Resource Utilization during Software Development

Marvin V. Zelkowitz

Department of Computer Science, University of Maryland, College Park, Maryland

This paper discusses resource utilization over the life cycle of software development and discusses the role that the current "waterfall" model plays in the actual software life cycle. Software production in the NASA environment was analyzed to measure these differences. The data from 13 different projects were collected by the Software Engineering Laboratory at NASA Goddard Space Flight Center and analyzed for similarities and differences. The results indicate that the waterfall model is not very realistic in practice, and that as technology introduces further perturbations to this model with concepts like executable specifications, rapid prototyping, and wide-spectrum languages, we need to modify our model of this process.

1. INTRODUCTION

As technology impacts on the way industry builds software, there is increasing interest in understanding the software development model and in measuring both the process and the product. New workstation technology (e.g., PCs, CASE tools), new languages (e.g., Ada, requirements and specification languages, wide-spectrum languages), and techniques (e.g., prototyping, object-oriented design, pseudocode) are affecting the way software is built, which further affects how management needs to address these concerns in controlling and monitoring a software development.

Most commercial software follows a development cycle often referred to as the *waterfall* cycle. While there is widespread dissatisfaction with this as a model of development, there have been few quantitative studies investigating its properties. This paper addresses this problem and whether the waterfall chart is an appropriate vehicle to describe software development. Other models, such as the spiral model and value chaining, have been described, and techniques like rapid prototyping have been proposed that do not fit well with the waterfall chart [1, 2]. This paper presents data collected from 13 large projects developed for NASA Goddard Space Flight Center that shed some light on this model of development.

Data about software costs, productivity, reliability, modularity, and other factors are collected by the Software Engineering Laboratory (SEL), a joint research project of NASA/GSFC, Computer Sciences Corporation, and the University of Maryland, to improve both the software product and the process for building such software [3]. It was established in 1976 to investigate the effectiveness of software engineering techniques for developing ground support software for NASA [4].

The software development process at NASA, as well as in most commercial development environments, is typically product-drive and can be divided into six major life-cycle activities, each associated with a specific "end product" [5, 6]; requirements, design, code and unit test, system integration and testing, acceptance test, and operation and maintenance. In order to present consistent data across a large number of projects, this paper focuses on the interval between design and acceptance test and involves the actual implementation of the system by the developer.

In this paper, we will use the term "activity" to refer to the work required to complete a specific task. For example, *coding activity* refers to all work performed in generating the source code for a project, the *design activity* refers to building the program design, etc. On the other hand, the term "phase" will refer to that period of time when a certain work product is supposed to occur. For example, *coding phase* will refer to that period of time during software development when coding activities are supposed to occur. It is closely related to management-defined milestone dates between the critical design review (CDR) and the code review. But during this period other activities may also occur. For example, during the coding phase, design activity is still happening for some of the later modules that are defined for the system and some testing activity is already occurring with some of the modules that were coded into the source program fairly early in the process.

Address correspondence to Professor Marvin V. Zelkowitz, Department of Computer Science, University of Maryland, College Park, MD 20742.

0164-1212/88/$3.50

In the NASA/GSFC environment that we studied, the software life cycle follows this fairly standard set of activities [7]:

1. The *requirements* activity involves translating the functional specification consisting of physical attributes of the spacecraft to be launched into requirements for a software system that is to be built. A functional requirements document is written for this system.
2. A *design* activity can be divided into two subactivities: preliminary design activity and detailed design activity. During *preliminary design,* the major subsystems are specified, and input-output interfaces and implementation strategies are developed. During *detailed design,* the system architecture is extended to the subroutine and procedure level. Data structures and formal models of the system are defined. These models include procedural descriptions of the system; data flow descriptions; complete description of all user input, system output, input-output files, and operational procedures; functional and procedural descriptions of each module; and complete description of all internal interfaces between modules. At this time a system test plan is developed that will be used later. The design phase typically terminates with the CDR.
3. The *coding and unit test* activity involves the translation of the detailed design into a source program in some appropriate programming language (usually Fortran, although there is some movement to Ada). Each programmer will unit test each module for apparent correctness. When satisfied, the programmer releases the module to the system libraian for configuration control.
4. The *system integration and test* activity validates that the completed system produced by the coding and unit test activity meets its specifications. Each module, as it is completed, in integrated into the growing system, and an integration test is performed to make sure that the entire package performs as expected. Functional testing of end-to-end system capabilities is performed according to the system test plan developed as part of preliminary design.
5. In the *acceptance test* activity, a separate acceptance test team develops tests based on functional specifications for the system. The development team provides assistance to the acceptance test team.
6. *Operation and maintenance* activities begin after acceptance testing when the system becomes operational. For flight dynamics software at NASA, these activities are not significant with respect to the overall cost. Most software that is produced is highly reliable. In addition, the flight dynamics software is usually not "mission critical," in that a failure of the software dos not mean spacecraft failure but simply that the program has to be rerun. In addition, many of these programs (i.e., spacecraft) have limited lifetimes of 6 months to about 3 years, so the software is not given the opportunity to age.

