in Table 3. Because the base product was developed with our non-SPL process for one instrument, the actual effort was 2.5(years for 2 instruments)/2(instruments) = 1.25 year for all the engineers in our engineering section. Therefore, the estimated effort for the three product project was 1.25×1.17=1.46 year for all the engineers. On the other hand, the actual effort was 1.5 year for all the engineers. Given no difference in the number of engineers between two projects, the estimate error rate was 2.7%, which indicates the estimate was practical.

Table 6 Ratios (%) of core assets in UI, DB, and SH.

	Connection A	Connection B	Connection C
Estimated Ratio	95.4	93.7	76.0
Actual Ratio	96.5	98.1	60.8

Table 6 summarizes the ratios of core assets, expressed as a percentage, in the total line of codes for the architectural elements of UI, DB, and SH. The estimated and actual ratios are compared in the table. The lines of codes other than those for core assets became the source codes dedicated to the instruments. There are few differences in the results between estimates and actual ratios in Connections A and B. The estimates were very precise as the entire development was undertaken without fluctuations. The ADM method definitely enhanced precision.

However, the actual rate for core assets fell for Connection C. This was due to requirements that were added after development started. Nevertheless, the rate of core assets of more than 60% indicated that core assets were available.

To wrap up the estimate error rate and the commonality extraction rate described in the above, these prove that it was advantageous to control costs in the estimate phase with the ADM method.

The whole and parts of the architecture could easily be understood through WBS generated by the ADM table because the final goals for individual changes were indicated on the WBS sheet. We could conclude that development was efficiently enhanced with the help of WBS introduced by using the ADM method.

The core assets extracted here were utilized for five new instrument models and two new conveyance road models in the next four years. This indicates how effectively and practically the core assets were extracted with the ADM method.

6. LESSONS LEARNED

We learned many invaluable lessons in over a year of experience in a real project. Although this was our first SPL experience, we are satisfied with the results in which higher productivity was attained. In what follows, we would like to share the lessons we learned with those who may have suffered from productivity difficulties in long-term projects where few references to the development of core assets have been available.

Organizational support to start with SPL

A great variety of integration products are being developed by the entire Hitachi Group. The production engineering (PE) teams at Hitachi, Ltd. are dedicated to strengthening the productivity of their products. SPL was one of their methodologies that they recommended to enhance the productivity of embedded software engineering. Our engineering team had joined the embedded software reform program driven by the Hitachi PE team before we started the project described in this paper. The program shared the effective methodologies used by Hitachi group companies in the meeting once every two months. After our project was chosen to be a jointly promoted case project in the program, we received support through the monthly progress management meeting where the project technical challenges are addressed by our engineers and the specialists from the PE team and the R&D headquarters at Hitachi, Ltd. In addition, two engineers from the PE team participated in our project. They worked to promote SPL in the project as they were members of our project. The project members could continue to insist on the importance of core assets throughout the long term project despite being inexperienced with SPL. We could not achieve our goal without organizational support led by Hitachi, Ltd.

Balance core assets and application programs

We adopted a reactive approach in SPL because we have few references to source codes to extract core assets as discussed in Subsection 4.1. In a proactive approach the development cost of core assets will be covered every time they are frequently applied to subsequent new systems. The more applications of core assets will lead to the more values customers will enjoy. Adopting a reactive approach, we were urged to create values of core assets in one long term project.

This caused us to be careful not to generate excessive core assets. The risk of our project was to postpone the completion of all the planned instrument connections due to too aggressive versatility of core assets beyond our technical skills. The commonality for core assets is extracted comparing the differences in specifications of new planned products. If the difference is too big to be implemented in one program with features, it may take longer time than planned. For example, we reduced core assets in an architectural component ICOM (Instrument Communication). A communication program can be typically standardized to absorb the specification difference by parameters and software libraries. Nevertheless, we made a decision to build an application program for each instrument communication because we found a huge gap among the instruments about how to handle communication state management.

Not only did we reduce core assets, but we also increased them. Although it was an aggressive plan to build core assets in the SH (Specimen Handler) component, we made a decision to increase them by enhancing the abstraction level of programs. And this abstraction was a must because how to handle samples is the core of CIIMS. Thus, a perspective view by the ADM method gives us a chance to consider the cost comparison between core assets and application programs.

