Case Study

Empirical studies in software development projects: Field survey and OS/400 study

Dien D. Phan a, Douglas R. Vogel b, Jay F. Nunamaker, Jr. c

a Department of Business Computer Information Systems, College of Business. St. Cloud State University. St. Cloud MN 56301 (612) 255-2174 USA

b Department of Management Information Systems, College of Business and Public Administration. University of Arizona. Tucson. AZ 85721 (602) 621-2748, USA

c Centre for Management of Information. College of Business and Public Administration University of Arizona. Tucson. AZ 85721 (602) 621-2748, USA

Abstract

Despite ongoing efforts to enhance the processes and techniques used in the development of software projects at all stages, software development projects continue to suffer problems in meeting user expectations, schedule, and budget. The purpose of this paper is to address the issues of management and control in large development projects and to present the results of our independent study on the development of the OS/400 R.1 development project, a very large software development project at IBM Corporation. Rochester. MN. The results of a field survey of software development professionals are summarized and compared with those of the OS/400 development. Furthermore, experience gained from the OS/360 development project is revisited and new insights are discussed. The paper concludes with lessons learned and project success factors.

Keywords: Software project management: Software engineering OS/400: Waterfall process: Parallel development: Development collaboration: Software tools: Reusable code

I. Introduction

1.1. Background

Despite continuous efforts to improve management and control, problems continue to exist in today’s software development projects. Chief among the reasons suggested as the cause of this situation is that we have failed to apply effective and efficient strategies and techniques to control resources, scheduling, and requirements in software projects. Although a reasonable amount of research has demonstrated that improvements in software engineering techniques and processes increase the chances for successful delivery of software projects, few published studies have been related to the social, technical, and managerial aspects of complex and large-scale development projects.

1.2. Goals and plan of study

The purpose of this study was to gain insights regarding factors that contribute to the successful delivery of large projects. The goals of this study were to determine:

- What major problems and opportunities in the development of large, complex software projects need more attention?
What are the critical success factors for meeting user requirements and producing software of good quality while staying within schedule and budget?

What new insights and lessons have been learned? Specifically, what are the implications of such insights as “Adding manpower to a late software project” “Chief programming team”, and Plan to throw one away” that Brooks discussed in connection with experiences gained through OS/360 system development[4]?

In order to address these concerns, in-depth research on management of software development projects was conducted at the University of Arizona from 1987 to 1990. Empirical studies included a survey of software development professionals and a case study of the development of the OS/400 software.

2. Field survey of software professionals

Our study of software project management began in 1987 at the University of Arizona with a survey of practicing computer professionals to determine their views regarding the planning, control, and management of medium and large-scale software projects [101. After a field test with software development staff of the Computer Centre at the University of Arizona, the survey questionnaires were sent to 827 randomly selected members of the Association of the Institute for Certification of Computer Professionals (AICCP).

Respondents were asked to comment on the following topics: (1) general characteristics of software project development, (2) definition of project problems/opportunities, (3) characteristics of project planning and control, (4) staffing and management of human resources, (5) the effects of user feedback on project design and development, (6) reasons for cost overruns, (7) causes of project delays, (8) use of project management tools/techniques, (9) actions taken to handle delayed projects, (10) strategies employed to coordinate and control resources, and (11) total project efforts. There were 191 responses, a 23% return rate. A screening process to eliminate responses from small development projects yielded 143 qualified responses. Respondents included project leaders and managers (47%), technical professionals (programmers, analysts, and consultants) (37%), and top executives (16%). Most respondents had at least five years of field experience, in keeping with AICCP membership qualification procedures. Some 56% of the projects were carried out by a single team led by a team leader and 44% were run by multiple teams. Initial planning of development projects was most often the responsibility of the project manager or leader (50.7%); it was less often done by an ad-hoc planning group (34.7%) or steering committee (20.3%) and was rarely the responsibility of a planning department (9.4%). These percentages total more than 100% because planning responsibilities in some projects were shared.

