A Tale of Two Projects

Dewayne E Perry
Advanced Research in Software Engineering (ARiSE)
Electrical and Computer Engineering*)*
The University of Texas at Austin
perry@ece.utexas.edu

***Abstract*— The structure of an organization, the processes, and technologies used have a significant effect on software developments. The hypotheses explored here are that 1) an organization is not independent of process, and 2) process is not independent of technology. The two projects described remarkably display the same strong trends even though they were done in very different parts of the business, in different kinds of software developments, and in different geographic locations. Both projects had an organizational structure of strongly empowered teams, understood the technical problems and their solutions at a fundamental level (a good match of core competencies and expert knowledge of problem), and used some innovative software engineering technology.**

***Index Terms*—Project Organization, Process Improvement.**

I. Introduction

AT&T found itself in a world that is much more dynamic in its demands than had been true in the past. The communications marketplace had become a highly competitive one with both demands from customers and pressures from alternative suppliers. There are two important problems that arose in this situation: 1) how do you change a well-established project structure, and 2) what do you change it to?

To gain an understanding of these problems, we investigated how two experimental development projects, in very different environments and organizations within the same company, dramatically improved time to market, and slightly improved cost and quality. Our analysis leads us to conclude that these experimental projects were successful because they recognized that organizational structure, process, and technology are interdependent and must all be manipulated to optimize time to market, cost, and quality.

In the following introductory sections, we describe a model of software projects, delineate our hypotheses, and discuss the project selection criteria and project variables. We then discuss two projects in two sections each: first, we give an overview of the standard process and organizational structure; and second, we discuss the experimental project structure, its rational and the results. Finally, we summarize what we consider to be the important aspects of these two experimental projects and how they support our hypotheses.

A. Models of Software Projects

Traditionally, software projects have been viewed in terms of the waterfall model [6]. This model provides a standard structure for the software artifacts and a standard sequence of activities for the project. More recently, software projects have been considered in terms of the spiral model [2]. This model is particularly effective in providing a structure for incrementally reducing various risks in building software systems, especially large and complex ones.

In both of these models the emphasis is primarily on the software process with little attention to organizational or technological issues. It is our contention that these issues are equally important and that they have not been given sufficient attention. We therefore propose a model of software development projects that has three components:

- Organization,
- Process, and
- Technology.

The organization component defines the management and organizational structure of the project. The process component defines activities, transformations, dependencies and interactions that take place in producing the software artifacts. The technology component defines the technical aspects of the artifacts and the tools that are applied to them.

B. Hypotheses

We have two hypotheses that we want to demonstrate in this study:

- Organization is not independent of process, and
- Process is not independent of technology.

Clearly, one can imagine cases where one might consider organizational issues separately from process issues. We believe, however, that in the default case, the two are coupled together and should be considered as interrelated and interdependent. Similarly, there are levels of abstraction where aspects of process are independent of particular technologies. In general, however, the two are coupled and should also be considered as interrelated and interdependent.

We show the validity of our claims in the discussions of the two experiments below.

C. Project Selection Criteria

We have selected these two projects for the following reasons:

- They have executed a complete cycle of development.
- They have well-documented post mortems of their experience.

- They have quantitative data about interval, quality and cost.

Despite the richness of the experiments, we advise some caution for the following reasons: these are not controlled experiments and some of the results could be partially due to the well-known “change effects” or “being watched effects”.

D. Project Variables

There are three interrelated, macroscopic variables by which we measure product development projects: cost, quality, and time interval. There are a wide variety of ways in which these variables can be optimized in any particular development project. We will not discuss this general problem, but instead focus on only one aspect for the purposes of this analysis – namely, time interval.

The time interval is a function of how you view the artifact and break it up into pieces of work, how long it takes for each piece of work to get done, and the ability to proceed concurrently on different pieces of work.

There are four reasons why focusing on the time interval variable is sufficient. First, reduction in the interval enables a project to give a faster response to customer needs. Second, in any established process that has evolved over time, there are inefficiencies that have accumulated in that process. Pushing on the time interval is a useful way of uncovering those inefficiencies. [1] Third, while there are parts of the process that are necessarily independent of the particular value added to the customer, there are some aspects which provide no such value. Finally, a reduced interval by definition has less partially worked inventory and thus reduced carrying costs.

