A Pragmatic Approach for Software Maintenance Process

Sanjeev Kumar Punia
(Associate Professor)
IIMT College of Engineering, Gr. Noida

Anuj Kumar, Ph.D
(Professor)
Accurate Institute of Engineering & Technology, Gr Noida

Trilok Rawat
(Associate Professor)
IIMT College of Engineering, Gr. Noida

ABSTRACT

This paper describes the use of a process support tool that is used to collect metrics for upgrading our electronic retail system. The incremental prototype lifecycle approach is used in which each increment is categorized by an effort type and a project component. The different effort types used to span all phases of development are as acquire, build, comprehend and design. The project components include data and process models expressed by an object oriented modeling language and process algebra respectively. The components are build using C++ classes and function templates that include source and data files. This categorization is independent from incremental prototype approach and equally applicable to other software lifecycles also. The process support tool i.e. process wise integrator (PWI) ensure the consistency between models and C++ source code. It also supports the interaction between multiple developers and multiple metric collectors.

Keywords

Process, comprehend, object-oriented and process metrics

1. INTRODUCTION

The process modeling is very potential and powerful technology that may be utilized in order to understand the experiments and development process further. Rombach *et.al* [1] suggested that process technology may be enhanced by combining the process modeling and software measurement. Shepperd [2] stated that research work can be divided into different parts those may attempt to collect product metrics by using the process model as a framework. Phalp [3] find that the modeling technique is used to display the measurement of data and process by make a single graphical model that combine metrics and process models.

All these techniques use process terminology, phase activities or boundaries for structural data collection. However, till now only small work is reported for the study of software maintenance for uses this complementary discipline. This paper is based on the study of collecting maintenance data with respect to four independent categories. Perry *et.al* [4] use the same approach but they examination the process at much more detailed level and apply the information into a more generic framework. This paper contains five sections where section 2 describes its adoption and process maintenance. The description of the application is shown in sections 3. Section 4 describes the implementation method that is used to measure and maintain process and last section explains our data collection procedures followed by the result and analysis.
The work reported here is based on the incremental lifecycle prototype. The incremental phase defines the changes to be made in most of the existing systems that use incremental lifecycle prototype. The refinement of the prototype use rapid design repeatedly. Allman *et.al.* [5] suggests that the achieving long term goals are crucial for development. Generally long term goals are not fully understandable so that decomposition of long term goal into manageable short term goals is morale boosting progress for the developers. The actual coding of implementation starts after the furnishing of the prototype. The separate test phase is not included into the code while implementation includes the same test that is used with the prototypes. The compilers perform code checking and debugging during development to produce a sets of run time test.

The main feature of this project is to encourage the developers to develop experiments with their own and customer ideas. Therefore, many experiments were performed with prototype features and implementation language during development.

2. LITERATURE REVIEW

Parnas [6] explained that, software engineers are not fully trained for design change although second release was being evolved by same developers of the original system. Turner *et.al* [7] explained that the developers still find it difficult to produce a good design for a new system on the first try if they take a similar project.

They suggest that this problem may be practically solved by implementing a subset of the problem initially and then enhance the implementation iteratively till the complete solution of the problem. Additional, they stated that the skill and productivity level increases by using the same team for the successive implementation where constraints did not force to reuse the previous implementation in progress. A subjective assessment indicates that developers reuse approximately half of the old code due to temptation in producing new code by using new function and class methods. This code reuse the approach matches the philosophy of Allman *et.al* [5] as they stated that it is never too late to start new code and discard all existing old code.

Turner *et.al* [7] explained that the lifecycle concept of performing activities systematically support the idea of careful planning prior to machine access for the effective use of the expensive computer resources while some researchers argument shows that the lifecycle concept is unsuitable for the development of evolving systems today.

Curtis *et.al* [8] stated that the sequential views are not fully accountable for the important process attributes like iteration and feedback loops. They explained that the concept of conventional software lifecycle has been significantly altered with acknowledge of the prototype. Agresti *et.al* [9] challenges the assumption that the development follows a rigid sequence of activities from requirements specification of coding and testing. The lifecycle model offers a large grained view of the development process and cannot represent critical lower level details of a project. Kellner *et.al* [10] states that many smaller processes are overlooked in lifecycle description as processes may be examined in terms of a whole phase instead of multiple numbers of sub processes during the operation phase that give a less detailed view.

Often, the real software development processes do not consist distinct phases. Balzer *et.al* [11] argues that the software methodologies are unrealistic for separate specification from implementation. They claim that every specification is an implementation of some other high level specification so the partitioning of the development process into specification and implementation phase is completely arbitrary.

