Software Development vs. Software Maintenance

Rahul Agarwal
Virginia Tech
rahulaga@vt.edu

Abstract

Software design and development has come a long way since the early ad hoc ways to development frameworks of today. We will try to understand what Software Project Management is and how it plays a critical role in the success of any software product. We will also analyze how Project Management applies to and differs for Software Development and Software Maintenance. While the success rate of software projects has been notoriously low, the exponential increase in the absolute number of projects since 1970s and given the ubiquitous use of software we can safely assume that improvement in software project management is underway and we will have improved development and maintenance of software in the future.

1. Introduction

Software Engineering practices from the typical 'Waterfall Model' view consists of:

(i) Requirements and Requirement Specifications
(ii) Design and Architecture
(iii) Programming and Integration
(iv) Testing and Software Delivery
(v) Maintenance

For the purpose of our discussion we would like to group the first four elements as 'Development' and the study Maintenance separately and this is shown in Figure 1.

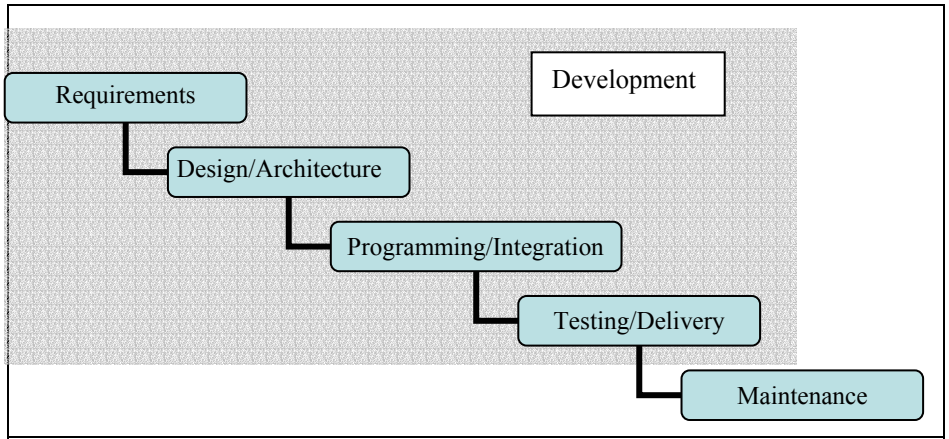

Figure 1: Software Development and Software Maintenance

While Software Development and Software Maintenance are two distinct and widely separated activities they are actually *part of the same life cycle* and the same importance needs to be assigned to both for overall success and good Project Management. However in the real world only about 2% of the research in past 10 years in computer science has been devoted to maintenance [Kemerer CF], so while I try to treat equally and objectively the subject of development and maintenance there is an awkward imbalance in the available literature and empirical studies.

As pointed out in [CS5984] for the year 1999 of the total IT spending 30% was for maintenance, 25% for new development and the rest on infrastructure and other expenses (Figure-2). Therefore even though maintenance has the largest share of the budget it is ironic that there is not much focus on this area. Let us now look at Software Development and Software Maintenance briefly.

Software Development in our discussion encompasses broadly the Rational Unified Process (RUP) [RUP, Royce 1998] and its four phases:

(i) Inception: It involves establishing the project scope and concept, demonstrating at least one

possible architecture, estimating cost and schedule and estimating potential risks.

(ii) Elaboration: It involves baselining the architecture and a completing the plan for the construction phase.

(iii) Construction: It involves developing quality software as rapidly as possible to optimize resource use and achieve (alpha, beta) releases.

(iv) Transition: It involves testing and deploying the new system and assessing user response and training end users and maintainers.

