International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:11, 2009

Design as Contract and Blueprint – Tackling Maturity Level 1 Software Vendors in an e-School Project

Yung-Pin Cheng, Ching-Huei Wang

Abstract—Process improvements have drawn much attention in practical software engineering. The capability maturity levels from CMMI have become an important index to assess a software company's software engineering capability. However, in countries like Taiwan, customers often have no choices but to deal with vendors that are not CMMI prepared or qualified. We call these vendors maturity-level-1 (ML1) vendors. In this paper, we describe our experience from consulting an e-school project. We propose an approach to help our client tackle the ML1 vendors. Through our system analysis, we produce a design. This design is suggested to be used as part of contract and a blueprint to guide the implementation.

Keywords—CMMI, Software Engineering, Software Design, Design as Contract.

I. Introduction

Software development life cycle is often partitioned into several stages. These stages include requirement acquisition, requirement specification, system analysis and design, coding (implementation), integration testing, testing, and maintenance.

Conventionally, the stages of requirement and specification determine "what to build." The goal of these stages is to define user and system requirements and write them into specifications. In past decades, many methodologies [4], [6] have been developed for requirement acquisition, elicitation, and specification. In practice, capturing the behavioral requirements with use cases has started to dominate the industry [3], particularly for the software outsourcing industry.

System analysis and design, on the other hand, determine "how to build." Typically, system analysis and design are carried out by the software companies, which we will call them "vendor" in the later sections. The result of the stage is an artifact called "design." A design may include system architecture, modeling diagrams (e.g., UML diagrams), module interfaces, and etc. Under normal circumstances, A design is an important artifact for programmers to follow, obey, and implement.

In general, the line between specifications and analysis&design is sharp. It is a common belief in software engineering community that specifications (either written formally or informally) should be neutral from analysis, design, and coding. Specifications can serve as a contract to be negotiated between vendors and customers. So, a specification must be understandable for customers. On the other hand, how specifications are designed and implemented is up to the vendors. As long as the final system conforms to the specifications, customers are satisfied. After all, customers usually do not have the expertise to assess some of their system's important properties, such as architecture or maintainability.

In this paper, we describe the work and experiences from a real consulting case. In this case, an e-school system developed by some vendor has been installed and deployed in a county of Taiwan for two years. For convenience, we shall refer to the system, the vendor, and the county as system *X*, vendor *X*, and *Y* county[1] in later sections. The end users of this system are school teachers and staffs. The total number of schools is more than 100 and the number of teachers and staffs is more than 3000. Unfortunately, system *X* has been plagued by system flaws, unreliability problems, and low usability problems. Last year, we were invited to investigate the problem, understand the system, and then propose a feasible solution for building a system of next generation.

The conflicts between *Y* county government and vendor *X* are the major concerns of our consulting work. The difficulties that frustrate *Y* county government are that most vendors do not have adequate capability in software engineering. Using terms from CMMI, these vendors are maturity level 1. We call them maturity-level-one (ML1) vendors. To address this specific problem of *Y* county, we propose making a design as a contract and blueprint in the development of their next-generation e-school system. To complete the contract, a vendor is required to build a system that conforms to the design (we call it blueprint) we produce, rather than some specifications in an ordinary case. We build the detailed design which serves as a blueprint to direct, guide, and constrain the programmers to keep the development on the right track. Consequently, the customer, *Y* county government, can have system properties like architecture, reliability, and maintainability, controlled at their hands.

In the following sections, we will first describe the problems of system *X* and the difficult situation in which *Y* county government is trapped. We also describe how we redesign the architecture, how we use UML to capture system behaviors to certain level of details, and why our proposed solution is justified. At last we also contrast our work with the popular Agile development method[10].

Y.-P. Cheng and Ching-Huei Wang is with the Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, TAIWAN e-mail: (ypc@csie.ntnu.edu.tw).