Due to these problems, many ideas of conventional software lifecycle model have been challenged and consequently largely rejected. However, many introduced terminologies still used and these terminologies reject the application of conventional lifecycle. They also addressed the software measurement and process modeling related problems. Rombach [12] stated that models and measures are inseparable and quality improvement plan requires measurably improved development processes.

3. METHODOLOGY

This paper involves developing a prototype process representation language for software processes specification. The application used for this study revolved around an electronic point of sale system developed by Greenwood *et.al* [13]. In last three years approximately 5K NCSL were developed. The data reported here belongs to release 2 of the electronic point of sale system which was developed two years ago. Thus the maintenance process deals with a large amount of legacy coding, maintaining and altering according to customer requirements.

The software models use the process of algebra and an object oriented modeling language for translation into C++ developed by Henderson [14]. We use a process wise integrator (PWI) evolution model that is developed by Parker *et.al* [15]. According to Greenwood [16], the project team play several roles in process wise integrator (PWI) model and two of them are explained below in brief.

1. **Developer's role:** It covers three types of main actions.
   - **a.** Effort actions: effort actions are used to measure the time spent by a developer by working on a particular component. The collected information is used to identify the name of specific component and the type of effort action performed. Here effort is being recorded for 1/4 of the day i.e. corresponds to 1.5 hours excluding break time.
   - **b.** Agrees actions: agree actions are used to find two components in a relationship those are agree with each other.
   - **c.** Change actions: change actions are used to correct the model supplied data. The user is able to indicate that a component has been changed without recording associated 1/4 day effort. All effort information is recorded by the developer after the expansion of the effort.

2. **Measurer's role**: It is shown by the metrics team that is responsible to extract the developer effort information from process wise integrator log. It covers two types of main actions
   - **a.** Modify action: modification is a default action included by process wise integrator which allows the role change possibility in the future.
   - **b.** Output effort log: output effort log is used to extract the effort log information and write to a standard text file.

According to Henderson *et.al* [17], the actions of the developer for recording purpose is based on the pumping model that is categorize into following four types.

- **a.** Acquire: acquire is used to acquire and customize the existing software that include the acquisition of other people's code and coding techniques from the literature.
- **b.** Build: built is used to code the low level modules, unit test and build the test harnesses.
- **c.** Comprehend: comprehend is used to understand the system that possibly involve literature surveys and experiment with hardware and software.
- **d.** Design: design is used to design high level models or platform software before coding and integration test planning.

4. ANALYSIS AND EVALUATION

4.1. Chronological analysis of developer activities

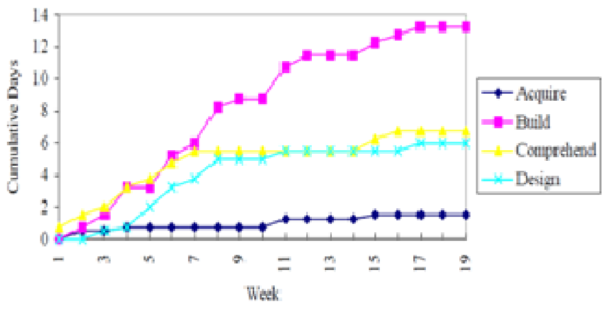

Figure 1

The graph of figure 1 shows the relation between times spent per task by the developer and component database. It shows that most time consuming activity of comprehension is performed in the initial stages of the project. The developers are completely concentrated to understand the activity during the first week of the project. During the first few weeks i.e. weeks 2 to 4, the developer spent a small amount of time to acquire and customize the existing software and tools those are to be use for experiment with coding techniques.

Figure 1 also shows, that the relationship between the cumulative time spent during design and the week number is almost linear between weeks 2 to 8. As suggested by Curtis *et.al* [8], the design time spent can be accounted by team meetings. In our case, the developers met to exchange information and to discuss the shared process support details. However, the design of the project was seems to be finalize from week 9 onwards while small amounts of time spent for adding extra functionality to the design in weeks 11 to 17 and the developer's efforts were almost solely concentrated on coding. After that almost all the available time spent on anti regressive activities such as the code restructure and the documentation updating.

As Gersick [18] suggested, that there is a critical point halfway by a project group where the team comes to a consensus in progress. This may be reflected by the delivery of the design and coding effort concentration. The flatter section of the graph indicate consolidation period in the project. After completely evaluating the task around week 7 there is a need of occasional couple time period to understand the project in weeks 15 and 16. Figure 1 also shows that the cumulative time spent for the developer activity like time and build is almost linear throughout the duration of the project.