A relationship between core assets and application programs was observed as we proceed with the estimate by ADM. First, those two types of programs are mixed inside even one architectural component depending on domain elements. Second, some of the source codes assigned to ADM cells have to be changed as denoted by dash (') regardless of core assets or

application programs. Three categories (c, d, and a) were assigned to every ADM cell when the base product was analyzed. A dash was attached to the three codes when three new products were analyzed. This means that core assets and application programs coexist in a coordinated manner. This does not say "we make core assets and you use them," nor "we use core assets and you make them." Both core assets and application programs can be a candidate for remodeling. It depends on the total development cost. We see the ADM table as an environment to adjust such costs.

The ADM table as core assets

We confined our variability scope to the three products planned on the road map. This decision was made to clarify the width of our specification control. Some readers may think that the core assets applied to only the three products are not a core program to generate as many products as possible. However, we think that our core assets include not only the program but also the ADM table. The table shows the difference between a legacy product and future products. In that sense, we can use the table as a change management tool to cope with another instrument connection. It gives us a possible change design of source codes in considerable detail as if it encapsulated the program. In fact the core assets described in this paper were connected to five the next five new instruments and two new conveyance roads in the next four years. This indicates how durable the core assets are.

Preparation for Estimation

The ADM method is an estimate technique to break down source codes into an architecture and domain mesh. This greatly enhanced the estimate precision as a result. Before starting this project, we prepared team building concentrating on capability to understand domain knowledge. A researcher from Hitachi, Ltd. joined our development projects two years ahead of our project to analyze domain knowledge because this knowledge in clinical instruments is greatly specialized. His domain knowledge was systematized in two years in relationship with source codes and shared with sub-leaders of each architecture component. That helped us analyze the source codes more precisely by both aspects: architecture and domain. That led to our commitment to a decision to rely on the estimate.

Possible contribution to regulated industry segments

The main reasons the goal of development was achieved were considered to result from a chain reaction where the quantity of remodeling was controlled, the precision of estimates was improved, and the recycling of source codes was increased. The matrix of architectural elements and domain elements contributed to the reaction. However, the matrix will not work for all types of projects. Our experience demonstrated that it will help to enhance productivity for large scale and long term projects where the domain scope is considerably specialized but rather stable. We think that the application of the ADM method can be expanded to regulated industry segments because the development processes in these segments tend to be rigid and prolonged and they have to comply with regulations. Regulations will specify due practices in segments and it may be possible for the domain scope viewed by users to become stable.

For example, laboratories related to measurement are generally required to prepare standard operating procedures (SOPs) as a document which all the operators should follow. SOPs include the procedures of data calibration and quality control. No matter how specialized equipments laboratories install, they have to control data quality through SOPs. Therefore, the specification of laboratory equipments tends to fit general SOPs. Data automation features of the equipments are encouraged to implement rather stable SOPs. Thus regulations related to SOPs can make the domain scope stable.

Core asset team embedded in product teams

The talented engineers who played an active part in the development of the base product became candidates to become team members in the development of core assets. It should be standard to maintain the independence of application-program and core-asset engineers because core assets should not be derived from the interests of some, and not all. Talented engineers, on the other hand, should intelligently play important roles in the development of new products. We settled on a compromised structure by considering the limited intelligence to be shared in total development. We decided that the core asset team would be embedded as a horizontal organization in product development teams as discussed by Takebe et al. [18]. Core assets and products are developed in parallel under managed communications between two types of teams through that team structure.

Deploy sub-leaders in projects

Talented engineers should be effectively assigned to the development team. The ADM method gives us tools to assign sub-leader. Sub-leader candidates are specialized in particular architectural elements, for example, UI. The limited number of candidates should be divided into core-asset and product-specific teams with the goal of efficient management. The number of sub-leaders is calculated as N multiplied by the ratio of lines of code (LOC) in core assets or product-specific programs to the total LOC, where N is the number of sub-leaders. Because the calculated results usually contain a decimal point, the number needs to be rounded up to take into consideration the difficulty of the programs. Otherwise, one sub-leader can be assigned to multiple teams if all the calculated results of the considered multiple teams are less than one.

Sub-leader allocation is a tough assignment. Some sub-leaders may not feel confident that they can take over a post. If their anxiety grows, it may cause the lack of motivation. One of the solutions is to discuss the allocation with all the sub-leaders based on objective data. This simple calculation objectively weighs the difficulty of posts. It worked to provide objective information to the allocation discussion and to balance the structures of teams.

Work Breakdown Structure (WBS)

The WBS introduced by the ADM method was effectively utilized in development. The engineers in development teams belonged to implementers of architectural elements, instrument-oriented implementers, or core-asset implementers. WBS played a role as a communication tool for various implementers. Because all the changes related to the final workflow elements were documented in WBS, it offered information about what engineers should discuss software interfaces. The engineers could also see which workflow elements their remodeling tasks would lead to. WBS overcame the difficulties caused by a variety of implementers and improved productivity.