[1]The county locates in the north-eastern part of Taiwan.

II. Fact Sheet

A. Software Industry in Taiwan

In this section, we list some facts of software industry in Taiwan. These facts constitute the context which leads to our work.

- Taiwan is a country well-known for its hardware design and manufacturing. The success of hardware OEM (Original Equipment Manufacturing) and ODM (Original Design Manufacturing) makes Taiwan a major force in global IT industry. In contrast, the Taiwanese software companies are overall less competitive in terms of the software engineering capability[7]. The number of companies which are qualified as CMMI level 2 is still in single digit. A large percentage of the companies do not take software engineering discipline seriously. Such an atmosphere can be traced back to the lack of software engineering education in computer science program (see [7] as well).
- In Taiwan's job market, entering a software company is not necessary the first choice of talented CS graduates. Many hardware companies pay more. Consequently, software companies, particularly those without financial advantages, have trouble hiring good programmers.
- With the popularity of web programming languages like PHP and ASP (which is much easier to learn than general-purpose languages like C++), many programmers without formal CS education enter this area of job market. Typically, these jobs are less paid and cannot attract talented programmers with formal CS education. This could be a cause that leads to underestimate the complexity of software development.

The facts we describe above, of course, is simply an observation from average cases. Good software companies which have financial advantages and market globally suffer much less from the above facts.

B. The e-School project

Several years ago, Taiwan government decided it was the time to initiate the idea of e-school from elementary level to junior high. Some major e-school requirements are: store students's grades and data electronically, exchange and track students's data when they transfer, support decision making for distribution of educational budget and funds, and monitor and assess the performance of teachers or schools.

Technically speaking, the e-school project should be of national-scale. However, due to political reasons, the central government decides to implement this e-school idea in a loose way: They only produce an XML standard for exchanging data between e-school systems of county government. It is county government's responsibility to develop an e-school system of their own or buy one from vendors.

Since then, there are only some e-school systems under developing or on-line for use. Most county governments make no move in a wait-and-see altitude. Some counties use an open-sourced e-school system developed by teachers. It is designed for a single school. Some counties tried a centralized e-school system (a.k.a., system *X*). System *X* is modified from the open-sourced system into a centralized version by vendor *X*. Two years ago, *Y* county government deployed this system *X*. They favor a centralized system because they want to draw useful information from the database to support decision making of education policies. System *X* is a thin-client system, where end users use web browsers to submit or get data from the centralized server. The language to implement the system is PHP. System X uses PHP scripts to access the database and returns the data in HTML to end user's web browsers.

Unfortunately, the two-year experiment with system *X* is not a pleasant experience to *Y* county government. The system is plagued by many problems. Here, we list them in items.

- The teachers in school may submit test scores at the same period of time. In peak time, large network connections jam the system occasionally. Many end users complained the loss of submitted data. They ended up in re-keying in the data.
- Before deploying the system, the user interfaces have never been surveyed by some usability test. End users complained about the complexity of user interfaces and are reluctant to use the system as a helpful toolkit. Many teachers key in their data on Excel sheet first and submit the data only when necessary.
- Before deploying the system, the system had not been fully tested under a formal testing procedure. End users are actually treated as testers in the beginning. Complaints about system flaws have never stopped. As a result, end users are reluctant to explore and use the whole system. They confine themselves to a small set of workable system functions. So, rather than enjoying the benefits and convenience of the e-school system, they passively resist this e-school policy.
- When there are flaws or requirement changes, the responses from vendor *X* take a long time. Besides, when they fixed a flaw, they introduced new flaws to the system.
- The system's source code is written without structures or explicit comments. In practice we call such code as "spaghetti code." There are very few documents for reference. Overall, no software engineering process has ever been conducted along the development.

The symptoms we described above, actually, shouldn't be a surprise to the software engineering community. These problems are typical software engineering problems.

