FC method: A Practical Approach to Improve Quality and Efficiency of Software Processes for Embedded System Revision

Kazuma Aizawa
Dept. of Computer Science
Shinshu University
and
Epson Kowa Corporation
1077-5, Otsu, Shimonogo, Ueda-shi
Nagano-ken, 386-1214, Japan
aiz@epkowa.co.jp

Haruhiko Kaiya
Dept. of Computer Science
Shinshu University
4-17-1 Wakasato
Nagano-ken 380-8553, Japan
kaiya@cs.shinshu-u.ac.jp

Kenji Kaijiri
Dept. of Computer Science
Shinshu University
4-17-1 Wakasato
Nagano-ken 380-8553, Japan
kaijiri@cs.shinshu-u.ac.jp

Abstract

We introduce a design method for revising embedded software system. Engineers can accept requirements changes of hardware components and functions reasonably because design documents are managed in small unit. We can apply this method stepwise because this method can be coped with a development process that heavily depends on the hardware structure. We report an application of this method in our company so as to validate it. From the application, we can confirm that the quality of software was improved about in twice, and that efficiency of development process was also improved over three times.

Key Words

Embedded Software System, Software Process Improvement, System Revision.

1. Introduction

Traditionally, software products in embedded systems are decomposed into tasks and the products are developed in each task by each software engineer. The reasons are that software products should contribute to make full use of hardware components, and that development method based on the tasks seems to be one of the best ways to do so. However, such method becomes unfit for embedded software today, because existing software products can not be easily modified so as to meet next versions of embedded system. Note that hardware components in embedded systems are frequently changed today. If such methods are used continuously, software products e.g., source codes and design documents, become unmanageable. As a result, software engineers cannot understand and update existing software products correctly and efficiently.

In this paper, we will propose a new design method, FC (Functional Component) method, to overcome such problems, and reports an experience to apply the method into a real software project. In FC method, software products are decomposed into functional components, each of which is independent to specific hardware components. Through the experience, we found that FC method helped software engineers to add new functions and to modify existing functions based on the existing software products. We also found that the number of defects decreased by about half, and that the number of reviews for each design document was decreased by about one third.

The rest of this paper is organized as follows. In the next section, we clarify usual practice of embedded software system development, and its problems. So as to resolve several parts of the problems, we introduce FC method in Section 3. We applied FC method into practice and compared the results with results of usual practice. In Section 4, we report such practices and discuss the differences so as to validate FC method. Finally, we summarize current results and show the future works.

2. Current Practices and Their Problems

In this section, we explain how we develop embedded software systems in our company today, and define some terminologies in this paper. At least in Japan, our company is typical one in the field of embedded software systems. So

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

as to clarify the characteristics of software development for revising a system, we first explain the way of developing brand-new systems. Next we explain the way of system revision by comparing with the previous way, and clarify the problems of system revision. Because of the economic and/or organizational reasons, we can not inherently solve all of the problems. We finally discuss which problems can be technically solved or not in this section.

2.1 Brand-new System Development

An embedded system consists of several hardware components and the system performs functions by using the components. For example of a graphic scanner system, a CCD camera, motors and other hardware components work cooperatively so as to preview a scanned image. From the view point of software engineers, requirements for the system are characterized by the kinds and structure of hardware components, the kinds of functions and their non-functional characteristics. For simplicity, we do not handle non-functional characteristics after this.