E. General Analysis Methodology

In our analysis, we emphasize the contrast between the prevailing development methodology and the one implemented in each of the case studies. As previously stated, we feel much is lost by trying to exclude the "adopted organizational structure" which has become the operating standard. Therefore, we characterize each organization as well.

II. The Y0 Packet Feature Development

The Y0 Packet Features Development (Y0FD) is composed of four features:

- ISDN Packet Business Group (IPBG)
- Conditional Notification & Channel Selection (CNCS)
- Packet Trunk Interface Standard (PTIS)
- Packet Multi Line Hunt Group (MLHG)

We have summarized in Table 1 the size of these features both by source code developed and technical head count years of effort expended (THCY) [5]. These numbers are estimates created during the initial development planning. The numbers are well within 5ESS norms.

The challenge to the development team was to reduce the development interval from 16 months to 12 months, while maintaining or slightly improving the quality of the product. The fault density, as delivered to the first customer (Q2), is the measure of the product quality. The goal for this development was .23 faults/KNCSL.

	KNCSL	THC	Faults/Q2
IPBG	12.8	9.3	3
CNCS	19.1	13.9	4
PTIS	14	10.2	3
MLHG	8.8	10.2	2
Total	54.6	39.7	12

Table 1 – Size of Y0 Packet Features

All code sizes are thousands of noncommentary source lines (KNCSL). The staff is in technical head count years (THC) and is the effort integrated over time from Q10 through Q2. Faults remaining are the number at delivery to the first customer.

A. Standard Development

The 5ESS software development process is an adaptation of the standard waterfall model (see Colson and Prell's paper on 5ESS projects [3]). We present a frame of reference by describing the relationship between process, organization, and product. Further, we identify the crucial dynamics of the current process: many formal *handoffs*[2] and quantized monthly intervals. [3]

Milestones	Definition	time (Ms)
Q10	FSD Complete	
Q9	Requirements DS	3
Q8	Design DS	3
Q7	DU DS. Coding, DU Test	4 to 6
Q6	Capability Test Pass	2 to 3
Q5	Fot Test Pass	3
Q2	SV/FOA Complete	4 to 5

Table 2 – Standard Development

We have listed the standard 5ESS development template of development milestone labels, definition, and typical interval in months. For historical reasons Q4 and Q3 are milestones tracking internal System Verification/First Office Application (SV/FOA) events and are not relevant here.

The standard development results from an assembly line like approach to developing software. At each stage a major

[1] It is considered common wisdom that reducing production intervals is always a good thing to do. Certainly the Boston Consulting Group would have you believe this to be true [7]. However, there is no experimental basis for their position; interval reduction is argued on a priori grounds. We note that there is mounting evidence that sometimes short term gains will not outweigh long term costs [8].

[2] A *handoff* is defined as propagation of a deliverable between individuals

[3] The minimum unit of time that any task duration estimate can be given in.

milestone is defined (see Table 2). Historically, as the 5ESS project's process and product matured, the throughput was increased by making one organization responsible for each stage. This can maximize throughput, but at the cost of many handoffs which are costly in time and difficult to coordinate. The cost of handoffs is the result of many different groups having to relearn some detail information before they can start their work. The coordination cost results directly from different pieces of the organization having to wait because of unexpected delays and their inability to efficiently schedule around them.

The other key idea is the need to have a basic, standard planning time unit for task duration. In 5ESS this unit is one month. The need for the standard unit is a direct result of the tight coordination required to control interval with many organizational handoffs.

The unit is determined by competing forces that are complex. The first force is the minimum task or subtask time interval on the assembly line. The second force is the minimum interval management seeks to have controlled. Opposing the desire for management to be infinitely precise is the third force--the cost of tracking these tasks and subtasks. The smaller the unit of time, the more sampling, hence, the higher the cost of tracking. These forces balance at roughly 1 month for 5ESS.