The waterfall model makes the assumptions that all activity of a certain type occurs during the phase of that same name and that phases do not overlap. Thus all requirements for a project occur during the requirements phase; all design activity occurs during the design phase. Once a project has a design review and enters the coding phase, then all activity is coding. Since many companies keep resource data based on hours worked by calendar date, this model is very easy to track. However, as Figure 1 shows, activities overlap and do not occur in separate phases. We will give more data on this later.

2. THE WATERFALL CHART IS ALL WET

Table 1 summarizes the raw data on the 13 projects analyzed in this paper. They are all fairly large flight dynamics programs ranging in size from 15,500 lines of Fortran code to 89,513 lines of Fortran, with an average size of 57,890 lines. The average work on these projects was 8.90 staff-months; thus, all represent significant effort.

In most organizations, weekly time sheets are collected as part of cost accounting procedures so that phase data are the usual reporting mechanism. However, in the SEL, weekly activity data are also collected. The data consist of nine possible activities for each component

Table 1. Project Size and Staff-Month Effort

Project number	Size (lines of code)	Total effort hours[a]	Staff-months
1	15,500	17,715	116.5
2	50,911	12,588	82.8
3	61,178	17,039	112.1
4	26,844	10,946	72.0
5	25,731	1,514	10.0
6	67,325	19,475	128.4
7	66,260	17,997	118.4
8	—[b]	—[b]	—[b]
9	55,237	15,262	100.4
10	75,420	5,792	38.1
11	89,513	15,122	99.5
12	75,393	14,508	95.4
13	85,369	14,309	94.1
Average	57,890	13,522	89.0

[a] All technical effort, including programmer and management time.
[b] Raw data not available in data base.

