PAPER Special Section on Software Engineering for Embedded Systems

Practical and Incremental Maintenance of Software Resources in Consumer Electronics Products

Kazuma AIZAWA[†,††], **Haruhiko KAIYA**[††a)], ***Nonmembers***, ***and*** **Kenji KAIJIRI**[††], ***Member***

SUMMARY We introduce a method, so called FC method, for maintaining software resources, such as source codes and design documents, in consumer electronics products. Because a consumer electronics product is frequently and rapidly revised, software components in such product are also revised in the same way. However, it is not so easy for software engineers to follow the revision of the product because requirements changes for the product, including the changes of its functionalities and its hardware components, are largely independent of the structure of current software resources. FC method lets software engineers to restructure software resources, especially design documents, stepwise so as to follow the requirements changes for the product easily. We report an application of this method in our company 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 four times.

key words: *consumer electronics products, product revision, software maintenance, software process improvement*

1. Introduction

Traditionally, software components in a consumer electronics product are designed so as to correspond to tasks in the product, and the components are implemented by each software engineer. The reasons are as follows. First, software components should contribute to make full use of hardware components, and the tasks normally correspond to hardware components. Second, software design method based on the tasks seems to be one of the best ways to do so. As a result, existing software components can not be easily modified if hardware components in the products are changed. Unfortunately, such hardware components are frequently changed along with the revision of the consumer electronics products today. Therefore, software design method based on tasks becomes unfit for embedded software in consumer electronics products today. If such method is used continuously, software resources e.g., source codes and design documents, become unmanageable. As a result, software engineers cannot understand and update existing software components correctly and efficiently.

In this paper, we will propose a new design method, FC (Functional Component) method, to overcome such problems, and report an experience to apply the method into a real software project. In FC method, software requirements are decomposed into functional components, each of which is independent to specific hardware components. FC method also lets software engineers to make full use of existing software resources. Through the experience, we found that FC method helped software engineers to revise an existing software product effectively and efficiently. We also found that the number of defects decreased in about half, and that the average effort for review was decreased in about one fourth.

The rest of this paper is organized as follows. In the next section, we clarify usual practice for developing embedded software in consumer electronics products, and its problems. So as to resolve several parts of the problems, we introduce FC method in Sect. 3. We applied FC method into practice and compared the results with results of usual practice. In Sect. 4, we report such practices and discuss the differences to validate FC method. In Sect. 5, we discuss the characteristics of our work related to the other research results. 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 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 for consumer electronics products [1]. So as to clarify the characteristics of software development for revising a product, we first explain the way of developing a brand-new product. Next we explain the way of product revision by comparing with the previous way, and clarify the problems of product revision. Because of the economic and organizational reasons, we can not inherently solve all of the problems. We finally discuss which problems can be technically solved or not.

2.1 Brand-New Product Development

A consumer electronics product consists of several hardware components and the product 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 product are characterized according to the kinds of hardware components and their structure, the kinds of functions and their non-functional characteristics. For simplicity, we do not handle non-functional characteristics after this.

Manuscript received September 24, 2004.
Manuscript revised January 4, 2005.

[†]The author is with EPSON AVASYS Corporation, Ueda-shi, 386–1214 Japan.
[††]The authors are with the Graduate School of Science and Technology, Shinshu University, Nagano-shi, 380–8553 Japan.
a) E-mail: kaiya@cs.shinshu-u.ac.jp

DOI: 10.1093/ietisy/e88–d.6.1117

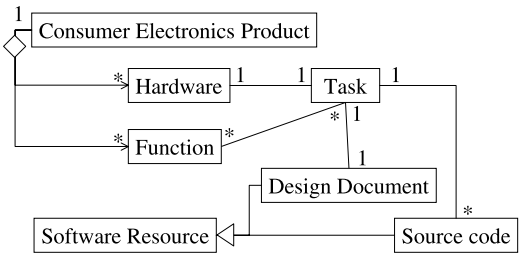


Fig. 1 Components in brand-new products.

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

2.2 Revised Product Development

Nowadays, brand-new consumer electronics products are rarely developed [2] because we should release products quickly and cheap. Instead of brand-new products, we release new products by revising existing products and by using their related resources. We call such products as 'revised products', and their development as 'product revision' in this paper. So as to complete product revision successfully, we should efficiently reuse software assets as much as possible. As a result, consumer electronics products are repeatedly revised in general.