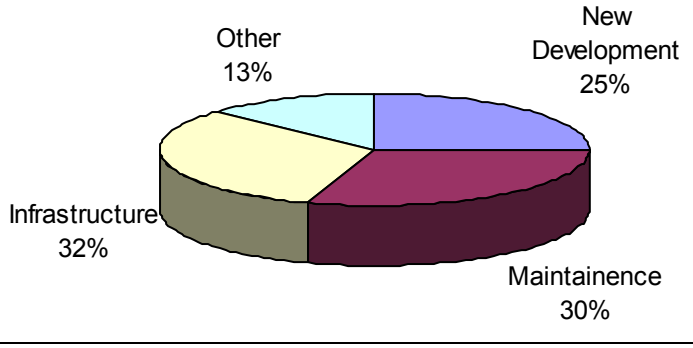

Figure 2: IT Spending (1999) [CS5984]

Each phase has iterative steps and starting from requirements gathering it culminates with product deployment and training. For our discussion we assume that the software development ends once the product is deployed and handed over to the users.

Software Maintenance involves making changes to the software *after delivery* in order to correct faults and deficiencies found during usage, or to improve performance or any other attribute as well as to adapt the product to a new environment. These are generally classified as [Favre]:

(i) Corrective – due to error detections
(ii) Adaptive – due to change in environment
(iii) Perfective – for improving quality (e.g. performance)
(iv) Evolutive – due to changing user requirements
(v) Preventive – due to forseeable errors but with no immediate benefit (e.g. Y2K problem)

Software maintenance is not given the priority it deserves and as such it is often neglected by the management [Charette 1997]. There are many risk factors in maintenance as the maintainers have to work with an existing system, have little or no documentation, have many layers of patches and fixes on top the original design, and little flexibility in changing the overall structure of the product. They also have to deal more with customer relations as customers *expect* the system to be running smoothly and are not as forgiving as in the development stage.

There are a number of reasons suggested by [Kemerer CF] for poor maintenance and ironically lack of budget/resources is not an issue. The main problems are:

(i) Poor quality of existing software
(ii) System architecture and design done without maintenance in mind and
(iii) Inexperienced personnel

In my opinion the most important factor is the failure to attract top talent and personnel in this critical field since maintenance is generally not as exciting as creating something new.

This article will rigorously try to examine the role of Software Project Management in development and maintenance. In trying to compare and contrast Software Development and Maintenance with regards to Project Management we need to first examine the role of Project Management and its critical aspects and then apply these objectively to both (Section 2). We can then compare the results (Section 3) and conclude with their differences and analyze how to bring balance into these processes (Section 4).

2. Software Project Management for Development and Maintenance

A successful project will meet customer expectations, cost, schedule, quality, features, and make a profit for the developer [Royce 1998]. While these seem daunting, a good software project management approach can balance the trade-offs and make them a reality. Software Engineering is different from conventional Manufacturing Engineering processes and has the properties of complexity, conformity, changeability and invisibility [SE]. Managing software on economies of scale is also not possible. The marginal cost [Gwartney] associated with every additional function point or line of code is not negligible as suggested by classical economic theories for other engineering disciplines. Managing software is thus inherently different from conventional

products; therefore we need to study Management with respect to software as a special topic.

"Project Management is a system of management procedures, practices, technologies, skill, and experience that are necessary to successfully manage an engineering project." [Thayer]. With regards to software it becomes known as Software Engineering Project Management and involves dealing with five major aspects common to any management activity [CS5984, Thayer]. These are:

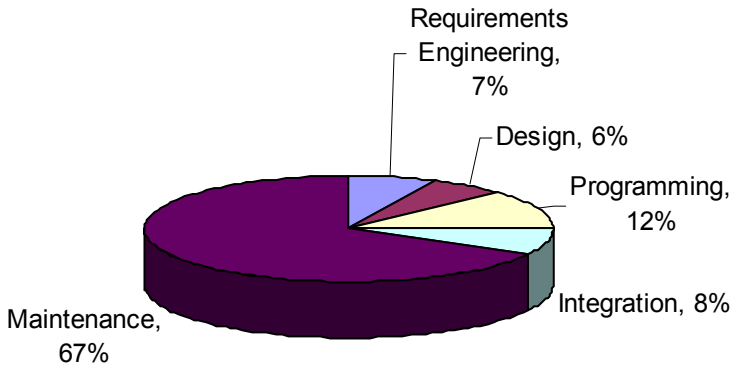

Figure 3: Relative Costs of Phases of a Life-cycle [Schach 1999]

- *Planning*: This involves pre-determining the complete course of action to meet project and organizational objectives by specifying goals and objectives and developing strategies, policies, plans and procedures for achieving them. This broadly includes risk assessment and mitigation, determining future course of action and preparing budgets and plans.