4.2. Chronological analysis of the project database

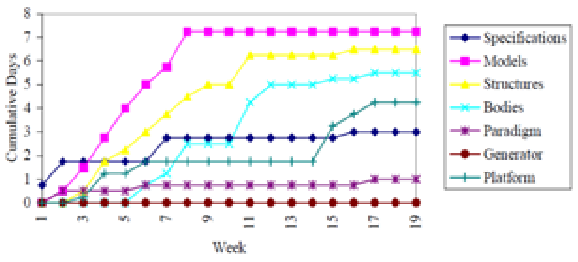

Figure 2

The graph of figure 2 shows the relation of the cumulative time spent for each types of the component in the database. The graph shows that the relationship between modeling cumulative time and week number is fairly linear during early stage in the projects. From week 9 onwards, only small time is given to modeling and subsequent effort was concentrated on coding.

This effort is captured by the code structures and code body components for the second and third largest proportions of developer time. Although no coding was done during the first week of the project but week 2 onwards the relationship between the cumulative times spent on code interfaces and the project week number is fairly linear. At this time, the developers worked on the overall structure of the C++ classes those are used to implement the models.
The production of code for these classes implementation is initiated much later in week 6. Figure 2 also illustrates that the majority of initial project effort was applied to the specification during the working of two activity periods i.e. in weeks 1 and 2 and in week 7 respectively. Again, from the critical point of week 8 onwards, the specification became finalized and all developer effort was channeled towards the implementation.

For developers, only a small time period is accounted for platform and paradigm early in the project. These two were associated with acquiring and comprehending the workings of existing tools and code. Figure 2 shows that no time is spent for the generator components at all as the developers did not construct any tools for aiding the model conversion into code.

5. CONCLUSIONS

Here we have collect the process data for each category in the process model by combined process models and process metrics in recent work. The collection of data effort against four independent categories as acquire, build, comprehend and design represents a departure from the orthodoxy. We conclude that generally, a large amount of the developer's time was devoted to relate tasks legacy products comprehend from previous releases initially. This leads us to suggest that project effort could be reduced by supplying additional documentation that give more details about the software functionality.

All of this abstract information activity provides a distinct and orthogonal view of the developer's personal process. For example, we can see how comprehend the existing system spans a number of process activities or phases. This approach provides us a much more detailed picture of the process than data collection against the process model categories that is independent for the underlying process model. Hence this process change invariantly and is applicable to all other situations irrespective of the project's process.

Although our study is on a relatively small scale but our preliminary findings suggest that such an approach gives us a far greater understanding of the software development process than traditional approaches which provide an activity based view only. Further work is continuing with different developers and large scale projects both to learn more about the nature of industrial software development and the applicability of our method.

6. REFERENCES

[1] Rombach and Pfleeger, "Measurement based process improvement", IEEE Software, 2004.

[2] Shepperd, "Quantitative approaches to process modeling", Colloq. on Process Planning and Modeling, London, 2002.

[3] Phalp, "An investigation of process modeling in practice", Ph.D. Thesis, Bournemouth University, UK, pg. 107 - 109, 2005.

[4] Perry and Staudenmayer, "People, organizations and process improvement", IEEE Software, 2008.

[5] Allman and Stonebraker, "Observations of the evolution of a software system", IEEE Computer, pg. 27 - 32, 2002.

[6] Parnas, "Designing software for ease of extension and contraction", IEEE Transactions on Software Engineering, 2009.

[7] Turner and Basili, "Iterative enhancement: A practical technique for software development", IEEE Transactions on Software Engineering, pg. 390 - 396, 1998.

[8] Curtis, Elam and Walz, "Study the process of software design teams", 5th Software Process Workshop, Kennebunkport, Maine, USA, pg. 52 - 53, 2009.

[9] Agresti, "The conventional software life-cycle: Its evolution and assumptions", IEEE Computer Society Press 2008.

[10] Kellner and Curtis, “Process modeling”, Communications of the ACM, pg. 75- 90, 2010.

[11] Balzer and Swartout, “On the inevitable intertwining of specification and implementation”, Communications of the ACM, pg. 438 - 440, 2009.

[12] Rombach, “Design measurement some lessons learned”, IEEE Software, 2010.

[13] Greenwood and Warboys, “Co-operating evolving components a rigorous approach to evolving large software systems”, Proceedings of the18th International Conference on Software Engineering, 2006.

[14] Henderson, “Object Oriented Specification and Design with C++”, McGraw-Hill, 2003.

[15] Parker, Bruynooghe, Butler, Hook, Cook and Greenwood, “Process wise Integrator: Sun hosted system”, ICL, 2005.

[16] Greenwood, “EPOS evolution process wise integrator”, tech. rep., Department of Computer Science, University of Manchester, UK, 1995.

[17] Henderson and Warboys, “Configuration description for component reuse”, 1st International Workshop on Software Reuse, Dortmund, Germany, 2001.

[18] Gersick, “Time and transition in work teams: Toward a new model of work development”, Academy of Management Journal, pg. 9 - 41, 2008.