As shown in Fig. 1, a consumer electronics product consists of hardware components and functions. Therefore, software engineers face two kinds of requirements changes in product revision; one is the changes of hardware components and another is functional changes. Changes of hardware components frequently occur because such components are improved quickly and because cheaper and/or more efficient components with the same functions are released. Changes of functions also occur frequently because we normally develop several products in the same product line [3] at the same time. There are various types of specialization of a product line that may be developed, and one of the type is functional specialization, where different versions of software are created for customers with different requirements.

In the case of our company, it takes about a year for a project of a product revision, and about ten software engineers engage in the project for the revision. Hardware components and their structure are usually defined by another company in advance. We cannot say the cycle time of system revision exactly because several projects based on the same product has progressed simultaneously and successively.

2.3 Problems in Product Revision

As mentioned above, we should efficiently reuse software assets as much as possible in product 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, such design documents cannot effectively support us during software revision, and we cannot change our current practice completely and immediately because of the economic and organizational 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 cannot reuse current design document completely. In addition, engineers cannot 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 hardware 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 Fig. 1, the structure of software is usually the same as hardware structure in brand-new products as shown in the top half of Fig. 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 Fig. 2. So as to reuse design documents and source codes as much as possible, software structure cannot 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 according to the degree of hardware changes. As a result, software engineers cannot 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 is usually related to many tasks, and design documents are written in each task as shown in Fig. 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 Fig. 1. In addition, the sizes of design documents are not uniform 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 Fig. 2. However, the related design document becomes fat and it becomes hard for software engineers to review such fat document.

The sizes of design documents for each function would not be also uniform because the ability and the role of a function are also different from other functions. Thus, we should intentionally decompose design issues into relatively small documents so that engineers can easily review them. In addition, each document should have the references to others as few as possible. In other words, design documents should be coupled loosely. If there are many references in a document, it will take a lot of time to review it. One of the criteria for the number of references in a document is the limited size of short-term memory. In a classic experiment Miller [4] found that the short-term memory can store about seven quanta of information, and our experiences also support this criterion. Consequently, the sizes of design documents become uniform during our experiences.

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 cannot 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 according to the degree of hardware changes.

It is better to develop software components and their structure independent to the hardware structure. However, we cannot survive without assets of existing software resources 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.

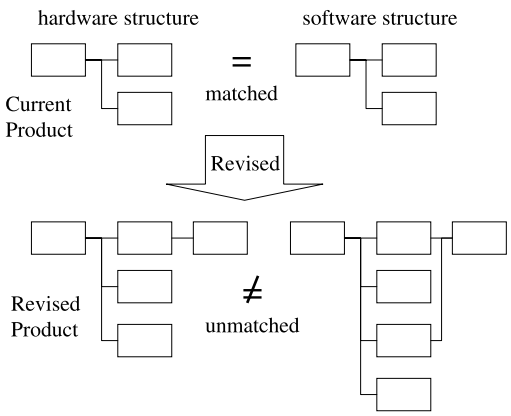


Fig. 2 Inconsistency between hardware and software structures.

3. FC Method

3.1 Goal of FC Method

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

1. Change the relationship among components in Fig. 1 to one in Fig. 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 products 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 Fig. 3 as a *Functional Component* (*FC*) in this paper. Above goals can be regarded as the requirements for good FCs.

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

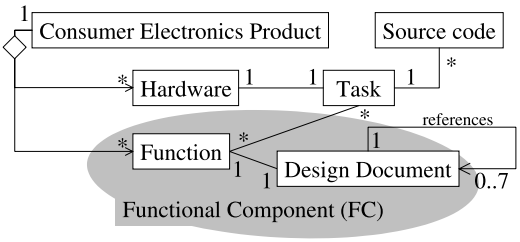


Fig. 3 Relationship among components in FC method.

1. Engineers can easily follow the behavior of functions.
2. 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.
3. Engineers can perform incremental development [5] 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.
4. 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 design issues written in the document are implemented. If the size of documents are unified, we can estimate such efforts by counting the number of documents.
5. Engineers can explore alternatives of a design issue easily because each design document is small enough to be rejected. Even if such rejection will occur, there is not so large impacts for progress and for the other products.
6. 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 Fig. 4. In addition, several documents could be related to the same task, e.g. task5 in Fig. 4.
   So as to mitigate the impacts followed by the situation in Fig. 4, the size of documents for each function should be reduced as small as possible as shown in Fig. 5. For example, engineers should take most tasks and documents into account when they review document1 in Fig. 4. On the other hand, they only take into account task2 and 3 into account when they review document1a in Fig. 5.