C. The dilemmas of Y county government

The problems of system *X*, of course, are not new. CMMI is the most well-known standard to overcome such problems. However, *Y* county government must face some constraints and limitations. We list the dilemmas as follows:

- Taiwan's government projects must open to the public. Typically a project is given to the lowest legitimate bidder. If the projects are hardware constructions, such as building construction, this policy works fine for most of the time. Typically, hardware construction requires little maintenance and the maintenance may not require the same vendor. However, to a software project, the

maintenance is a big problem. The cost to replace a vendor is very high and vendors know that.
- It is legal for *Y* county government to limit the bidders to CMMI qualified vendors. However, setting such a limit would simply result in no bidders in current Taiwan's software environment. This phenomenon has been described previously.
- *Y* county government has no confidence in the capability of other candidate vendors. Records show the bidders of the e-school project in other counties have the same origin.

Overall, these facts lead to the consulting work in this paper.

III. Searching a Process Method

One year ago, we were invited to analyze the problem and propose solutions for the future. *Y* county government wants to replace system *X* with a new one. Since the replacement cost is very high, the new system should have following properties to compensate for the replacement cost.

1) They want a system that can last for decades and evolve successfully for requirement changes.
2) In case the vendor can not continue their maintenance for some reasons, they want a system which can be maintained and modified without the original vendor.
3) They want a system that is fully tested before deployment.
4) The system's user interface should undergo a rigid usability test.
5) A system that can scale and survive large number of connections in peak hour.

Their future plan is either have the new system released to new vendors or assemble a development team[2] under their computer center. In either ways, their concerns are:

- If they open the project to vendors again, how can they assure that these goals are achieved?
- If they implement the system by their own development team, an amateur team, how can they assure that these goals are achieved? After all, their programming skills are uncertain.

To meet the above expections, we first need to come up with a process improvement method to address the difficult situation where county *Y* is in.

In past decades, different kinds of software process and process improvements have been proposed to address the software engineering problems. CMMI no doubt is the most well-known process improvement standard to assess a software company's capability maturity. Besides the standard, several methods have been also proposed in these years, such as the unified process [8], [9], Win-Win spiral model[1], [2], and etc. In practice, Agile method[10] which advocates programming in pair, test-driven manner, also attracts lots of attention recently.

[2]Because of the limits in law and budget concern, it is difficult for a county government to hire talented programmers from outside. So, the programmers will be drafted from school teachers. Some teachers are talented and show great interest in programming but they typically do not have formal CS education or training.

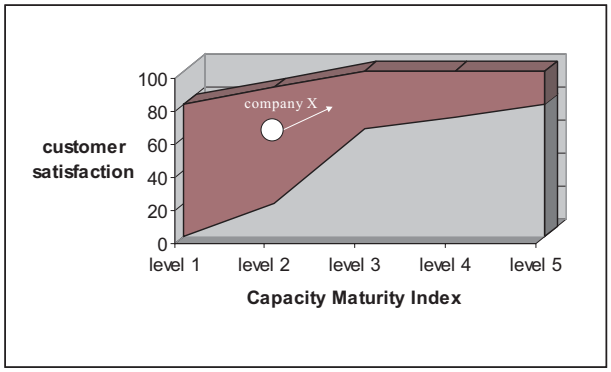

Fig. 1. The relationship between customer satisfaction and capacity maturity levels.

These process methods or standards, however, are all based on an assumption:

"By applying software process and process improvement on developers, eventually developers should deliver products with good quality. Consequently, customers are happy and the ultimate goal — customer satisfaction – is achieved."