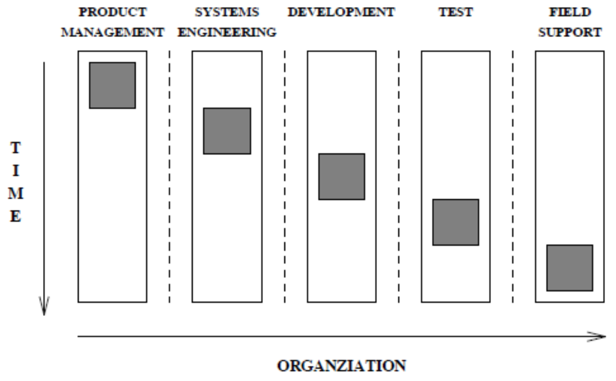

Figure 1 – Standard Development

This figure depicts the value added by the different organizations as the product is built, starting as a business case and completing as a product. The vertical axis portrays time – so as the developing product is built it moves forward (down) in time and across (left to right) through different organizations. The grayed squares inside the different organizational rectangles crudely show the critical path of the product development cycle.

B. Y0 Development

The Y0 development process alters two of these factors: 1) the many formal *handoffs* and 2) quantized monthly intervals. Instead of a functional organization approach, a team approach is used to minimize handoffs. This solution mitigates the monthly intervals, as well, because the team does not need as much formal review so management can reallocate resources. The milestones can be more naturally matched to the structure of the Y0 features and the team's talents.

Matching team milestones to the feature allows the team to exploit characteristics of the problem making the entire development less prone to fault insertion. For instance, in the Y0 feature development many difficult fault recovery scenarios of all the features are designed by the same expert. A direct result of exploiting the structure of the organization, process, and software engineering.

Figures 1 and 2 summarize the differences between standard and Y0 development. The interested reader is directed to [5] for further details of the Y0 feature development.

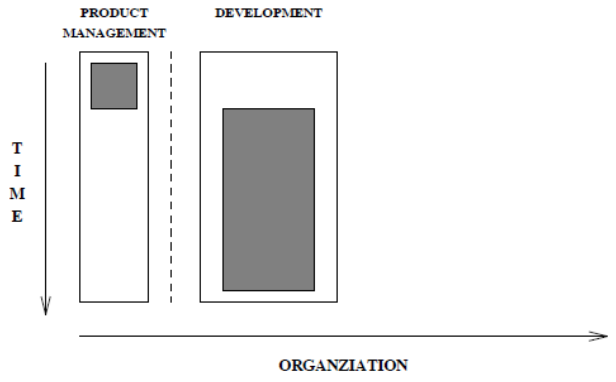

Figure 2 – Y0 Development

In contrast to Figure 1, we see the benefit of a shorten time interval since there are fewer hand offs between organizations. Glass and Sanders in their *AT&T Technical Journal* [4] in 1992 article observed the same phenomena for hardware development processes.

III. The FNMS-R3 Software Development

The FNMS-R3 software development [9] was an enhanced release of approximately 45 thousand non-commentary source lines (KNCSL) written in C++ on a base of around 140KNCSL undertaken by about 25 people. This enhancement consisted of three major features and a number of minor features.

The previous release (FNMS-R2) took about 16 months to complete. In general, the process was too unresponsive to customer needs and the products were too unstable in the field.

The goal of the new development process was to enable the team to shrink the overall cycle time, to improve the overall quality by removing defects early in the process, and to decouple the features from each other so that high priority features could be delivered as early as possible.

A. Standard Development

Prior to the current experimental process, there was no formal process in place for the development of features. The development was pretty much schedule-driven. That is, a development schedule was mapped out and used as the management plan directing the development process. The general intent of the schedule-driven process was to support incremental development (five major releases were planned, but the actual number of releases was much higher, primarily due to fixing problems, etc.). The system was cutover two months late.

Except for a one day high level design review and an external architecture review of the FNMS-R1 architecture, there were no design reviews or code inspections. Moreover, there was no formal unit testing and only minimal integration testing (with no clear exit criteria). Documentation was done after the fact – while the product was being soaked at a field site.

