Software Configuration Visualization

Akram Salah

Computer Science Department
North Dakota State University
IACC 258
Fargo, ND 58105

Email: akram.salah@ndsu.nodak.edu
Tel: (701) 231 8555
Fax: (701) 231 8255

Abstract:

The paper overviews an experience in using configuration visualization software as part of teaching software engineering. The development of software as part of the class added many values to the teaching of the course. The lessons learned from the experience are shared in this paper. The software is not aiming at handling the configuration management, rather it provides a clear display of the configuration in a dynamic way that allows the project manager to know the current status of the project at anytime. The whole class handled the development as one development group with four teams each handles one subsystem. The communication among teams and within teams was organized to gain from the experience. The project is currently being studied for further development.

Software Configuration Visualization

1. Introduction

This paper overviews an experience in teaching a first course in software engineering. The course is taught on the undergraduate level. Most of the students are seniors. They already know programming and programming techniques. The objective of the course is to have the students aware of the software development as a process and as the concepts of how to think of software development as a step further from programming. Many concepts had to be covered in the course as definition and awareness of problems. Those concepts are covered in the classical class form.

However, there are many other issues that had to be realized about software development and engineering that are only acquired through practice. It is more on the skill level than on the concept level. Those are for example how to form teams? How to deal with differences on opinions? How to agree as a team on common choices? How to handle changes in requirements as the software progresses? How to do documentation as a team? Etc.

Many other concepts that had to be realized on the concepts that are acquired through practice, such as how much time should be devoted to meeting? How much effort is to be assigned to documentation? What are the kinds of problems that arise from integration of software? Those kinds of problems appear only if software is developed on a team basis rather than individuals.

In this class setting, a project was used for more than one reason, first, which is the typical project experience, to apply concepts taught in class and gain experience. However, the project topic was chosen as a problem in software engineering, which is configuration management, so students learn about the topic as they are trying to solve the problem. This provides a better learning experience. Moreover, the project was handled as one general project for the whole class, though the project is divided to four subsystems and each team is designing and implementing one subsystem. This allows a semi-independent environment in development where each team is making their own decision-making and solution development, yet there is a standard coordination of interfaces and styles of development that had to be coordinated on the group level.

Through the class, and the project development, there were planned presentations for the purpose of agreeing on a general standards, however, it served also the presentation abilities and the communications among teams. It also increased the critical thinking and discussions, since all of them are involved in one general problem to be solved.

At the end of the course, a report was required from each team for the purpose of self-assessment. However, it was presented to them as to develop a legacy report stated as " if we will start making the project now, what we had done that we would do again and what we would have changed?" The report should comment on two sides of the project,

one on the technical issues, such as language and environment used, the design methodology, the documentation standards, etc. The other side is on the organization of the teams, the meetings, the task assignments, etc. This would comment on the managerial issues.

The final presentation of the project was held in public with members of the faculty attending and invited software developers from Microsoft. The comments were very encouraging. The class expressed very positive comments on what they have learned from the class and the setting. The experience was overwhelming and sharing it makes more value to get other similar experiences.

The paper is organized as follows, in section 2, a short overview of the issue of configuration management. In section 3, a description of the project and the approach taken to tackle the problem, which was different from the other software. In section 4, some of the lessons learned are stated. Then, a conclusion and future work.

2. Software Configuration

A configuration of a product identifies the product's components, and also the specific versions of the components. Configuration management is the discipline of coordinating software development and controlling the change and evolution of software products and components. It is an old discipline that has traditionally been studied in the context of systems manufacturing. Its application to software, however, is more recent; in addition, software has special features that make it different from traditional manufacturing. Software adds complexity to configuration management, because changes occur much more frequently than in kinds of products. Conversely, software configuration management is more amenable to automation, since all items may be stored on media that are accessible to computers.

A couple of features characterize software from manufacturing products:

1) Software products are intangible and measurement is not physical, unlike parts of mechanical or electronic devices. Thus it is hard to identify a product from another by physical comparison.

2) Software components may differ even if they are functionally equivalent. For example the same software product for different hardware configuration may contain different components. To the user the differences are transparent. It is the responsibility of the development to identify various components for future changes.

3) Software is active in nature and it reflects as behaviors in various situations. The variety of behaviors is described at the execution time. Thus the measurement and performance varies for different situations.