To further explain it, in Fig. 1, we borrow CMMI's five level index as the X-axis. Average customer satisfaction is Y-axis which ranges from 0% to 100% satisfaction. Note that the percentages we show in this figure are not collected from any real data. We merely attempt to use it for the purpose of explanation. In the figure, if a vendor's capability maturity is level 5, we expect the average customer satisfaction will maintain at a high satisfaction and the variance should be small. However, if a vendor is maturity level 1 (ML1), we expect the customer satisfaction can vary dramatically. For example, in the figure, the customer satisfaction can range from 0% to 80%. There are chances that a customer can still get good service from these ML1 vendors, if a vendor happens to have talented programmers or managers who can have the job done in some informal way. However, giving a project to ML1 vendors, a client cannot be sure about the quality of results. The quality of results can range from very bad, poor, satisfactory, to good. So, by applying the process improvement methods or standards mentioned above, in the figure, a vendor's capability maturity is hopefully heading to the right. In other words, process improvement (or methods) is an action taken by the vendors themselves. It is not obliged.

Unfortunately, waiting for the vendors to better themselves in terms of software engineering capability is a luxury we don't have. There are no positions for us or the county government to impose the process on vendors. This is not the way it works. So, we decide to approach the problem in a different way. It can be explained in Fig. 2. In the figure, we accept the fact that vendors (or the amateur development team) are ML1. We perform system analysis on this project to produce a design. We make the design as part of the contract and a blueprint for future vendors to follow. The blueprint is

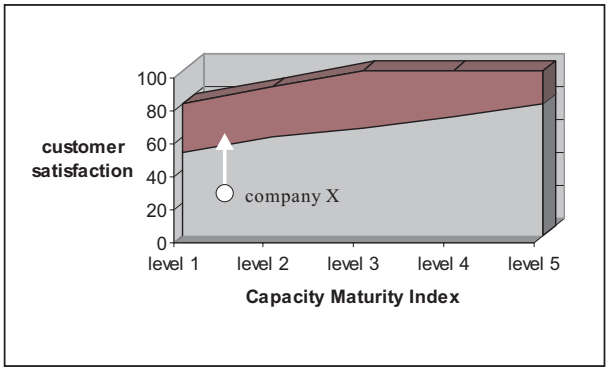

Fig. 2. Searching for a process to reduce the variance of customer satisfaction for low-level vendors.

actually a detailed design, it includes artifacts like architecture, UML diagrams, module interfaces, user interfaces, and etc. In other words, we relieve vendors (or the amateur team) of the burden in analysis and design activities. These activities are often critical and require wisdom from experienced experts.

As shown in Fig. 2, our process does not attempt to better a company to the right (higher capability maturity). Instead, we hope our process can narrow down the variance of the customer satisfaction for ML1 vendors (or the amateur team). The wedge shape of customer satisfaction in Fig. 1 can hopefully be changed into a belt shape of Fig. 2. Consequently, the mean value of customer satisfaction should also be increased (e.g., the company X).

IV. Making a Blueprint

To obtain a detailed design as a contract and a blueprint, of course, there are many questions remained to answer. These questions are:

1) How do we express our design?
2) How detailed is a design can be called a "blueprint"?
3) How can we make sure our design is workable and correct?

In these questions, some have explicit answers and some unfortunately do not. To the first question, the success of UML diagrams no doubt makes it a unique candidate for expressing the design. To the second question, the definition of a software blueprint, unfortunately, is vague. Some dictionary says a blueprint is a design. However, even the definition of a software design can be very confusing under different domains, scenarios, and context[5]. Since our work is a practical one, we simply define the blueprint in our own context.

A blueprint is a document which consists of architecture design, UML models (for describing static data relationship, dynamic behaviors, and workflow), design rationale, links to use cases, module interfaces, prototype user interface, and the needed text to guide, help, and aid the understanding of a system so that arbitrary programmers can implement the system without making efforts in highlevel activities like analysis, design, abstraction, and planning.

In the definition, the term "arbitrary programmers" is a little bit risky. We use the word simply to emphasize that our blueprint should be understandable for the programmers who never participate in our analysis or some forms of communication, i.e., the design should be self-explanatory. We also assume the programmers are familiar with UML. As a matter of fact, to what extent a blueprint is enough for programmers to understand is hard to define formally. After all, this is what make software engineering a difficult discipline.