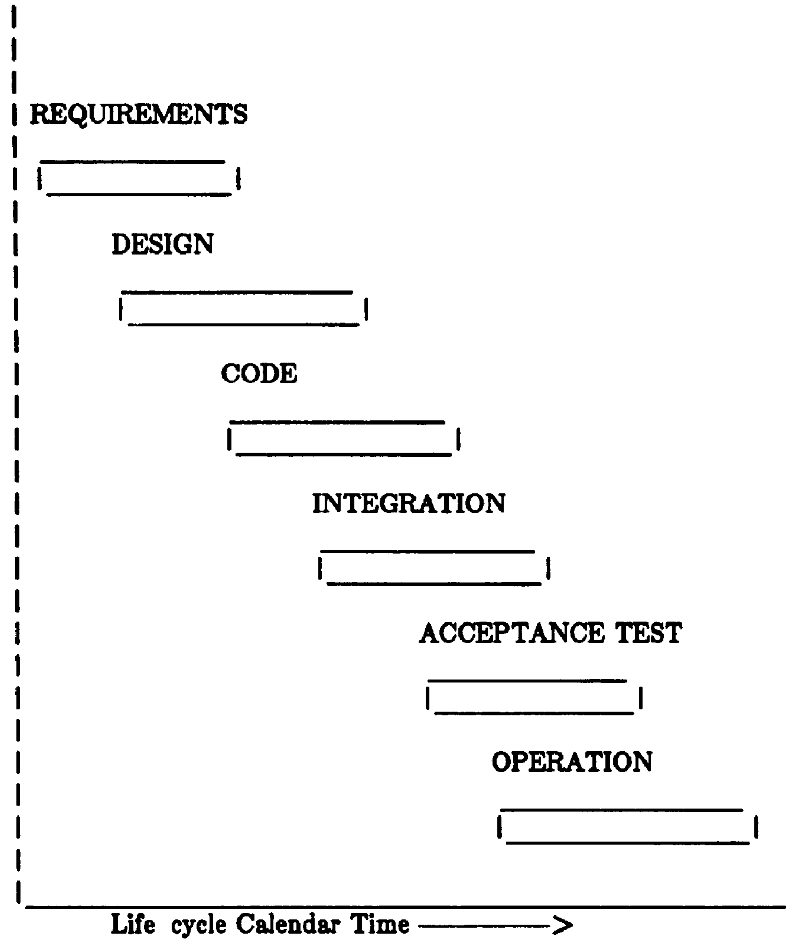

Figure 1. Typical life cycle.

(e.g., source program module). In this paper, these will be grouped as design activities, coding activities (including unit test), integration activities, acceptance testing activities and other. Specific meetings, such as design reviews, will be grouped with their respective activity (e.g., a design review is a design activity, a code walkthrough is a coding activity, etc.)

Table 2 classifies the data presented in this paper. Each column represents a type of work product (design, code, test). The "by phase" part represents the effort during that specific time period, while the "by activity" part represents the actual amount of such activity. "Other" does not enter into the "by phase" table, since these activities occur during all phases. At NASA, 22% of a project's effort occurs during the design phase, while 49% is during coding. Integration testing takes 16% while all acceptance activities take almost 13%. (Remember that requirements data are not being collected here. We are simply reporting the percentage of design, coding, and testing activities. A significant requirements activity does occur.)

By looking at all design effort across all phases of the projects, we see that design activity is actually 26% of the total effort rather than the 22% listed above. The coding activity is a more comparable 30% rather than the 49% listed by phase data, which means that the coding phase includes many other tasks. "Other" increased from 12% to 29% and includes many time-consuming tasks that are not accounted for by the usual life-cycle accounting mechanism. Here, "other" includes acceptance testing as well as activities that take a significant effort but are usually not separately identifiable using the standard model. These include corporate (not technical) meetings, training, travel, documentation, and various other tasks assigned to the personnel. The usual model of development does not include an "other," and this is significant since almost one-third of a project's costs are not effective at completing it. More on this later.

The situation is actually more complex, since the distribution of activities across the project is not reflected in Table 2. These data are presented in Tables 3-5. Only 49% of all design work actually occurs during the design phase (Table 3), and one-third of the total design activity occurs during the coding period. Over one-sixth (10.3% + 6.4%) of all design occurs during

Table 2. Development Effort

Project number	Design (%)	Code (%)	Integration act. (%)	Accept. test and other (%)
By Phase				
1	20.6	38.6	16.5	24.3
2	16.2	48.4	19.3	16.2
3	21.8	47.9	17.4	12.9
4	35.9	39.5	24.5	0.1
5	18.2	68.8	13.0	0.0
6	16.3	48.6	10.9	24.3
7	19.0	50.4	14.9	15.7
8	22.9	48.4	13.0	15.8
9	22.6	68.3	8.1	1.1
10	24.4	44.6	20.2	10.8
11	22.7	39.4	21.4	16.5
12	16.9	53.1	10.9	19.1
13	28.2	43.5	20.1	8.2
Average	22.0	49.2	16.2	12.7
By Activity				
1	17.4	16.4	9.9	56.3
2	30.1	39.4	20.8	9.7
3	26.3	20.3	19.3	34.2
4	27.3	28.7	6.0	38.0
5	31.0	35.5	9.4	24.1
6	14.9	21.8	24.0	39.2
7	20.2	25.9	14.3	39.6
8	11.0	13.9	9.3	65.8
9	31.3	43.5	18.9	6.4
10	38.2	37.3	6.1	18.4
11	29.3	31.0	17.2	22.5
12	23.7	46.5	24.0	5.9
13	32.6	36.3	15.6	15.6
Average	25.6	30.5	15.0	28.9