4) Functionality and performance of software evolve as the uses of the software changes. Thus, the total functionality of the software is dynamic in nature.

5) Software is executable on hardware. Since hardware is evolving continuously, software has to change accordingly. These changes should be transparent to users though the configuration changes internally.

6) Software is not just code and programs. Supporting documents both technical documents such as requirements and design are part of the software. Also, user documents such as user manuals also part of the software and they also evolve as the software changes. They also have versions and are controlled with the code.

7) With reusability and the use of library routines, either as part of the language, the development environment, or the development team library, increases the complexity of the management and control. If a problem arises in one product, it may or may not appear in other software products if they share the same reusable code. In this situation, change is complex since the locality, or globality, of change depends on many factors.

The atomic element in configuration management is a configuration item. A configuration item can be a requirement statement, a design unit, a test case, a document, program code, or a unit of any of those. In this project we concentrate on code as the only configuration item, though the product is built, together with the database to be able to keep track on any configuration item.

3. Software Configuration Project

The main objective of the project is to develop software that acts as a medium of communication among software development team as well as between team and project manager. It is not directed to manage configuration, it is rather as an agent to facilitate communication and provide information to team members such that it would reduce meeting times both for team members who are working on various components of the same software or between team manager and members. Also, it provides a visual representation about the current status of the development process. It reflects the completion status of each class or components in color, as well as it represents the structure of the whole software. This allows the manager to assess the situation instantly and evaluate the status of the project compared to the plan. It would make decision-making and project control easy and direct. Third, it keeps data about all processes accomplished through the development. This adds a value for analysis of this data as the project is concluded.

3.1 Assumptions:

The project is based on a couple of assumptions. First, most of the work on configuration management usually starts as the software shipped to the customer. As the software needs maintenance, there is always a problem in getting back to the information

about components, who wrote the code, how it was tested, what are the documents associated with the software, and so on. The assumption here is that the process of configuration management should start with the development process and proceeds in parallel with the development. This would provide more accurate information about the software and its components. Which parts that has developed from scratch and which parts that has been used from a library. Even for libraries, which parts that has used from an in-house library and other parts from a language library or other wise.

Second, generally in software engineering projects there is a problem in lack of accurate data about projects that allow for analysis and assessment of the development activities. These data is important in project time and activity estimation in any project. Even though there are empirical data that help in the process of estimation, based on the techniques used, this data has to be adjusted to teams, types of products, the development environment, etc. So the collection of data is absolutely important. However, gathering the data, based on personal and manual estimations, proved its problems. This tool gather the data in an automatic way that decreases the load of filling the data on programmers and keeps them instantly in a soft form allowing analysis instantly as well as later as the projects is done.

Third, the awareness of programmers on their own style and performance in the process of development is necessary. This tool allows each team member to look at his/her own data to have a self-assessment and improve as the development process proceeds.

Additional assumption was adopted concerning the development phases. We assume that we use object-oriented approach. The system is composed out of a set of classes. A programmer is assigned one or more classes. Each class is constructed out of attributes and methods. The stages of development for each class is as follows:

Stage 1: Defining the class structure, that is the attribute (names and types) and method names, arguments, and types.

Stage 2: Implementing methods. Each method is implemented separately. A class in stage 2 if all methods are coded.

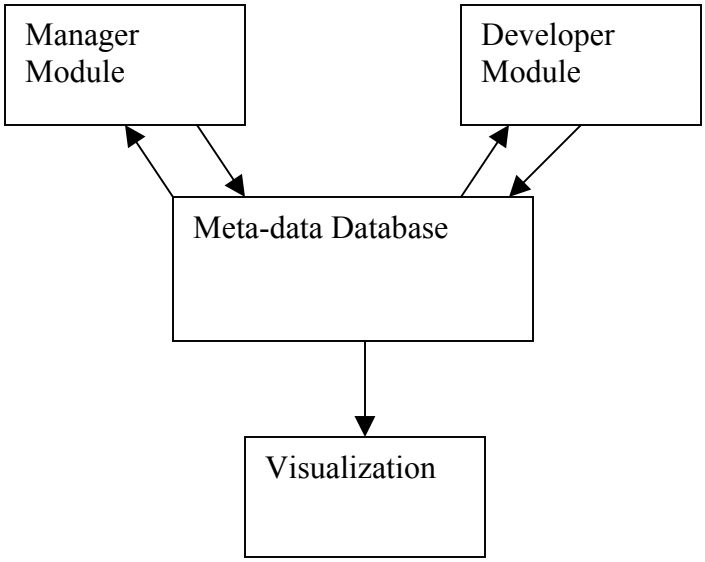
Stage 3: each method is tested as a unit testing separately. As the class is completed with the testing of all methods, it is in stage 3.