Recall that one of *Y* county's goals is to be able to maintain the system without the original vendor. A blueprint is the solution to this goal. We hope arbitrary programmers can simply read the documents and then become capable of modifying the system in a short time. To meet this requirement, a large amount of text has been written in the blueprint to explain the design rationale, which shall provide enough clues for future maintenance.

To question 3, our solution is to validate our design by prototyping. This part of work will be explained in more details in later sections.

V. System Analysis and Design of e-School System

In this section, we describe some interesting results from our system analysis and design on the e-school system.

A. Architecture analysis

Much of the innovation in programming in recent years has involved loose coupling. The invention of database driver methodologies such as JDBC and ODBC led to applications being loosely coupled with their back end databases, allowing best-of-breed databases to be chosen - and swapped out when necessary – without any little effect on the user interface.

Nowadays, with the prevalence of web browsers, many e-commerce systems use web browser as their user interface to construct a so-called 3-tier architecture. The first tier is a thin-client user interface (e.g., web browser) which does not implement any business logics. The second tier which accepts requests from the first tier, is where business logics are implemented. The third tier is the database. Note that 3-tier framework is merely a conceptual model. The second tier and the third tier do not necessarily locate on different machines. In Fig. 3, we show a 3-tier system where the second tier is implemented using PHP web application programming languages. PHP is a scripting language[3] based on the model of preprocessing HTML pages. When the PHP preprocessor in Web server notices a PHP language tag, "<?php", the PHP engine is invoked to execute the code between the tag. Most applications contains PHP script to access database and the result is sent back to client's browser.

PHP is indeed very popular, easy to learn, and easy to write. System *X* is implemented in this way. Recall that system *X*

[3]which has a head-to-head competition with ASP from Microsoft.

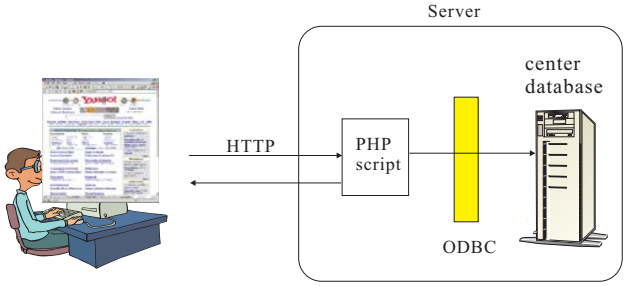

Fig. 3. A 3-tier architecture using PHP as an example.

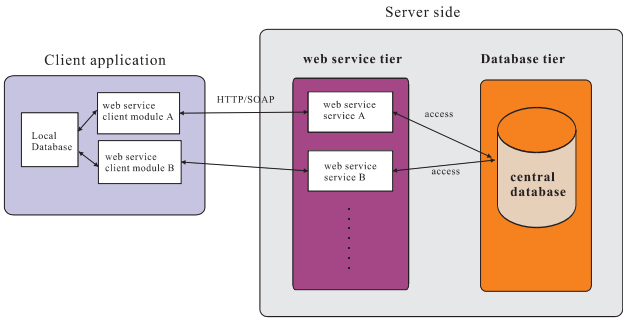

Fig. 4. The 3-tier architecture using web service.

suffers from the scalability problem in peak time. We analyzed the system and have the following observations:

1) System *X* implements user interfaces in web browsers. However, some works in the system require a convenient, powerful user interfaces. To accomplish these works, plenty of interactions and steps need to be taken care. Implementing such a user interface by a web browser has an obvious drawback - an HTTP request is sent whenever an action in the user interface is taken. This is the main cause for network congestion during peak time.
2) HTTP is stateless. Under the PHP architecture, a PHP script is executed as new every time. Same data which has already been generated in previous requests is computed again. As a result, the database is accessed frequently in a redundant way.
3) As for now, web browsers are not designed for implementing complicated user interfaces. There are several shortcomings of web browser. They are:
    a) The speed to render of a web page is slow because it needs to download and parse an HTML file. Its response time is no match for an application running locally.
    b) Its form does not memory newly keyed in data. A careless action (e.g., clicking the button “back”) or a failure to submit the data to server can lose all the data.