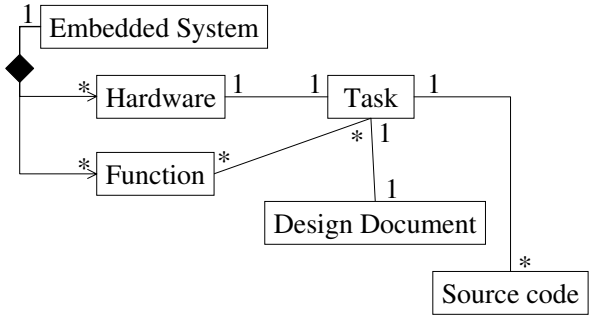

Figure 1. Concepts and Products in Brand-new System Development

By using Figure 1, which is written using a simple class diagram, we explain how to develop software products in an embedded system now. As already mentioned before, an embedded system consists of hardware components and functions. In Figure 1, we simply call 'hardware components' as 'hardware'. Because software in an embedded system works on a real-time and multi-tasking operating system, we should identify tasks on the operating system. Traditionally, such tasks and the structure of the tasks are defined by the kinds of hardware components and their structure, and each task are normally related to one hardware component as shown in Figure 1. There are several reasons of such tradition. First, software system should make full use of hardware resources. Second, hardware is still more important than software in embedded system projects. Because the tasks and their structure are decided in advance, design documents for software are written in each task. According to each document, software products such as source codes are developed along with waterfall model. Although functions of the embedded system are not simply related to design documents as shown in Figure 1, this kind of development processes has worked well.

2.2 Revised System Development

Nowadays, brand-new embedded systems are rarely developed [3] because we should release products quickly and cheap. Instead of brand-new systems, we develop embedded systems by revising existing systems and their related products. We call such embedded systems as 'revised systems', and their system development as 'system revision' in this paper. So as to complete system revision successfully, we should efficiently reuse software assets as much as possible. As a result, embedded systems are repeatedly revised in general.

As shown in Figure 1, an embedded system consists of hardware components and functions. Therefore, software engineers face two kinds of requirements changes in system revision; one is the changes of hardware components and another is functional changes. Changes of hardware components are frequently occurred because such components are improved quickly and because cheaper and/or more efficient components with same functions are released. Changes of functions are also occurred frequently because we normally develop family of similar products based on the same existing product and the products have different functions from others in the family. For example, the high-class model in a family has all functions but normal-class model only has the subset of them.

In the case of our company, it takes about a year for a project of a system revision, and about ten software engineers engage in the project. Hardware components and their structure are usually defined by another company in advance. Because several projects of system revision based on the same product has progressed simultaneously, we can not say the cycle time of system revision exactly. However, such projects are usually performed without interval.

2.3 Problems in Revised System Development

As mentioned above, we should efficiently reuse software assets as much as possible in system revision. In addition, we should of course assure the quality of software. For such software quality, design documents will help software engineers to identify the impacts of changes on software products and to follow the behavior of each function. Unfortunately, our current practice can not support such help in software revision, and we can not change our current practice completely because of the economic and organizational

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

reasons. In the rest of this section, we explain the problems in detail and explore what we can do under our restriction.

We have already identified two kinds of changes following system revision; one is changes of hardware components and another is functional changes.

When changes of hardware components occur, it is difficult for software engineers to reuse design documents efficiently. If one hardware component is replaced with another component and these two components are completely different, engineers can not reuse current design document completely. In addition, engineers can not reuse source codes for a task corresponding to the component. In most cases, engineers may revise design documents and reuse several source codes because replaced component is similar to old one. However, responsibility of a hardware component for a function could be changed under such replacement, and it becomes difficult to update design documents, so that engineers can identify the impacts of such changes and can follow the behavior of functions.

Because the structure of tasks depends on the hardware structure and design documents are written for each task as shown in Figure 1, the structure of software is usually the same as hardware structure in brand-new systems as shown in the top half of Figure 2. When hardware components are replaced, added and/or deleted, the whole structure of hardware components is usually changed. Although software structure should be changed in the same way as hardware structure, software structure is not changed so, as shown in the bottom half of Figure 2. So as to reuse design documents and source codes as much as possible, software structure can not be changed in the same way as hardware one. In addition, there are no enough budgets and time to reconstruct software products because the degree of software changes is not directly related to the degree of hardware changes, and the budget and time are decided by the degree of hardware changes. As a result, software engineers can not maintain design documents sufficiently, but several source codes are reused without suitable design documents.