S*tage 4:* the class is tested as one integrated class.

For each class that has subclasses, it stage is determined by the lowest stage of all its components. That is, a complex class that one component that is in stage 2 is in stage 2 even if all other components are in stage 4.

3.2 Architecture

The project was presented to the class with a given architecture. The architecture is a repository style with a central database to keep meta-data about modules. It keeps structure as well as properties of components. Typically in this architecture, the communication is done through the database. In addition to the database, there are three major subsystems, a team member module, a manager module, and a visualization module.



3.3 Scope of modules

3.3.1 Developer module:

Each developer in the software development team accesses the developer module. The purpose of it is to display task information about classes assigned to this developer. The developer gets all the tasks assigned to him with a planned time for completion. As the developer works on his/her own assigned classes, he/she enters to the system the stage that he completes. So if he/she finished the method definition, he changes the class state to stage 1. As he completes the methods, he changes the state to stage 2, and so on. A developer may work on more than one class at a time as far as he/she keeps the ongoing activity that he works on. So if he switch working from one class to another, he has to tell the system that. The system keeps track on time spent on each class separately. The module retrieves and keeps information to and from the database.

3.3.2 Manager module:

The manager module is assumed to be used by the team manager. The manager defines to the system the structure and the components of the software as he/she starts working on a project. He can enter his estimated time to complete each part. The manager also has an access to the personnel database with as assigned list to his/her project. The manager assigns a person to each task and input that through the system. Again the manager module is communicating with the meta-data database. The database acts as a board of communication.

If a maintenance request comes to a project, that is an already developed piece of code is to be changed, it is entered to the system as a task and assigned to a developer as a task and it is calculated as a time consumption unit. However, if a developer decides to change a program of his own after he/she has finished development, he has to enter it to the system and he cannot work on it unless the manager approves the change.

3.3.3 Visualization module:

The visualization module is a graphical module that takes the data from the meta-data database and displays it in a color tree. The tree structure shows the class-subclass relationships. Each element is displayed in one of four colors reflecting the current status of the component. Since the tree structure of the software usually is complex, the system displays one level of a tree. As the user click a branch, the sub-branch of this element is displayed. So it is user controlled.

The manager or the developer can invoke this module. For the manager, it is assumed as part of his job to control the project and it is needed that he has an image of the status of the project with all components at all times. However, it is allowed to be seen by the developer since the dependency of the software components on each other is rather high. Thus, a developer may want to look at the structure to see the status of other components that are related to his/her wok.

3.4 Constraints

We assume some constraints to be adopted by the visualization software. They are all seemed logical.

1) Creating a new module or sub-tree is restricted to be done by the manager. This seems appropriate since the manager has the responsibility of maintaining the structure of the software. The manager inputs the initial structure as it is designed. Any change is assumed to be a change on design and thus has to be through the manager. Even if the developer wants to decompose his/her work into sub-components, he/she has to discuss it with the manager first. The manager then would add it.

2) Only the developer who is assigned the development of a component does a change in the status of this component.

3) In case of any change as a maintenance, that is a rewriting of code to add functionality or performance other than what has been assigned initially, has to be approved by the manager first.

4) The status of a composite component is assigned to the least of the status of its components. We started by assuming that we can derive a formula to calculate the completion of a composite from the completion ratio of the components, but we had to settle for the mentioned policy.

4. Lessons Learned from Project

This software had been developed in an undergraduate class for software engineering, Principles of Software Engineering. It is the first class on software engineering, and it was a new experience to the students with the process, working as a team and develops a real software size rather than examples or simple cases. There are many lessons learned from the experience, many of them were expressed in the final report for the students.

From the instructor point of view, it was an experiment to handle software product that is parallel to the development process that is taught in the course. The project was part of the educational process of software. The material covered in class was applied in the project. Many concepts about design, documentation, team meetings, agreements on standards, communication, etc. were practiced through the project. So the project was a case in which most of the concepts of software engineering were applied. This is part of the belief that many of the concepts are better realized through practice.