In the above observations, observation 1 is the key factor which prevents system *X* from scaling up during peak time. One trivial solution to the problem is increasing the network bandwidth, server’s CPU speed and memory (for parallel-running HTTP daemon process and PHP preprocessor), or investing load balancing hardware. However, these are all hardware solutions, which have its limits in hardware physics and cost. We expect a solution which can solve the scalability issue radically.

B. Architecture redesign

After we understand the causes, we propose a new architecture which is built under web services. Web service no doubt is the hottest technology to build a distributed enterprise system nowadays. The progress of XML and the commitment from heavy players like Microsoft, make web service a technology that worths investment in the long run. It will gradually replace technologies like CORBA or DCOM for building distributed applications.

Observing that the frequent interactions in user interfaces may cause the network congestion, we redesign the architecture as in the Fig. 3. In this design, the thin-client principle is reversed a little. The client side application (the 1st tier) is implemented as window applications. These applications connect the server via web services, where web services serve as 2nd tier. Under this architecture, in most of the time, the client applications stay off-line to server. Only when end users take particular actions, the connection is then established to invoke the web services. For example, in the application which computes student test scores, initially, the application will invoke a web service to download student’s names, student ids, and course ids from the server. Then, the application stays disconnected and off-line from the server. A teacher can calculate student grades in a spreadsheet-like user interfaces on a local machine, which provides more convenient functionality than a regular Excel sheet. A teacher is allowed to save its results to local files and bring them around different machines. When a teacher finishes its grading, it can use the application to submit the grades to the server by invoking another web service. So, every end user’s interaction with the application no longer triggers an HTTP connection to the server. Database is also much less accessed since same data will not be generated repeatedly. This is how we “radically” solve the scalability problem.

C. An architecture anti-pattern

Concluded from the above results, using web browser to practise thin-client principle must be careful. It depends on the complexity of the user interfaces. A wrong design choice may induce low performance, low scalability, low usability, and long response time of user interfaces.

In practice, many e-commerce systems are implemented using browser+PHP solution simply because end users do not interact with the system in a complicated way. Typically, their interactions are of patterns like COMMERCE SITE, BUY/SELL or SHOPPING CART [11]. Under these circumstances, we believe the browser+PHP solution works adequately. To deal with scalability problem, increasing hardware investments or adding some caching system, is still the ultimate solution.

On the other hand, for enterprise systems which are only open to their staffs or limited end users, if the nature of their work requires frequent user interface operations, the browser+PHP solution apparently is a lousy one. This example also shows that ML1 vendors could blindly chase the thin-client principle without understanding the story behind the scene. Also, it is a very good anti-pattern which worths documenting.

We believe many experienced architects have discovered this design choice from different systems. Many enterprise systems are indeed seldom implemented by the browser+PHP solution. When web service technology were not available then, these systems are typically implemented in a client-server framework. More advanced ones could be built under CORBA, DCOM, or RMI in Java.

VI. Prototyping to Validate Design

In a regular development process, analysts, architects, and developers typically work together as a team. So, when a design has flaws, analysts and architects can get feedback from developers to correct the flaw immediately. Nevertheless, that is a luxury we don't have. We use the term "blueprint" to emphasize that the design we produce should be a proved and tested design. To assure that, a lot of prototypes were built and tried before the design is written.