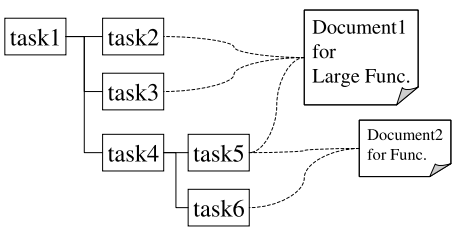


Fig. 4 Task structure with large documents.

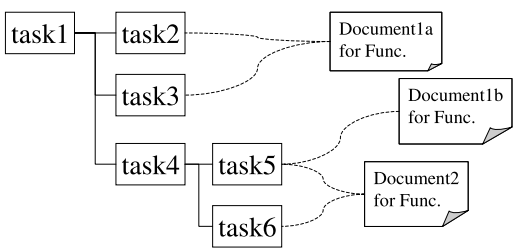


Fig. 5 Task structure with small documents.

3.2 Procedure

3.2.1 Overall

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

1. Get the requirements for embedded software in a consumer electronics product. There are two kinds of requirements; one is changes for hardware components and another is changes of functions. These requirements normally do not take resources of an existing product to be revised into account.
2. Find an existing product and its resources to be revised for the requirements. Build a development team using engineers who belonged to the team of the existing product 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 product was developed in usual way, the documents correspond to each task as shown in Fig. 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 Figs. 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 issue so that design issues 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. According to the discussion in Sect. 2.3 and our experiences, we define the concrete requirements for good FCs as follows.

- Each FC may have the references to other FCs, and the number of references shall be about seven.
- The number of pages of each FC shall be about five in A4 sized paper except the cover and reference pages.
- Each FC shall be enough small to be reviewed within about one hour.

In general, the inspection itself should be relatively short (no more than two hours) [3]. The review meetings in FC method do not have to follow the formal inspection process, but the role of these meetings is the same as the role of inspections.

We divide design issues into FCs in two steps. First, we use Cleanroom approach [6], [7]. 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 cannot be refined into other black boxes. Finally, a state box specification is refined into clear box specifications if the state box cannot be refined into other state boxes. We regard such clear boxes as the candidate of FCs first.

Second, we continue to decompose clear boxes if the boxes do not satisfy the concrete requirements above. The way how to continue to decompose them depends on the ability of the project leader, but the leader especially takes into account for the first requirement above, the limitation of the number of references in each FCs. Note that we never omit details of each clear box, but we divide a clear box into several small clear boxes intentionally. This division is sometimes against the good logical structure of software itself, but we give priority to facility for engineers to review design documents.

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 kinds of software, requirements for embedded software are continually requested during a development process. 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 FCs that 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 is taken charge by an engineer, one document is maintained and updated only one engineer. We call such engineer as a responsible 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.

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. First, there are explicit references to other documents in each FC document, thus the engineer can immediately identify which FC documents should be referred. Second, there is a responsible engineer for each FC document, thus engineers can ask the responsible engineer about the design issues maintained by the responsible engineer. If engineers in a team are geographically distributed, some kinds of CSCW support will be needed. Engineers put what they have communicated not into design documents directly but into other documents, and the other documents were not reviewed in the review meetings. We try to keep the design documents small so as to shorten the length of review meetings. Thus there is no contradiction even if the engineers communicate with each other frequently.

As already mentioned in the first paragraph in Sect. 2.3, we cannot change our current practice completely and immediately. Thus, we have to use both design documents based on FC method and design documents written for each task together. Here we show how to do so. Figure 6 shows the outline. In the figure, 'FC Document' means design documents by FC method and 'Old Document' means design documents written for each task. 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 old documents.

The way of decreasing the ratio of old documents and its outcome deeply depend on the ability of each engineer. Thus, each engineer can achieve the way at his/her best effort, and he/she can incrementally improve his/her ability during his/her job. Currently, simple web based tool is used to identify the relationships among FC and old documents in our company.

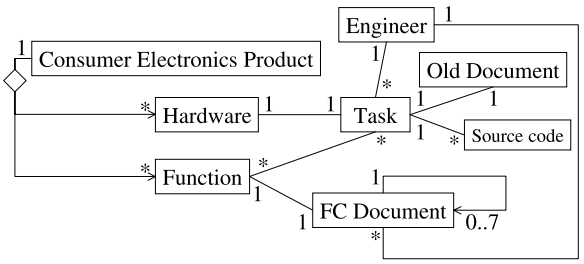


Fig. 6 Relationship among components in FC method (version 2).