When functional changes occur, it is not so easy to identify impacts of such changes because each function usually related to many tasks, and design documents are written in each task as shown in Figure 1. As a result, design decisions for a function are distributed to many design documents for tasks, and it is not so easy to follow the behavior of each function too.

Even if there are no significant changes of both hardware components and functions, it is not so easy for engineers with documents of each task to identify change impacts and to follow functions' behaviors. One reason is that such design documents are usually too large to be reviewed at once, and another reason is that such documents do not correspond to each function directly as shown in Figure 1. In addition, the sizes of design documents are not uniform

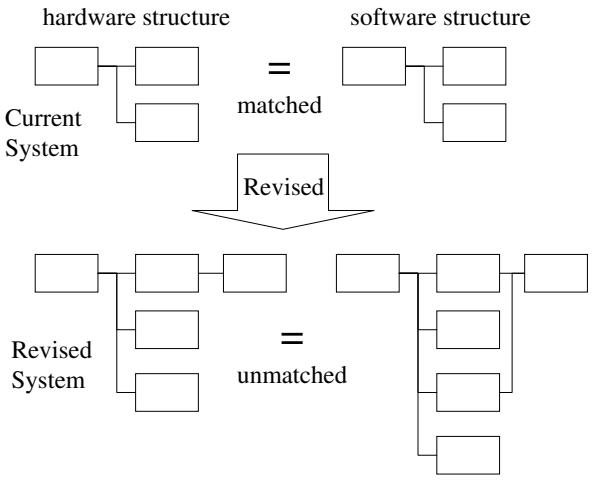

Figure 2. Inconsistency between Hardware and Software Structures

because the ability and the role of a hardware component are intrinsically different from others. When impacts of changes can fall into one task, we do not mind inconsistencies as shown in Figure 2. However, the related design document becomes fat and it becomes hard for software engineers to review such fat document.

Whenever changes of hardware components and functions occur, it is better to reconstruct design documents so as to match new hardware components and functions. However, we can not do so because the degree of software changes is not directly related to the degree of hardware changes and, the budget and time are decided by the degree of hardware changes.

It is better to develop software components and their structure independent to the hardware structure. However, we can not survive without assets of existing software products and they strongly depend on tasks and each task depends on hardware components. In addition, software engineers should take hardware components and their structure into account for performance requirements.

As a result, software engineers should stepwise revise software assets so as to meet requirements changes. At the same time, software engineers improve design documents so that they can easily identify impacts of such changes and follow the behavior of functions.

3. FC Method

3.1 Goal of FC method

So as to improve our software process, we hold up the following goals.

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

1. Change the relationship among products in Figure 1 to one in Figure 3. In other words, maintain design documents not for each task but for each function.
2. Reduce the size of design documents as small as possible.
3. Unify the size of design documents as much as possible.

We should achieve the goals not at once but stepwise because we have revised a family of systems repeatedly, and we do not have enough budgets and time in each revision so as to achieve the goals at once.

We call a pair of a function and its design document in Figure 3 as a *Functional Component (FC)* in this paper. Above goals can be regarded as the requirements for good FCs.

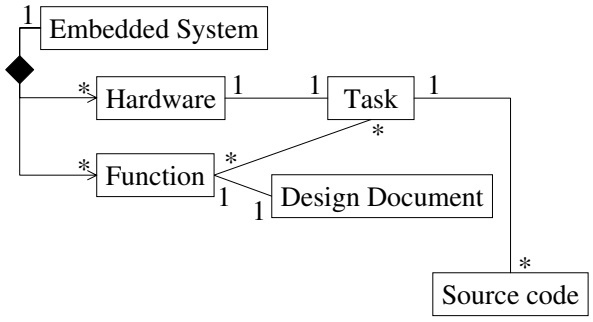

Figure 3. Relationship among Concepts and Products in FC method

If these goals are achieved, software engineers can satisfy requirements for embedded software systems reasonably. Expected consequences and effects by achieving the goals are as follows.