The development was organized along functional lines: systems engineering, development, and system test. Problems that arose from the separation of these functions included:

- interface problems between systems engineering and development;
- lack of support in reviewing requirements (which were late and constantly changing) in a timely manner;
- an inactive MR review board – status meetings were reduced to fighting fires and managing crises.

B. FNMS-R3 Development

The FNMS-R3 development process altered three things in order to achieve their goals. First, they added some standard quality gate techniques: design reviews, code inspections, etc. Second, they decreased interval time by decoupling features that could be developed in parallel and by changing from a functional organization to an interdisciplinary team organization. In order to make them responsive and able to deliver their products as quickly as possible, the teams were empowered to be responsible for their feature from feature specification through integration of these features into the existing system. Third, within the individual feature developments, team members were encouraged to do as much in parallel as possible.

The results of these changes were as follows:

- The cycle time was reduced by about 25% to 12 months (despite the learning curve associated with installing a new process).
- Decoupling features enabled short features to be implemented and delivered very quickly. One of the major features was delivered three months ahead of the other two features.
- Defects were removed earlier with very few problems encountered after integration testing.
- The team organization increased the effectiveness of the development process with team members assuming various roles that were previously in different functions. Moreover, the team approach significantly increased the effectiveness of communication among team members.

Thus, both the development interval and the product quality were increased by effectively exploiting the structure of the organization and product, and introducing sound software engineering techniques.

IV. Observations & Conclusions

A thorough reading of the case studies and referenced material should draw the reader to the same observations that we have made. Both projects remarkably display the same strong trends even though they were done in very different parts of the business, in different kinds of software developments, and in different geographic locations. Both projects had an organizational structure of strongly empowered teams, understood the technical problems and their solutions at a fundamental level (a good match of core competencies and expert knowledge of problem), and used some innovative software engineering technology.

Further, we note that both projects displayed strong conviction in execution fundamentals. What they said they were doing, they were doing. It is hard enough to improve a process, much less trying to do it when the state is unknown.

A final caution is in order. All good ideas can be applied in such a way as to not provide the expected result. We note that fundamentals are important and should be emphasized and that many of these steps discussed in the preceding paragraphs are intimately related to each other. Although it is difficult to prove in a mathematical sense, we believe that organization is an integral part of process and cannot be separated from process. Moreover, we believe that attempts by organizations to apply the development map ideas described here without consciously adopting the required organizational structures will lead to failure.

References

[1] Thomas J. Allen and Oscar Hauptman. “The Influence of Communication Technologies on Organizational Structure”, Communication Research 14:5 (October 1987), pp 575-587.

[2] Barry Boehm. “A Spiral Model of Software Development and Enhancement”, IEEE Computer, 21 (May 1988), pp 61-72.

[3] Joseph S. Colson, Jr., and Edward M. Prell. “Total Quality Management for a Large Software Project”, AT&T Technical Journal, 71:3 (May/June 1992), pp 48-56.

[4] Kathleen K. Glass and Lucinda M. Sanders, “Managing Organizational Handoffs with Empowered Teams”, AT&T Technical Journal, 71:3 (May/June 1992), pp 22-30.

[5] J. D. Huang, T. E. MacGregor, K. A. Radtke, and D. J. Rajkarne. “Software Development Process for Y0 - Packet Features”, Memorandum for File, 15 August 1991.

[6] Winston Royce. Managing the Development of Large Software Systems”, IEEE WESCON Proceedings, August 1970, pp 1-9. Reprinted in Proceedings of the 9th International Conference on Software Engineering, Monterey CA, March 1987, pp 328-338.

[7] George Stalk, Jr. and Thomas Hout. Competing Against Time: How Time-Based Competition is Reshaping Global Markets. New York: The Free Press, 1990.

[8] George Stalk, Jr. and Alan M. Weber. “Japan's Dark Side of Time”, Harvard Business Review, July-August 1993, pp 93-102.

[9] H. T. Yeh. “Re-Engineering a Software Development Process for Fast Delivery - Approach & Experiences'”, Proceedings of the First International Conference on the Software Process: Manufacturing Complex Processes, Redondo Beach, CA, October 1991, pp106-111