In analyzing survey responses, the ANOVA and Spearman rank correlation tests were used. ANOVA analysis was used to test the significant differences in the responses of three groups of respondents (technical leader-manager and executive). If there were significant differences among the responses of the three groups, separate analysis of answers to the pertinent questions was performed for each group. Responses were also tested for correlation with the project success indicators (i.e., meeting user requirements, suffering cost overruns, and suffering late delivery) using the Spearman Rank Correlation test at 5 percent significance level. Based on the correlation strength resulted, project characteristics and/or factors were tallied, ranked, and analyzed.
Although software project management historically has been notorious for cost overruns, late deliveries, and failure to meet user expectations. the results of this survey indicated that progress had been made. A significant sign of improvement was the high proportion, nearly three fourths, of projects that were considered to have met user requirements. Nevertheless, problems in managing software development projects remained. Chief among these was difficulty in the estimation of project costs, resources and scheduling requirements. The survey responses also indicated that efforts and resources frequently had not been properly allocated to various tasks. Respondents indicated that they would have liked to have put more effort into front-end work, such as planning, feasibility studies, and documentation, and less effort into design, development, and debugging. They noted that software project teams lacked political power to deal with external organizational units.

Responses to nine of the total of 100 questions analyzed by ANOVA showed some significant differences from one respondent group to another. Further frequency analysis of responses to these questions by group revealed some interesting behaviours and patterns among IS groups in organizations. On the one hand, the executives and leader-managers indicated that they relied more on software tools, especially PERT/CPM, Gantt, and data base tools, than did members of the technical professional group (programmers, analysts, and consultants). On the other hand, both the executive and leader-manager groups tended to have a more positive view of the management of projects than the technical professional group. For example, a good percentage of executives and leaders-managers perceived that project leaders possessed excellent technical skills (68.8% and 34.7%, respectively), much higher than the percentage so perceived by technical people (15.8%). Similarly, more executives (66.7%) than leaders and managers (37.3%) or technical people (28.2%) felt that project problems were usually well defined.

Overall, the survey revealed that respondents shared similar opinions about causes of project failure and they recommended similar actions to improve project success in terms of meeting schedule, budget, and user expectations. Common project characteristics and/or factors, listed by order of correlation strength with success indicators, are:

1. All feedback from end-users should be studied promptly.
2. Major design changes should be minimized during development. Unless they were critical, these changes should be postponed to the next version.
3. Communication and coordination problems should be minimized.
4. Project leaders must possess good management skills.
5. Staff turnover should be minimal.
6. Staffing must be well balanced among projects and subprojects.
7. Feedback from the environment must be communicated to project team members immediately for corrective actions.
8. Quality assurance efforts should not be reduced to meet schedule and budget.
9. Compliance with customers’ requirements should be honoured. i.e., no attempt should be made to persuade customers to accept less desirable features.
10. A good interface to user and environment must be maintained.
11. Project problems, goals and objectives must be clearly defined.
12. A sufficient numbers of specialists should be available to work on the development team.
13. Steering committees and user advisory groups should be used effectively.
3. **OS/400 development study**

To gain more specific insights into management strategies and practices of a typical project that is large and complex, a case study of the development efforts of the OS/400 was conducted by the authors at the IBM development laboratory in Rochester, Minnesota. It is estimated that over 50 million lines of code, including code used in testing modules, test files, and test data, were written for this project by the development team. This voluminous code was then screened and refined to become the 3.6 million lines of code included in the final product.

The OS/400 development was part of the Silverlake Project initiated in mid-1986 to develop a new generation of midrange computers, the AS/400. IBM hoped that these would be the replacement for its matured Systems 36, Systems 38, and 4300 series computers. The OS/400 project was a large-scale effort to develop the operating system for the new computer family. For IBM, the early 1980s had been tumultuous years during which the company lost a great deal of its mid-range computer market share to its archival Digital Equipment Corporation. IBM’s 9370 mini-computer line, expected to be the ‘Vax Killer’ when introduced a year earlier, had failed to accomplish this goal. Worse still, an earlier ambitious attempt to build a new machine, code-named Fort Knox, had been abandoned before completion in 1985 [2].

Faced with a continuing need to develop a successor for its Systems 36 and 38, IBM studied the feasibility of merging the two product lines. A project proposal was approved in February 1986 and top executives were appointed to lead the project. In May of 1986, the entire Rochester laboratory was reorganized to carry out the mission. By August 1988, after 26 months of writing and refining millions of lines of new code and over a million lines of reused code, the project was completed on time, within budget, and in conformance with user requirements. The successful delivery helped IBM to overcome tough challenges and reverse downward trends for its entire mid-range computer market.

3.1. **Development strategies**

The goal of the OS/400 development project was to respond quickly to increasing demands for modern business systems. The continuous stream of innovations in the computer industry had resulted in tremendous improvement: more power and higher quality at lower cost. However, while payoffs for some niches were very high, many of these products were obsolete before they reached the market. With limited time and resources, the challenge was to identify and seize the opportunity to satisfy the plethora of customer needs that had sprung up in the mid-range computer marketplace. The key elements in project planning were the setting of goals and priorities [9]. Strategies used in the successful delivery of this project included refining requirements and controlling market demands, managing and controlling environment and resources, developing components in parallel, utilizing efficient tools, and controlling software quality.