- Engineers can easily follow the behavior of functions.
- Engineers can clearly separate the concern about design from the concern about implementation because design documents are written in each function. When the documents were written in each task, engineers tended to take implementation issues into account too much during design phase.
- Engineers can perform incremental development[4] and can make each increment to be relatively small. When increments are small, engineers can satisfy unexpected changes of requirements with small loss of work.
- Engineers can estimate their efforts because the size of design documents are unified. From our experiences, the size of a document is correlated with the effort with which designs written in the document are implemented. If the size of documents are unified, we can estimate such efforts by counting the number of documents.
- Engineers can explore alternatives of a design easily because each design document is small enough to be rejected. If such rejection will occur, there is not so large impacts for progress and for the other products.
- Engineers can decrease the number of tasks related to a design document, and they do not mind the mutual relationships among tasks and functions so much. FC method intrinsically makes the number of documents increase because documents are written in each function and a function is related to many tasks in general as shown in Figure 4. In addition, several documents could be related to the same task, e.g. task5 in Figure 4.

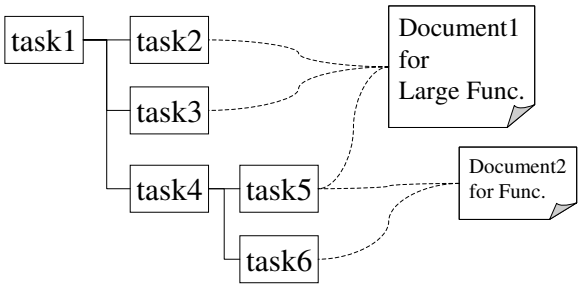

Figure 4. Task Structure with Large Documents

So as to mitigate the impacts followed by the situation in Figure 4, the size of documents for each function should be reduced as small as possible as shown in Figure 5. For example, engineers should take most tasks and documents into account when they review document1 in Figure 4. On the other hand, they only take into account task2 and 3 into account when they review document1a in Figure 5.

3.2 Procedure

3.2.1 Overall

As we mentioned above, software engineers should stepwise change our software development process because of the economic and organizational reasons. Here we show the overall way of FC method.

1. Get the requirements for an embedded software system. There are two kinds of requirements; one

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

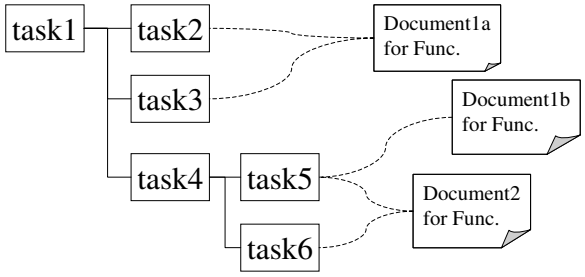

Figure 5. Task Structure with Small Documents

is changes for hardware components and another is changes of functions. These requirements normally do not take assets of an existing system to be revised into account.

2. Find an existing system and its assets to be revised for the requirements. Build a development team using engineers who belonged to the team of the existing system if possible.

3. Put members of the team in charge of each design document. Basically, an engineer should take charge of documents that were taken by him before. If the existing system was developed in usual way, the documents correspond to each task as shown in Figure 1. If FC method was already introduced in its development, the documents correspond to each task or each function.

4. Identify FCs that satisfy the requirements. We will mention how to identify them in later part of this section.

5. Identify the relationships between existing tasks and FCs. As shown in Figure 3 and 4, the relationships are normally many to many mapping.

6. Put members in charge of each FC. If a FC is related to several tasks, decide a member who takes charge of the FC by referring the skill of each member and/or by checking which task is most related to the FC.

7. Complete each design so that designs can be implemented. This part is also mentioned in the following part separately.

3.2.2 Identify Functional Components

The core of FC method is how to identify better FCs (functional components) as many as possible. Currently, we use Cleanroom approach[5, 6] loosely. In Cleanroom approach, requirements are refined into black box specifications at first. A black box specification is refined into state box specifications, each of which encapsulates state data and services, if the black box can not be refined into other black boxes. Finally, A state box specification is refined into clear box specifications if the state box can not be refined into other state boxes.