- *Organizing*: This involves developing an efficient project and organizational structure "to allocate artifacts and responsibilities" [Royce 1998] in a pre-defined manner across teams. Organizations are generally organized with an Organization Manager together with four management authorities overseeing the Process, Environment, Quality and Personnel and a number of individual Project Managers. A project is organized with a software management team overseeing the system engineering, administration, architecture, development and assessment teams. Each team is relatively autonomous and responsible of the set of tasks assigned.

- *Staffing*: This involves selecting and training the vital human resources of the organization. It broadly includes recruiting qualified personnel, mentoring them into the organizational culture, providing necessary training and conducting periodic employee assessment and feedback.

- *Directing*: This involves providing leadership, motivation, guidance and creating an atmosphere conducive for achieving organizational and project goals set out earlier. Efficient communication among all personnel at all levels of the management and conflict resolution are essential.

- *Controlling*: This involves measuring and evaluating performance and progress of the organization and project towards meeting the goals. Projects metrics to promote visibility, monitoring and reporting systems are used and standards applied to assess the quality of the products.

These encompass all the desired functions for managing a project however to do so *successfully* we need to take care of a number of aspects that are now discussed with respect to, and later applied to development and maintenance.

Life-Cycle

Choosing the correct life-cycle is important and it determines how maintainable the software product will be [Zagal 2002]. The important life-cycle models are Water Fall model, Spiral model, Exploratory model, Incremental model and Reuse-based model; each with its pros and cons. While each has the typical five stages described in the introduction there are variations how they are achieved. The stages maybe undertaken exclusively and the next step undertaken only upon completion of the current one or with repeated development of software artifacts with improvements in a number of ways including, rapid prototyping, customer involvement, and creating more risk driven approaches. Further details about Software Development stages can be found in [SE, RUP, Royce 1998] and are not discussed here.

According to [Schach 1999] the maintenance part of the life-cycle accounts for 67% of the project cost

(Figure 3). Therefore in choosing the life-cycle [Zagal 2002] suggests a shift from the traditional approach by incorporating the maintenance into the very core of the development life-cycle in a "Maintenance-Oriented" model. Thus to build software with maintenance in mind a number of changes shall be required at every stage of development. This may however not always be possible given various constraints, for example performance requirements might conflict with modularity requirements. Trade-offs will therefore have to be made and generally since maintenance is not accorded a high priority it is often neglected.

For Software Maintenance [DOE 2002] proposes a basic maintenance model using seven stages based "on the same software engineering principles and practices that lower risk and improve quality during the planning and development stages of the life-cycle."

These seven stages are derived by grouping "logically related" segments of work.

(i) Problem/Modification Identification Stage
(ii) Analysis Stage
(iii) Design Stage
(iv) Construction Stage
(v) System Test Stage
(vi) Acceptance Stage
(vii) Delivery Stage

As we can see these stages are very similar to the development process and similar management strategies can be applied.

Project planning and Cost Estimation

A software project plan is as intangible as the software itself and iterative processes are required to formulate a good plan [Royce 1998]. There are a number of planning strategies and a good plan is essential for estimating time, effort, resources and cost required to build a software product.

We can decompose our plan into smaller more manageable task sets using *Work Breakdown Structures (WBS)*. WBS decomposes the project into discrete work tasks, assigns responsibilities and sets framework for scheduling and budgeting. Conventionally these WBS were too structured around the product design, had too little or too much detail and could not be compared with others.