A. User Interface Prototypes

In the two-year experience with system X, one major complaint is the complexity and confusion of the user interface. The poor user interfaces make end users reluctant to use the system. On the other hand, during the analysis, we began to understand that school teachers (including elementary level and junior high level) are difficult and picky than we expect. First, the age of teachers can range from 20s to 60s, across 5 generations. Second, the majors of teachers can range from arts, literature, physical education, to science. As a result, their computer skills vary considerably. The vendor had a hard time in educating and training them.

To make future training and education easier, we conducted several user interface usability surveys. The user interface prototypes are designed based on a philosophy - treating the end users as if they knows little about computer. The major user interfaces are much wizard-based. Step-by-step wizard style user interface is constructed to smooth the learning curve. As to powerful users, they can turn on the so-called "power option" to avoid the step-by-step dumb user interface and play with powerful functions.

After the prototypes were tested, surveyed, these user interfaces are written into use cases to specifically restrict future user interfaces design. Note that UML standard does not specify the syntax and format of how to write a use case. In practice, a use case captures a contact between the stakeholders of a system about its behaviors. Use case are fundamentally a text form, although they can be written using flow charts, sequence charts, Petri nets, or programming languages. Under normal circumstances, they serve as a means of communication from one person to another, of among people with no special training. Simple text is, therefore, usually the best choice[3]. In our case, however, the use cases are primarily used to communicate from us to programmers. They are people with needed training. When writing plenty of use cases for the e-school system, in our opinion, using text to describe the requirements of user interfaces, their alternatives, or extensions are tedious. A prototype user interface is worth 100,000 words. Written requirements specifications trying to describe the look and feel of a user interface were nowhere near as effective as a user interface prototype or screenshot[1]. So, we boldly embed the prototyped user interfaces in the use cases. These use cases, however, are no longer neutral as a use case should be. They are actually part of the design.

B. Web service prototypes and interfaces

In this analysis, we choose web service as the technology for client applications to communicate with the server. In this work, the web services are simply used as a means to replace older techniques like remote procedure calls, CORBA, or DCOM. These web services will not be open to the public.

We built some prototypes to try out the .NET platform then write down the web service interfaces for developers to follow. A typical web service in .NET is similar to a procedure call like:

```
DataSet scoremgr_get_score(string school_id,
                           int semester_year,
                           int semester_no,
                           string login,
                           string passwd)
```

To make sure we do not miss any parameters, we often built a small prototype, tried the idea, and validated the results to see if we miss anything.

After the web service interfaces are clearly defined, we further explain its behaviors by sequence diagram like Fig. 5. In this sequence diagram, we explicitly tell the programmers the sequence of database queries and the tables involved. Following the diagram to implement a web service shouldn't be a difficult job.

VII. Discussions and Conclusions

In this paper, we describe a consulting work to tackle ML1 vendors in an e-school project. The approach is to construct a detailed design which serves as a contract and a software blueprint. This approach is a process specifically designed for a particular stakeholder (in this case, the *Y* county government), where we play the role as a consulting team, an architect team, and a design team to produce a software blueprint that guides and restricts ML1 vendors to implement the future system. The correctness and precision of the blueprint are validated by prototyping. We have built prototypes of different kinds to validate different problems.

Overall, this approach can be analoged to a typical process in other engineering discipline such as civil engineering. In the discipline, a building can be designed by an architect to produce a blueprint and then constructed by other teams. Such a way, of course, is not common for developing software but there are reasons that lead us to the approach.

After all, constructing a building and a software has fundamental difference in nature. A building is static. It does