Table 3. Design Activity During Life-Cycle Phases

Project number	Design phase (%)	Coding phase (%)	Integration test (%)	Accept. test phase (%)
1	41.8	33.9	10.0	14.3
2	53.6	31.2	9.2	6.0
3	33.3	37.1	19.7	9.9
4	45.3	32.6	22.0	0.1
5	17.4	69.1	13.5	0.0
6	58.9	30.7	4.3	6.2
7	63.9	15.3	6.8	14.1
8	28.1	56.9	7.1	8.0
9	61.8	38.2	0.0	0.0
10	57.8	27.2	7.0	8.0
11	58.7	13.7	16.67	10.9
12	58.9	32.8	5.9	2.4
13	60.5	24.7	11.9	2.9
Average	49.2	34.1	10.3	6.4

testing when the system is "supposed" to be finished. In almost one-third of the projects (4 out of 13), about 10% or more of the design work occurred during the final acceptance testing period.

As to coding effort, Table 4 shows that while a major part (70%) does occur during the coding phase, almost one-quarter (16% + 7%) occurs during the testing periods. As expected, only a small amount of coding (7%) occurs during the design phase; however, the table indicates that some coding does being on parts of the system while other parts are still under design. These data have the widest variability as a range from 0% (project 10) to over 22% (project 3).

Similarly, Table 5 shows that significant integration testing activities (almost one-half) occur before the integration testing period. Once modules have been unit tested, programmers begin to piece them together to build larger subsystems, with almost half (43%) of the integration activities occurring during the coding phase.

Due to the wide variability of the "other" category in Table 2, Table 6 presents the same data as relative percentages for design, coding, and integration testing with the other category removed. As can be seen, design took about one-third of the development effort and varied between a low of 25% and a high of 47%—a factor of almost 2. On the other hand, coding took an average of 42% of the relative effort and varied between 36% and 49%—a factor of only 1.36. Testing ranged from a low of 7.5% to a high of 39.5%, with an average of 22%, for a relative factor of over 5.

From Table 2, the "other" category was 29% of the effort on these projects, and of the 13 measured projects, other activities consumed more than one-third of the effort on six of them. The other category consists of activities such as travel, completion of the data collection forms, meetings, and training. While these activities are often ignored in life-cycle studies, the costs are significant. Table 7 presents the distribution of other

Table 4. Coding and Testing Activity During Life-Cycle Phases

Project number	Design phase (%)	Coding phase (%)	Integration test (%)	Accept. test phase (%)
1	1.4	78.3	11.3	9.1
2	0.0	72.8	19.7	7.5
3	22.2	56.2	11.8	9.8
4	16.4	58.5	25.1	0.1
5	21.2	68.7	10.1	0.0
6	0.5	77.3	11.3	10.9
7	1.3	73.9	15.6	9.2
8	14.7	54.7	21.0	9.7
9	5.2	91.1	3.1	0.6
10	0.0	73.0	22.5	4.5
11	2.2	70.5	20.1	7.2
12	0.3	74.8	8.3	16.6
13	4.6	63.6	26.9	4.9
Average	6.9	70.3	15.9	6.9

Table 5. Integration Activity During Life-Cycle Phases

Project number	Design phase (%)	Coding and unit phase (%)	Integration test (%)	Accept. test phase (%)
1	0.0	17.8	27.4	54.7
2	0.0	45.2	30.1	24.7
3	6.1	53.9	21.1	18.9
4	21.0	39.3	39.7	0.0
5	28.4	71.0	0.6	0.0
6	1.0	40.9	17.6	40.5
7	0.5	54.1	26.3	19.2
8	2.9	33.8	19.2	44.1
9	0.0	66.4	29.2	4.4
10	0.0	23.1	41.5	35.5
11	0.0	36.4	35.1	28.5
12	0.1	32.7	22.4	44.8
13	1.5	49.5	28.8	20.2
Average	4.7	43.4	26.1	25.8

activities across all phases. While such effort varies widely from project to project, no general trends can be observed, except that it does take a significant effort as a percent of total costs.