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, embedded software was revised by extending existing software. Requirements for the revised software 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 software was developed, there was no plan for such revision. Therefore, the existing software and its related resources were not ready for such revision. Because the existing software was developed in usual manner as mentioned in Sect. 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 software were revised by extending existing software, that was different from the existing software for Project TC. However, requirements for revision and the characteristics of the existing software were almost the same as the case of Project TC. The most significant difference between Project TC and FC was that two different products were revised at the same time in Project FC, but only one product 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.

We don't have to take into account of the learning effect of the members because all the members have enough experiences to develop this kind of software. In other words, we cannot say that some improvements in Project FC was due to the experience in Project TC because the members were already familiar with this kind of software before both Project TC and FC in this paper. Concretely, they have about 5.9 years experience in this kind of software on average, and they had already participated in more than five similar projects on average before Project TC and FC. Even the least experienced members have two years experiences.

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 codes (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.

We may regard that the software quality was improved about in twice because the defect density of Project TC is 7.3 and the density of Project FC is 3.5 (7.3/3.5 = 2.08).

4.3.2 Number of Reviews and Design Documents, Length of Reviews

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 the number of documents and reviews, an average length of review meetings and their derived data.

First, we discuss the difference about the average length of review meetings. As shown in Table 2, the average length of reviews in Project FC (1 hour) was clearly shorter than the length in Project TC (8 hours). Thus we may conclude two things. First, design documents in Project FC satisfied the last requirement in Sect. 3.2.2. Second, the requirement was not satisfied while we did not impose FC method on the members in Project TC. Thus we may conclude that the review activities became better by FC method. FC documents would contribute to shorten the length of reviews because FC documents are relatively smaller than design documents in Project TC, and FC is designed so that

Table 1 Defect density of each project.

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

Table 2 Number of reviews and design documents, average length of reviews.

Project	# of Doc.	# of Review	Average Length of Review (Hours)	Total Hours for Review (Hours)	# of Review / # of Doc.
TC	40	89	8	712	2.26
FC	480	318	1	318	0.66

the coupling among FC becomes low.

Second, we discuss the productivity. As shown in Table 2, total spending hours for review meetings in Project FC was clearly smaller than the hours in the Project TC, even though two different products were revised at the same time in Project FC. Thus, the productivity was improved about in four times (712/(318/2)=4.5). This result also supports the advantages of FC method.

Finally, we discuss the redundancy. As mentioned in the first paragraph in Sect. 4.3.2, design documents are reviewed redundantly and that's one of the reasons for inefficiency. Average review numbers for each design document will show the extent of such redundancy. Table 2 shows that how often one design document was reviewed in average. In the case of TC, one document was reviewed in more than two review meetings (2.26). On the other hand, one document was reviewed in less than one review meeting (0.66) in the case of FC. In other words, several FC documents were reviewed at the same time in a meeting. As a result, we may infer that we could avoid redundant review tasks by FC method, because design documents were reviewed less frequently in FC than those in TC.

5. Related Work

There are many kinds of products with embedded software, and a consumer electronics product is one of the important product today. The characteristics of the product were summarized in [8] about ten years ago, and they are still true now. In this section, we discuss related works with respect to several viewpoints.

5.1 Applicability to Actual Organization

We have already known many methods [9], [10] that seem to be effective for developing embedded software. However, most companies for embedded software cannot introduce such methods immediately because they cannot invest money and time in educating their engineers. Because FC method does not include difficult and advanced idea, e.g. object-oriented methods and formal methods, most engineers can easily shift the way of developing software. FC method can be regarded as a relay point to more difficult methods. For example, engineers can easily and intuitively identify an abstract data type by grouping several functional components.

In an article [11], the necessity for software process improvement (SPI) in embedded software was argued, and there already exist several proposals and practices in this area [12], [13]. By using FC method, engineers can use both legacy and FC documents simultaneously. By measuring the ratio of FC documents to all documents and by tracing the changes of the rate, we can provide another SPI method by using FC method.

5.2 Reusability of Legacy Resources

For some kinds of embedded software, issues in real-time processing are very important and formal methods are suitable for resolving these problems [14], [15]. However, issues about predictability and lead-time reduction [8] are more important than those about real-time processing in the field of consumer electronics products.