7. CONCLUSIONS

We presented a new tool for estimates called the ADM method in this paper. The source codes were broken down into matrix cells of architectural elements and domain elements in the method and were transformed to the portion of core assets to be extracted from one legacy product. In addition, the method helped with team formation and WB. The ADM method generated a development plan to improve the productivity of overall development.

The core assets developed here were applied to the CIIMS of new products for the next four years. In the future we would like to report the effectiveness after repeating to use core assets through the method.

8. ACKNOWLEDGEMENTS

We would like to thank Mr. Yasuaki Takebe and our learned colleagues who pushed forward the case project. We would like to thank Mr. Kiyomi Mori of Hitachi Automotive Systems, Ltd. and Dr. Kentaro Yoshimura of Hitachi, Ltd. for their invaluable input.

9. REFERENCES

[1] H. Mitsumaki, R. Kodama, and H. Kuriyama. Module Assembly Type Automated Blood Analyzer Providing Optimized Solutions to Clinical Laboratories. *Hitachi Review*, 79(10):757-762, 1997.

[2] About progress of IT-izing and the competitive power of domestic business. Retrieved September 8, 2013, from Ministry of Economy Trade and Industry Japan: http://www.meti.go.jp/committee/materials/downloadfiles/g70124b06j.pdf.

[3] Software Product Lines | Overview (online). Retrieved September 8, 2013, from Software Engineering Institute, Carnegie Mellon University: http://www.sei.cmu.edu/productlines/.

[4] K. Yoshimura and T. Kikuno. 5 Software Product Line Adoption for Embedded Systems(<Special Feature>A New Wave of Software Reuse-Widening Software Product Line Development-). *IPSJ Magazine*, 50(4):295-302, 2009.

[5] K. Yoshimura, D. Ganesan, and D. Muthig. A Method to Assess Commonality and Variability of Existing Systems into a Product Line (<Special Issue>Software Engineering Theory and Practice). *IPSJ Journal*, 46(8): 2482-2491, 2005.

[6] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addition Wesley, 1999.

[7] A Framework for Software Product Line Practice, Version 5.0. Retrieved September 8, 2013, from Software Engineering Institute, Carnegie Mellon University: http://www.sei.cmu.edu/productlines/frame_report/index.html.

[8] W.B. Frakes and K.C. Kang. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 31(7) 529-536, 2005.

[9] K. Kang, S. Cohen, J. Hess, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-021, ADA235785). Retrieved September 8, 2013, from Software Engineering Institute Carnegie Mellon University: http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm.

[10] P. Hofman, T. Pohley, A. Bermann, et al. Domain Specific Feature Modeling for Software Product Lines. *Proc. 16th International Software Product Line Conference (SPLC)*, (2012), 229-238.

[11] K. Yochimura, F. Narisawa, and T. Kikuno. A Method to Analyze Cross-cutting Features Based on Logical Coupling Sets of Product Release History. *IPSJ Journal*. 50(11)2654-2664, 2009.

[12] K. Schmid. A comprehensive product line scoping approach and its validation, *Proc. the 24th International conference on software Engineering, ICSE '02*, (2002), ACM, 593-603.

[13] L. Northrop. Software product lines essentials. Retrieved February 4, 2014, from Software Engineering Institute, Carnegie Mellon University: http://www.sei.cmu.edu/library/assets/spl-essentials.pdf.

[14] C. Stoermer and L. O'Brien. MAP – mining architectures for product line evaluations. *Software Architecture, 2001. Proc. Working IEEE/IFIP Conference*, (2001), 35-44.

[15] J. Knodel, I. John, D. Ganesan, and et al. Asset Recovery and Their Incorporation into Product Lines. *WCRE '05: Proc. 12th Working Conference on Reverse Engineering*, (Washington, DC, USA, 2005), IEEE Computer Society, 120-129.

[16] H. Koziolek, T. Coldschmidt, T. de Gooijer, and et al. Experiences from Identifying Software Reuse Opportunities by Domain Analysis. *Proc. 17th International Software Product Line Conference (SPLC)* , (2013), 208-217.

[17] L. G. Jones and J. K. Bergey. *Exploring Acquisition Strategies for Adopting a Software Product Line*. CARNEGIE-MELLON UNIV COLORADO SPRINGS CO, 2010.

[18] Takebe, Y., Fukaya, N., Chikahisa, M., Hanawa, T., and Shirai, O. Experiences with software product line engineering in product development oriented organization. in *Proceedings of the 13th International Software Product Line Conference*, Carnegie Mellon University, (2009), 275-283.