3. CONCLUSIONS

Using data from the SEL database, it seems that the software development process does not follow the waterfall life cycle but appears to be more a series of rapids as one process flows into the next. Significant activities cross phase boundaries and do not follow somewhat arbitrary milestone dates. The classical product-driven model has many shortcomings.

In the SEL environment, as well as elsewhere, other classes of activities take a significant part of a project's resources. At almost one-third of the total effort, it

Table 6. Relative Activity

Integration act. (%)	Project number	Design act. (%)	Coding and unit act. (%)
1	39.9	37.5	22.6
2	33.3	43.7	23.0
3	39.9	30.8	29.3
4	44.0	46.3	9.7
5	40.8	46.8	12.3
6	24.6	35.9	39.5
7	33.5	42.8	23.6
8	32.2	40.7	27.1
10	46.8	45.7	7.5
11	37.8	40.1	22.1
12	25.2	49.4	25.5
13	38.6	43.0	18.4
Average	36.2	42.2	21.6

Table 7. Other Activities Effort in Each Phase

Project number	Design phase (%)	Coding and testing phase (%)	Integration test (%)	Accept. test phase (%)
1	23.3	32.2	18.1	26.5
2	0.0	9.1	26.4	64.6
3	21.7	47.8	16.8	13.7
4	46.2	30.2	23.6	0.0
5	11.0	67.7	21.3	0.0
6	18.2	44.2	9.0	28.7
7	14.4	51.6	14.5	19.5
8	26.5	47.7	11.4	14.4
9	15.9	65.5	18.7	0.0
10	12.4	30.2	35.9	21.5
11	21.4	32.2	18.9	27.6
12	47.3	46.6	4.6	1.5
13	42.5	30.0	12.7	14.9
Average	23.1	41.2	17.8	17.9

might be part of an explanation of why software is typically over budget. Estimating procedures often use a work breakdown structure where the system is divided into small pieces and estimates for each piece are summed up. Inclusion of a significant "other" usually does not occur.

Newer technology is affecting this traditional model even more. In one NASA experiment, a prototype of a project was developed as part of the requirements phase [8]. In this case, 33,000 lines of executable Fortran were developed at a cost of 93.1 staff-months—already a significant project in this SEL environment. When viewed as a separate development, the prototype had a life cycle typical of the data presented here, but if viewed as only a requirements activity it puts a severe strain on existing models.

Current models do not handle executable products as part of requirements. Other questions arise: Are Ada package specifications design or code? Are executable specification languages specification or design? When does testing start?

It is clear that our current product-driven models need to be updated. Other models, such as the spiral model, which is an iterative sequence of risk-assessment decisions, or value chaining, which addresses value added by each phase, are alternative approaches that need to enter our vocabulary and be further studied for effectiveness.

ACKNOWLEDGEMENT

This work was partially supported by grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland. Judin Sukri provided most of the analysis of the data used in this report. We also wish to acknowledge the help of Frank McGarry of NASA/GSFC in collecting and interpreting the data used here.

REFERENCES

1. B. Boehm, A Spiral Model of Software Development and Enhancement, *ACM Software Eng. Notes* 11(4) 22–42, 1986.
2. B. Boehm, Improving Software Productivity, *Computer* 20(9) 43–57, 1987.
3. F. E. McGarry, et al., Guide to Data Collection, NASA Goddard Space Flight Center, Code 552, Greenbelt, MD, August 1982.
4. V. R. Basili and M. V. Zelkowitz, Analyzing Medium-Scale Software Development, 3rd International Conference on Software Engineering, Atlanta, pp. 116–223, 1978.
5. A. Wasserman, Software Engineering Environment, *Adv. Comput.*, 22, 110–159, 1983.
6. M. W. Zelkowitz, Perspective on Software Engineering, *ACM Comput. Surv.*, 10(2), 198–216, 1978.
7. F. E. McGarry, G. Page, et al., Standard Approach to Software Development, NASA Goddard Space Flight Center, Code 552, Greenbelt, MD, September 1981.
8. M. V. Zelkowitz, The Effectiveness of Software in Prototyping: A Case Study, ACM Washington Chapter 26th Tech. Symposium, Gaithersburg, MD, pp. 7–15, 1987.