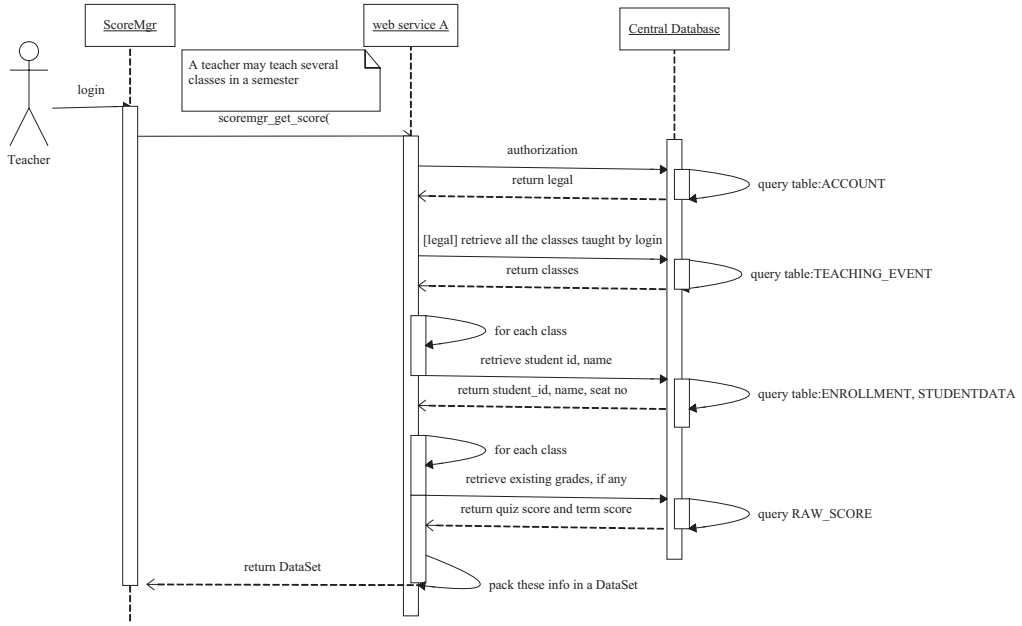

Fig. 5. The sequence diagram to describe the web service behaviors.

not have behaviors. A building's blueprint which often does not need much text to explain can be understood by arbitrary qualified civil engineers. In contrast, despite the success of UML, if a design is only expressed by UML diagrams, it is apparently not readable nor understandable. There are missing pieces. In this work, we supplement a lot of text, such as design rationale, the context, and etc. Such supplements, however, are up to the designer's writing skills. Besides, the problem of using text is that a reader can easily miss some critical points from large amount of text. These are the problems which makes software engineering so different from other engineering disciplines, and they are worth exploring.

The results of this work, obviously, is an opposite to the Agile method[10]. The manifesto of the agile alliance are:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

When there is value in the terms on the right, they value the items of the left more. They believe code is the best documentation. Here, we do not intend to compare our results with their beliefs, simply because the assumptions of our approach and theirs are totally different. Their process improvement is designed to impose on developers, whereas our approach is not. Nonetheless, we believe software processes is difficult to be unified. An appropriate software process must be chosen carefully or modified from existing ones to balance the weights among different factors.

REFERENCES

[1] B. Boehm. Anchoring the software process. *IEEE Software*, 13(4):73–82, July 1996.

[2] B. W. Boehm, A. Egyed, D. Port, A. Shah, J. Kwan, and R. J. Madachy. A stakeholder win-win approach to software engineering education. *Annals of Software Engineering*, 6:295–321, 1998.

[3] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2004.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[5] A. H. Eden and R. Kazman. Architecture, design, implementation. In *ICSE 2003*, pages 149–159, 2003.

[6] S. J. Greenspan, J. Mylopoulos, and A. Borgida. On formal requirements modeling languages: RML revisited. In *International Conference on Software Engineering*, pages 135–147, 1994.

[7] I. f. I. I. J. S. Ke, President. Software industry in taiwan. In *The Seventeenth International Conference on Software Engineering and Knowledge Engineering*, 2005.

[8] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[9] P. Krutchen. *The Rational Unified Process: An Introduction, 2nd ed.* Prentice Hall, 2000.

[10] R. C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2004.

[11] M. Weiss. Patterns for web applications. In *The 10th Pattern Languages of Programs (PLop 2003)*, 2003.