For the issues in lead-time reduction, a family of techniques for reusing software components seems to one of the effective way. For example, patterns for embedded software are proposed [16]. UML notations are used to write design and specification for embedded software [17]. An implementation independent design notation and its refinement method are proposed [15]. Design issues are divided into several aspects each of which can be easily certified with respect to a certain characteristic in requirements [18]. These techniques are effective, but they do not mind to reuse existing legacy software resources. Practically, we never scrap documents and source codes that worked well in previous release even if they are mal-formed. FC method allows engineers to reuse and to modify such legacy resources.

5.3 Capability to Improve Organization and Each Engineer

As mentioned before, each engineer can easily monitor his/her improvement simply by monitoring the ratio of FC documents to all design documents. On the other hand, it is not so easy for engineers to identify their improvement when they use object-oriented and/or formal methods. So as to encourage each engineer to use more sophisticated methods, we need SPI method to persuade engineers to understand/feel their advantages in addition to teach such methods.

One of the ways is to provide metrics and measurement to know his/her own improvement [19]. We can find several researches how to teach different concepts in embedded software [20]. However, we cannot find researches how to persuade engineers/students to feel advantages of advanced concepts and techniques.

5.4 Modeling Process and Products

One of the characteristics of this research is to model a consumer electronics product with respect to its embedded software as shown in Fig. 6. We can specify how to cope with

legacy software resources and requirements changes when we revise the product within limited budgets and time. We can also know how many current software resources are improved with respect to independency of hardware components.

We can find models for embedded software, and they are designed for different purpose. For example, a model is designed for traceability and evolution of embedded systems [21], and it also provides the way to analyze impacts by requirements changes. Our model also supports traceability among requirements, design and implementation, but there is not explicit way to analyze such impacts.

Cost estimation model is provided in [22]. Our model also handles the cost estimation by using the number of FC components. Because we do not validate our cost estimation way, we cannot decide our way is enough or not. If our model is too simple to estimate costs, we will refer the model in [22].

More general, rigorous and formal model is also provided [23]. Such model will contribute to compare and combine various kinds of models including ours.

6. Conclusion and Discussion

In this paper, we introduce a software design method, namely FC method, for revising consumer electronics products with embedded software. For validating FC method partially, we applied FC method into practice in our company.

At least in Japan, companies of embedded software are not so large enough to introduce state of the art methods immediately. To introduce such methods, a company should have sufficient education systems. However, small companies have no enough money for education systems in general. We found one kind of data in a report [1]. In the report, they investigated what is the most important policy for embedded software industries (Figure 27 in [1]). The results were different according to the scale of companies. Large companies thought that it was import to build skill standards and qualification systems for skills. On the other hand, small companies needed support for education systems.

FC method can be applied into companies like ours, because FC method can be coped with our current practice. By using FC method, we can also improve our practice stepwise because FC method allows us to use both legacy and FC design documents at the same time.

In Sect. 3.1, we listed six expected consequences and effects of FC method. We finally discuss which consequences could be substantiated or not through the analysis in Sect. 4.

1. One of the main activities in the review meetings is to follow the behavior of each function. As shown in Table 2, the average length of reviews became shorter. After Project TC and FC, we interviewed members of these projects, and confirmed that they could easily follow the behavior in Project FC.
2. We don't have data directly supporting this effect. However, we also confirmed that engineers could focus on the design issues in Project FC. Especially, meeting facilitator said that they could avoid arguing subjects that were not related to software design.
3. We don't have data supporting this effect too because incremental development was not achieved. Fortunately, there were few unexpected changes of requirements in Project TC and FC.
4. In the Project FC and TC, we didn't explicitly introduce an estimation method like function points [24]. However, most engineers said that each of them could intuitively identify the progress of his/her works simply by counting the number of documents corresponding functions were completed. Thus they could agilely managed their own software development processes.
5. We don't have data supporting this effect too. However, the impact by rejecting a design document would not be strong in the case of Project FC. As shown in Table 2, the number of documents for a system in Project TC was six times as many as the number for a system in Project FC ((480/2)/40=6). Thus, the impact would be six times stronger when a document was replaced with an alternative in Project TC.
6. We don't have data directly supporting this effect too. However, most documents in Project FC were referred in only one review meeting but those in Project TC were referred in more than two review meetings as shown in the last columns in Table 2. If a document related to many functions and/or tasks, it would be referred frequently. Thus, the members in Project FC would be able to decrease the number of interrelationships among functions and tasks.