Although FCs almost correspond to such boxes of Cleanroom approach, we think much of unifying the size of each specification. Therefore, refinement is continued even if all parts of the specification are refined into clear boxes. Currently, we try to unify the pages of specification documents so as to review each document within one hour. Unified number of pages is defined by each project leader,

Another significant difference between FC method and Cleanroom approach is that engineers do not strictly obey formal verifications. For example, we do not verify formal correctness and do not apply stepwise refinement rules strictly. There are several reasons about this. One is that engineers normally do not have skills enough to verify specifications formally. In the same case of other software systems, requirements for embedded software systems are continually requested during a development process too. Another reason is that strict application of formal verification seems to be harmful for such development process.

Instead of formal verifications, informal reviews for FCs are performed so as to minimized the impacts among FCs and so as to minimized the charge of each engineer.

When a new requirement is requested during the process, engineers sometimes have to identify FCs again or to modify them. Because engineers structurally decompose requirements into FCs, engineers can easily identify where they have to modify.

3.2.3 Implement Functional Components

Another important point is how to implement design documents for each FC, and how to utilize (old) design documents for each task. As mentioned above, design documents are developed for each function. After the design phase, each task is implemented by an engineer according to waterfall model in the same way as current practice.

Because each design document does not correspond to a task directly, the engineer should refer several design documents at the same time. However, this fact does not become a disadvantage of FC method because design documents are small enough to review more than one documents, and engineers can communicate with each other right now. Because this kinds of communication helps them to build mutual understandings about the system, it seems rather an advantage of FC method. If engineers in a team are geographically distributed, some kinds of CSCW support will be needed.

Because each design document is taken charge by an engineer, one document is maintained and updated only one

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

engineer. When an engineer finds a part in a design document which should be changed or be transferred into another document after design phase, the engineer should not modify the document by himself but should request it to an engineer who takes charge of its design. The engineer may sometimes modify it by himself because the engineer himself takes charge of its design.

Finally, we explain how to cope design documents by FC method with documents by the usual way. Figure 6 shows the outline. In the figure, 'FC Document' means design documents by FC method and 'Old Document' means design documents by the usual way. Because each engineer still takes charge of tasks, he also takes charge of documents of such tasks. At the same time, he takes charge of FC documents. Therefore, he has the responsibility to cope design documents by FC method with documents by the usual way.

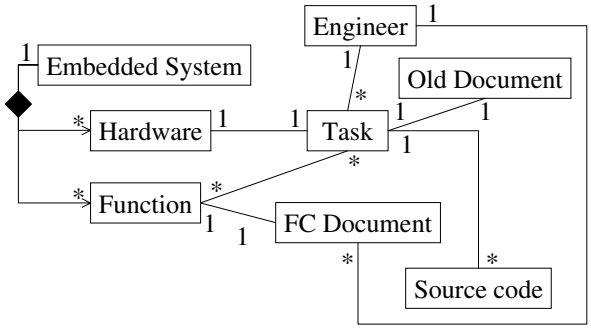

Figure 6. Relationship among Concepts and Products in FC method (version 2)

4. Using FC method into Practice

We applied FC method into a real project, say *Project FC*, in our company. So as to confirm the effectiveness of FC method, we report Project FC and another project, say *Project TC* (Task based Components), which was carried out in usual way, and compare these projects.

4.1 Project TC

In Project TC, an embedded system was revised as an extended system of an existing system. Requirements for the revised system are as follows;

- Several functions should be added because of market trends.
- Several hardware components should be replaced because new and good hardware components can be available.

When the existing system was developed, there was no plan for such revision. Therefore, the existing system and its related products were not ready for such revision. Because the existing system was developed in usual manner as mentioned in Section 2, design documents for software were written in each task, and each engineer maintained each design document and implemented source codes corresponding to each document.