**Management and control of requirements**

In order to establish solid project goals and objectives, a careful study of customer requirements and market opportunities was undertaken, beginning with customer surveys and literature searches. Customer requirements were grouped and converted into project requirements, which were ranked according to their relative importance. The resources required to develop key requirements were then estimated.
Defining and implementing resource control strategies

Fulfilling customer requirements in order to seize a market niche in a timely manner while constrained by limited resources was exemplified by Frederick Brooks’s analogy of putting “ten pounds in a five pound sack”. In addition, various studies [1], [5], [11] found that in a situation involving schedule and resource compression, quality assurance efforts may suffer. However, thanks to having been given great autonomy, the management was able to circumvent this. Deviating from traditional IBM culture and structural inertia, IBM headquarters gave IBM Rochester autonomy to manage its market demands and to control resources by employing such strategies as innovative human resources management, collaboration, coordination, merging and spreading of workload, reorganization, and contracting for programming services.

Managing human resources. In 1986-1987 IBM was restructuring its workforce in response to fierce competition and a sluggish DP market. This meant that it was a very difficult time in which to staff the newly launched, large-scale Silverlake Project. Many employees had been transferred to marketing and customer support functions; some others had accepted early retirement. To gain necessary technical skills, the project had to hire additional programmers, transfer workers from other projects, retrain non-programming employees, and hire additional tempo-raw workers. The project also employed tempo-raw workers, student interns, contract programmers, and service vendors. Positive personnel management practices included fair rewards, room for advancement through dual career opportunities and a tail hierarchy, open communications, and various other motivation techniques. As a result, the personnel turnover rate remained much lower than that of high-tech industry in general. despite schedule pressures with much overtime.

Coordination, merging, reorganization and linkage.

Since technical expertise played so vital a role, continuous participation of technical experts in every phase was encouraged through coordination groups! such as joint committees, task forces, and system Design Control Groups (DCC) at system, functional, and departmental levels. The DCC group was made up of highly-skilled software designers and architects who coordinated technical design for the development teams. The primary focus was on providing timely solutions and serving as a source of global system knowledge. The group interacted with management on system planning and strategy, system requirements, and determining how easily components would fit into the overall system.

Additional measures to handle increased resource dependence were sometimes necessary. Merging of departments through reorganization or spreading of the workload across multiple departments was often employed. From time to time, reorganization of departments within functions were made to help in adaptation to change in the goals and environment. The project also received help from contractors and many partners. IBM entered contracts with partner companies to develop and test new software. It provided design information, computing resources, and even capital investment to its partners. The chain of partner companies centred around IBM Credit Corporation “as somewhat similar to the very successful Japanese Keiretsu chain of corporations [6].

Development automation tools

As a large software engineering shop, this project relied extensively on CASE tools and development support: the internal Integrated Development Support System (IDSS) was the integrated tool used to support development activities. IDSS provides tools for various development life cycle stages, including design, code, compile, test and debug, integrate, build, and maintain. Supporting the IDSS data management were application and library control system data bases. IDSS collected information about
activities performed and made the information available for query, reporting, and analysts. Although there was room for improvement in tool use, our study estimated that a 22% increase in productivity in this project can be attributed to the use of tools [11].

![Redesigned development process to reduce cycle time](image)

**Redesigned Development Process to Reduce Cycle Time**

The use of the waterfall model in development projects required man’s tasks to be implemented in sequence. Unfortunately, the waterfall process seriously conflicts with efforts to reduce development cycle time. The challenge was to shorten the cycle to fill market demand without hurting software quality. The viable solution was to identify tasks that could be implemented in parallel.

As illustrated in Fig. 1, early participation of customers in reviewing OS/400 requirements and designs allowed participating customers to receive new software and start their Beta tests at the same time as the IBM team performed the Alpha test. Another major contribution to the cycle time reduction efforts was the use of the incremental approach in software development process. Initially, all modules were simulated by dummy stubs. Whenever new modules were completed, their dummy modules were replaced and the system-to-date was tested. This approach allowed the components to be developed and tested in parallel and the system tests to be performed frequently. At the integration test stage, versions of OS/400 software were compiled, integrated, and tested weekly. As the Beta test stage progressed, a few finished versions were ready for delivery when an upgraded version was near completion.

**Controlling software quality**

User involvement and a good feedback process helped the project detect and fix problems at early stages, resulting in cost savings and high software quality.