References

[1] Commerce and Information Policy Bureau. Report on the Actual Conditions of Embedded Software Industry in Japan. Technical report, Ministry of Economy, Trade and Industry, Japan, 2004. (In Japanese), abstract is available via http://www.meti.go.jp/policy/it_policy/technology/softjittaityousa-gaiyou.pdf

[2] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: The state of the practice," Software, vol.20, no.6, pp.61–69, Nov./Dec. 2003.

[3] I. Sommerville, Software engineering, International computer science series, 6th ed., Addison-Wesley, 2001.

[4] G.A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," The Psychological Review, vol.63, pp.81–97, 1956.

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

[6] H.D. Mills, M. Dyer, and R. Lnger, "Cleanroom software engineering," Software, vol.4, no.5, pp.19–24, Sept. 1987.

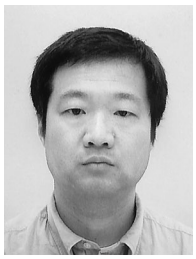
[7] J.H. Poore and C.J. Trammell, Cleanroom Software Engineering, Blackwell Publisher, Oxford, England, 1996.

[8] J. Rooijmans, H. Aerts, and M. van Genuchten, "Software quality in consumer electronics products," Software, vol.13, no.1, pp.55–64, Jan. 1996.

[9] M. Awad, J. Kuusela, and J. Ziegler, Object-Oriented Technology

for Real-Time Systems, Prentice Hall, 1996.
[10] B. Selic, G. Gullekson, and P. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994.
[11] A.C. Lear, "Shedding light on embedded systems," Software, vol.16, no.1, pp.122–125, Jan./Feb. 1999.
[12] R.S. Oshana and R.C. Linger, "Capability maturity model software development using cleanroom software engineering principles — Results of an industry project," Thirty-second Annual Hawaii International Conference on System Sciences, vol.7, p.7042, Jan. 1999.
[13] J. Taramaa, M. Khurana, P. Kuvaja, J. Lehtonen, M. Oivo, and V. Seppanen, "Product-based software process improvement for embedded systems," 24th EUROMICRO Conference, vol.2, pp.20905–20912, Aug. 1998.
[14] M. Jersak, K. Richter, R. Ernst, J.-C. Braam, Z.-Y. Jiang, and F. Wolf, "Formal methods for integration of automotive software," Design, Automation and Test in Europe Conference and Exhibition, pp.20045–20050, March 2003.
[15] J.W. Rozenblit and S. Schulz, "Refinement of model specifications in embedded systems design," Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp.159–166, April 2002.
[16] S. Konrad and B.H.C. Cheng, "Requirements patterns for embedded systems," IEEE Joint International Conference on Requirements Engineering, pp.127–136, Sept. 2002.
[17] G. de Jong, "A UML-based design methodology for real-time and embedded sytems," Design, Automation and Test in Europe Conference and Exhibition, pp.776–778, March 2002.
[18] S. Kim, F.B. Bastani, I.-L. Yen, and I.-R. Chen, "Systematic reliability analysis of a class of application-specific embedded software frameworks," IEEE Trans. Softw. Eng., vol.30, no.4, pp.218–230, April 2004.
[19] H. Suzumori, H. Kaiya, and K. Kaijiri, "VDM over PSP: A pilot course for VDM beginners to confirm its suitability for their development," Proc. COMPSAC2003, pp.327–334, Dallas, Texas, IEEE, Nov. 2003.
[20] L. Motus, "Teaching software-intensive embedded systems at Tallinn Technical University," Third IEEE Real-Time Systems Education Workshop, pp.30–35, Nov. 1998.
[21] A. von Knethen, "Change-oriented requirements traceability: Support for evolution of embedded systems," International Conference on Software Maintenance, pp.482–485, Oct. 2002.
[22] J. Axelsson, "Cost models for electronic architecture trade studies," 6th IEEE International Conference on Complex Computer Systems, pp.229–239, Sept. 2000.
[23] M. Broy, "Modular hierarchies of models for embedded systems," First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, pp.183–198, June 2003.
[24] A.J. Albrecht and J.E. Gaffney, "Software function, source lines of code, and development effort predicition: A software science validation," IEEE Trans. Softw. Eng., vol.9, no.6, pp.639–648, Nov. 1983.

Kazuma Aizawa is a PHD candidate at Shinshu University, Japan. He is also an engineer in EPSON AVASYS Corporation.



Haruhiko Kaiya is an associate professor of Software Engineering at Shinshu University, Japan.
http://www.cs.shinshu-u.ac.jp/~kaiya/

Kenji Kaijiri is a professor of Software Engineering at Shinshu University, Japan.