As the topic of configuration management itself, it is an important problem though the class time was not enough to cover advanced issues such as configuration management. The choice of the topic was aiming at covering the topic as part of the project with an appreciation for the effort that should be assigned to the management of components. As part of the project and class, a reading assignment was required on the issue of configuration management and the automated tools used in the market for it.

In the following part we discuss specific practices that contributed to the success of the project.

4.1 Team, class, and Project structure

The class had 22 students. The architecture of the project was decided previously and was presented to the class. The class was asked to form four groups, one for each subsystem. The group formation was up to the students based on their interest and skills (see next item). Each team has a designated coordinator. This allows each team to proceed in their requirements and design in a semi-independent way. The project documentation was group based, so each team has to, collectively, develop one document.

To proceed with this organization we had several levels of meetings. Each team has to meet discuss their subsystem and its requirements, design and implementation. They assign tasks and follow on them. Another meeting was done for the four coordinators to discuss issues that interact within the four teams, mainly the issues of interfaces between the subsystems and standards. Another meeting was for all the 22 students to discuss the progress and discuss problems if any. This meeting also was used for prototype presentation and agreement on standards.

It was noted that the dependency on the metadata and its access method was high. So the database team had to work their design first and present it to the whole class first. They also worked out modules to facilitate the access method and how to store and retrieve data. The other teams used this information through their implementation.

There was no restriction on the language or the platform used. The database team used a MySQL system that resided on the department server. All the documentation was residing on the same environment.

4.2 Two projects

As part of the course plan, the students were assigned a project on individual basis in the first week of classes. This project was to design software to produce a cover page for assignments. The requirements for this project, though it was vaguely expressed, include some graphics, data entry, using printers. Through the first week, the progress of this project was used to discuss why requirements have to be precise and how to make it clear. There was no standard format for documentation, though documentation was required.

Each student presented his/her software solution in a class session with discussions and questions on the details of the solution and how it handles variety of options. The discussion was very rich since each student has a solution and not all of them were alike. Most of the discussions were directed to appropriateness of the solution and its characteristics, rather than right and wrong.

This assignment of the first project was completed before the assignment of the configuration visualization project. The arrangement of two projects gave the class students a chance to know each other technically and through discussions of their software and development method. It also, indirectly, revealed language and platform orientation for the whole class. As we asked students to form teams, it led to a better organization of teams. Each one discovered some of his preferences compared to the class and they knew each other.

4.3 Meeting Room

The class meets for three sessions a week. It was planned that two sessions are dedicated to software engineering concepts and one meeting for project. At the start of the course,

there was a problem of student participation. They were hesitant to express their opinion and some were reluctant to comment on others work. So, I decided to change the setting. I reserved a meeting room and the project session was transferred to this room. It is not a change in plan, just a change in the room.

This change turned to be absolutely effective. Starting from the first meeting, students started to discuss issues, express themselves in a better way. Students with leadership capability start to stand out and were recognized by the class and myself. I added some items that were not in my plan originally on how to conduct meetings, agendas, discussion rules, etc.

4.4 Web site for documentation

As the project progresses, students start to feel the time required for meeting, on various levels, is high. We discussed the issue in a general meeting and some of the students came up with a couple of suggestions that have been implemented by the students.

1) Creating a mailing system for online discussions among teams on various levels. This reduced much of the meeting times a got the meetings to be more effective in decision-making.
2) We created a shared page for documentation. Each team would post their document for team and group communication.
3) The team coordinator meetings were art of this suggestions.

5. Conclusion & Future work

In this paper we overview an experience in teaching software engineering with a project to develop software for configuration visualization. The paper describes the project and the lessons learned from the practice. The project proved to be very successful in many ways. The students enjoyed the development. They were given the freedom to define their own requirements and design. The project was set as four interacting subsystems; each subsystem is handled by one team. Class presentations were used for communication. Meetings, using standards were a good contribution to the project. However, because of the limitations on time, the project concentrates on code only. The project with all its documentation was handed to a graduate student to study it further and do more detailed study of the problem. The graduate student participated in class as an assistant.

References

[1] Kramer J, NG K, Potts C, Whitehead K (1988) Tools Support for Requirements Analysis, Software engineering Journal, Vol. 3, No. 3 pp. 86-96.

[2] Ghezzi C, Jazayeri M, Mandrioli D. (2003) Fundamentals of Software Engineering, 2/e Prentice Hall.