**Reviews.** The development methodology required three reviews: high level design, low level design, and code. The first examined functions, interface, control blocks, and data structures of the software to ensure that the design met the requirements and that the components fitted together. The low level review focused on the logic flow, pseudocode, and error conditions. The final reviews involved program walk-
throughs by experienced programmers in order to ensure that the programs properly implemented the design and followed coding standards.

*Ear/v customer involvement and feedback.* Customer involvement and feedback to the development process helped to make sure that requirements and specifications were met. Traditionally, the age-old DP-User communication gap prevented many development projects from effectively involving end users. To increase the level of user involvement, IBM Rochester business executives and project team members set up regular meetings with customers. Together, they reviewed and validated project requirements and test results.

*Testing.* Experience, tools, and processes carried over from prior projects helped developers build test plans and test cases early, even before coding started. All software must go through four tests: unit, component, system (Alpha), and early-ship (Beta). In the unit test, programmers tested their individual programs and modules, in the component test, all parts were tested together to ensure proper operation of their internal and external functions, as well as the correctness of the logic between modules. Completion of Alpha and Beta tests ensured the proper functioning of the entire system in operation mode, in all tests, errors were recorded for monitoring and tracking of progress.

In preparation for testing, defects were modeled based on historical data collected. Discrepancies between model and actual defect rates were investigated and studied carefully to enable improvements and corrections. Common causes for bad-fixes were also studied for future prevention and detection. These causes were listed in order of frequency of occurrence:

1. Not thorough enough testing,
2. Problems diagnosed correctly, but incorrect instructions given.
3. Not all steps of problems tested.
4. Incomplete test bucket for total testing, and
5. Inadequate review process.

3.2.  Project results

The announcement of the AS/400 systems with OS/400 in August 1988 was heralded and recognized widely in the computing industry and trade journals. Internally, the entire IBM Rochester plant became the model for many other divisions. The success of the project was overwhelming and from 1986 to 1989 IBM Rochester reported that its software productivity increased by 30%, with a threefold increase in quality while the cost of error detection declined. Revenue per employee increased by 30% and development time was cut in half (21. [81. [121. Best of all, the success of this project helped IBM Rochester to earn the distinguished Malcolm Baldrige National Quality Award in 1990.

3.3.  Room for improvement

Despite this impressive record, there is still room for improvement — especially in distributed processing, development tools, and collaboration support. First, the distributed processing environment and the development tools were not well integrated. Because the source microcode library, compilers, and development tools resided in the IDSS, developers had to write and compile their source programs in the IBM S/370 mainframe where IDSS operated. Compiled object codes were then manually downloaded via network to the AS/400 for linkage and testing. However, the link between the IBM S/370 and AS/400 systems at that time was awkward, unreliable, and time consuming. In many instances, it could take up to
six hours for code to be transmitted! A client/server environment was still under development and was not available at the time. This environment was needed to automate the creating, editing, compiling, linking and testing of software in various host computers. Second, lack of a Graphical User Interface (GUI) debugging tool made it difficult and time consuming to debug programs. Third, communication for cooperation and collaboration with developers at other geographical locations was limited to electronic mail, telephone calls, meetings, and video conferencing. In order to protect the confidentiality of the development efforts, developers were not allowed to use electronic bulletin boards and conference disks, though they would have helped solve development problems more easily. Finally, the recent shift of the development paradigm toward Object Oriented Design and Object Oriented Programming has created the need for new, powerful, and efficient tools to support this environment. In summary, if the client/server environment for software development had been available and the tools had been more appropriate, the development time and cost could have dropped even further.

4. What has been changed?

How do modern management strategies, process, and techniques compare with the experience gained from previous software development projects?

1. The optimistic programmer. Although overoptimistic estimation was often a major cause of project delay and cost overruns, this problem was not caused by programmer optimism. In fact, we found that the project planning and estimation tasks were often the responsibility of project leaders, managers, and steering committees. In contrary to popular belief, programmers and analysts may have a less positive view of their project than their leaders.

2. Adding people to late project. Although our survey of software developers showed that adding personnel often was a popular action, the impacts of adding more people to a late project were not well verified. The results from our survey and project study also showed that adding more people to a late project was popular but not effective in reducing delays; however neither showed that it tends to delay the project further. People were added to the 05/400 project throughout the implementation and testing stages, with many more near delivery, but no negative impacts were seen. The reason that this was so appears to be because the project was not fully staffed and was constantly under deadline pressure, a situation in which project members tend to work much harder and thus accomplish a higher level of productivity than under normal conditions.