4.2 Project FC

In Project FC, other embedded systems were revised as extended systems of another existing system, that was different from the existing system for Project TC. However, requirements for revision and the characteristics of the existing system were almost the same as the case of Project TC. The most significant difference between Project TC and FC was that two different systems for the existing system were revised at the same time in Project FC, but only one system was revised in Project TC. Therefore, the size of software products, such as source codes and design documents, in Project FC were intrinsically different from the size in Project TC. However, the size of requirements changes for a system in Project FC was almost the same as the size for a system in Project TC.

Project TC and FC were performed by almost the same members and terms in average. The differences are as follows.

- Project FC was started after Project TC was finished.
- About 30% of project members were changed between two projects.

4.3 Comparison and Discussion

We want to confirm that FC method will contribute to improve the quality of software products and efficiency of work. We measure the following two metrics for this purpose.

4.3.1 Defect Density

The quality of software is basically characterized by the defect density. Therefore, we compare defect densities of two projects. Because these two projects were performed by almost the same engineers of the same company, we simply use kilo lines of code (KLOC) as the product size. As the result, we calculate defect density as the following equation.

$$\text{Defect density} = \frac{\text{Number of defects}}{\text{KLOC}}$$

Table 1 shows the results. Clearly, the quality of software seems to be improved in Project FC.

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

Table 1. Defect Density of Each Project

Project	KLOC	# of Defects	Density
TC	34.410	252	7.3
FC	62.858	221	3.5

4.3.2 Number of Review per Design Document

We have already argued that inefficient tasks were performed in our current practice. Especially, we reviewed one design document redundantly because such documents were written in each task and it was necessary to review one document many times so as to follow the behaviors of functions.

Table 2 shows that how many times one design document was reviewed in average. Clearly, the number is decreased in Project FC, and we may infer that we could avoid inefficient review tasks by FC method.

Table 2. Number of Review per Design Document

Project	# of Doc.	# of Review	# of Review / # of Doc.
TC	40	89	2.26
FC	480	318	0.66

5. Conclusion and Discussion

In this paper, we introduce a develop method, namely FC method, for system revision of embedded software. For validating FC method partially, we applied FC method into practice in our company. At least in Japan, companies of embedded software systems are not large enough to introduce state of the art methods e.g. OCTOPUS[2] and ROOMS[7] immediately. However, FC method can be applied into companies like ours, because FC method can be coped with our current practice. We believe we can improve ourselves so that we can use state of the art methods stepwise by using FC method.

Current FC method do not provide all of the expected effects as mentioned in Section 3.1. For example, an estimation method for efforts is not provided yet. In embedded software revision, it is very important to estimate the efforts corresponding to additional requirements because we should ship our product on time so as to survive in the market. We are going to update FC method so that engineers can estimate efforts by using the size of design documents. That is the another reason why we stick to unifying the size of design documents. We can use existing estimation method like function points [1] easily if the size of each design document reflect the effort to implement the design. This is our next goal of this research project.

References

[1] A. J. Albrecht and Gaffney. Software Function, Source Lines of Code, and Development Effort Predicition: A Software Science Validation. *IEEE Trans. on Software Engineering*, 9(6):639–648, Nov. 1983.

[2] M. Awad, J. Kuusela, and J. Ziegler. *Object-Oriented Technology for Real-Time Systems*. Prentice Hall, 1996.

[3] B. Graaf, M. Lormans, and H. Toetenel. Embedded Software Engineering: The State of the Practice. *Software*, 20(6):61–69, Nov./Dec. 2003.

[4] J. McDermid and P. Rook. *Software Engineering Reference Book*, pages 15/26–15/28. CRC Press, 1993. Software Development Process Models.

[5] H. D. Mills, M. Dyer, and R. Lnger. Cleanroom Software Engineering. *Software*, 4(5):19–24, Sep. 1987.

[6] J. H. Poore and C. J. Trammell. *Cleanroom Software Engineering*. Blackwell Publisher, Oxford, England, 1996.

[7] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, 1994.

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**