3. Chief programmer team model. The chief programmer team model was once very popular. However, the results from our survey and study showed that this model is no longer valued. There are several reasons: team members, especially highly skilled ones, often resented being given orders. Furthermore, increased linkage and coordination with external groups, facilitated by the usage of groupware and information highway, have reduced reliance on a chief programmer. Finally, the existence of Design Control Groups made the chief programmer team model inappropriate.

4. Amount of code and data for testing. There is very little information in the literature about the amount of test code and test data, called `scaffolding code", built during development. In the study, we found the ratio of scaffolding code to actual code in the final product to be at least 10 to 1. There are many reasons: first, the high quality standards of today's software require more test cases and test data; second, the use of the new incremental approach to integration and testing requires a large number of dummy modules; third, the use of modern development tools and the abundance of reusable test cases make it easier for developers to build more test cases and prototypes.
5. Lessons learned and critical success factors

Based on expert judgment and feedback, the key success factors, listed by level of importance are:

— Feedback mechanisms from end-users must be provided. End-users should be involved in the entire development process. Problems encountered should be reported and recorded as soon as possible.

— Good inventory of reusable code and designs help to reduce the development time. Re-used designs also allow developers to understand how to build components in parallel. Almost one third of the total code in the final product was reused from prior efforts.

— Project management techniques and quality control standards should be enforced via thorough reviews, inspections, and tests. Project changes and discovered defects should be closely monitored. When schedule slippage occurred, upgrading priorities and some overtime can be more effective than adding workers. Test cases and quality models should be built in advance for better planning and control of quality assurance efforts.

Table 1 Project management critical success factors

<table>
<thead>
<tr>
<th>Project characteristics &amp; success factors</th>
<th>Survey</th>
<th>OS/400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good communication and feedback</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Changes during implementation minimized</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ample inventory of re-used code</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Project leaders possesses good managerial skills</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Low personnel turnover</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Project members from multiple backgrounds</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Quality assurance not reduced to meet schedule and budget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Full compliance to requirements</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Team members are specialists</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Problems and goals well-defined</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Environment and resource dependence well controlled</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>More emphasis project planning and front end work</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Effective and powerful tool use</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2: Efforts devoted to life cycle activity

<table>
<thead>
<tr>
<th>Development Activities</th>
<th>Survey</th>
<th>OS/400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility, analysis &amp; planning</td>
<td>17%</td>
<td>13%</td>
</tr>
<tr>
<td>Design</td>
<td>15%</td>
<td>25%</td>
</tr>
<tr>
<td>Coding</td>
<td>27%</td>
<td>19%</td>
</tr>
<tr>
<td>Quality assurance</td>
<td>5%</td>
<td>28%</td>
</tr>
<tr>
<td>Debugging</td>
<td>14%</td>
<td>14%</td>
</tr>
<tr>
<td>Coordination</td>
<td>6%</td>
<td>-</td>
</tr>
<tr>
<td>Changes</td>
<td>14%</td>
<td>-</td>
</tr>
<tr>
<td>Others</td>
<td>7%</td>
<td>1%</td>
</tr>
</tbody>
</table>
Software versions must be well-managed to minimize impacts of design changes during development. The use of the incremental approach in integration and testing of software allows versions to be compiled, integrated, and tested frequently.

Coordination and collaboration efforts are needed to control resources and to fulfill requirements. The project goals and requirements should be driven by market needs. The project should rely on collaboration, coordination, cooperation, merging, and use of contract programmers to manage resource dependence.

Favourable publicity and good customer support should be exploited to maintain the firm's competitive advantage. If the software under development meets user expectations and has its niche, the project does not have to be kept completely secret before delivery.

The project should optimize the work force mix of critical and non-critical skills and should allow shifting employees from one area to another. Additional supportive personnel practices include:

- Maintain employee job security,
- Reward fairly and provide ample opportunities for advancement, and
- Recruit development team members from multiple backgrounds and strive for a balance of specialists and generalists.

A comparison of the results of the field survey with those of the OS/400 development study is summarized in Tables 1 and 2. A “-” denotes that the factor or activity was not observed.

Although we could not confirm and match every result from the two studies, the results were not contradictory. Interestingly, a comparison of the results of our survey with those by Genutchten (7) in 1991 show them to be consistent with ours.

Acknowledgement

We gratefully acknowledge the support and approval of IBM Corporation, Rochester Minnesota to its former employee, the first author, to study the OS/400 development project as part of his Ph.D. dissertation.

References