In the Evolutionary approach 'levels' are defined for different workflows and phases and this makes them more useful. There are two perspectives, top-down and bottom-up and they detail the plan from macro-level to the micro-view and vice versa using the WBS partitions [Royce 1998]. The plan is essential for both development and maintenance however it would have to be determined separately to meet the different objectives.

Software costs are determined by [Royce 1998]:

(i) Its size (function points, lines of code)
(ii) Processes used
(iii) Personnel (experience, training)
(iv) Environment (tools, techniques)
(v) Quality

With a quantum improvement in technology we can now use higher order languages (Java, C++), employ the Object Oriented paradigm, and use iterative development based on pre-defined frameworks like J2EE and Microsoft .NET. These together with Visual development and integration tools and faster and significantly cheaper hardware can improve software economics dramatically.

The environment for software development and maintenance must be suitably automated and these needs depend on the scale of the project. It is essential to setup the infrastructure to meet the requirements for successful iterative development of any software project. Automation is needed at all process levels and is part of a good project plan. The different automation needs depend on the workflows of the project and these are [Royce 1998]:

(i) Management using metrics and cost estimation
(ii) Environment using change management and version control
(iii) Requirements using vision statements, use cases and iterative models
(iv) Design using visual modeling
(v) Implementation using editors, compilers, linkers and debuggers
(vi) Assessment and Deployment using test automation and defect tracking

Artifacts and Milestones

These are strongly related to our next point of discussion (Metrics) and are the critical tangible (printed manuals) and intangible (software code) process products that need to be examined at every phase. [Royce 1998] classifies artifacts created in each

of the stages into (i) Management and (ii) Engineering Sets. Generally all the artifacts are composed of text, graphics, UML, or other ad hoc notations and formats.

The Management artifacts comprise

(i) Planning Artifacts - work breakdown structure, business case, release specifications, software development plan
(ii) Operational Artifacts - release description, status assessments, software change order database, deployment documents and environment

The Engineering artifacts comprise

(i) Requirements Set - vision documents and requirements models
(ii) Design Set - design models, test models and software architecture description
(iii) Implementation Set - source code baseline and associated files, and component executables
(iv) Deployment Set - integrated executable product, associated files and user manual

These artifacts are used in determining progress and according to Boehm [Boehm 1996] there are three milestones to achieve:

(i) Life-cycle objectives (LCO)
(ii) Life-cycle architecture (LCA)
(iii) Initial operational capability (IOC)

Each of these milestones can be applied to any suitable development process like waterfall or spiral. The most important goal of LCO is to determine the feasibility of the project. LCO thus tries to determine the objective and scope of the project, determine normal and abnormal scenarios of the proposed system usage, determine the stakeholders that include users, customers, developers, maintainers, the inter-operator organizations if the system is linked to external systems, and also a public representative if it involves their safety, privacy or other issues.

LCO also includes gathering system requirements and sufficient information to identify system architectures. It recommends using the WWWWWHH principle of assessing Objectives (Why), Milestones and Schedules (What, When), Responsibilities (Who, Where), Approach (How), and Resources (How much).

Most LCA components are derived from LCO and after an iterative process leads to the final system and software architecture. Both emphasize on system feasibility study. IOC includes software development, documentation, licensing, deployment and training.

While most of these milestones apply specifically to software development we must also identify milestones for maintenance. In my opinion these are similar to the ones discussed above.

Project Metrics

Software project metrics are an important part of any software development and they make progress and quality tangible and provide a basis for estimating cost and schedule for project completion. Metrics can be summarized into the following categories:

(i) Cost: Measuring actual vs. budgeted labor, hardware, software, office and other costs
(ii) Schedule: Measuring actual vs. planned deliverables completed, deliverables not completed, milestones met and milestones not met
(iii) Risk: Measuring anticipated vs. actual events and impact
(iv) Quality: Measuring actual vs. planned reviews, defects in code/documentation, major/minor defects and the origin of defect.

There are seven core metrics defined in [Royce 1998] for Software Development and classified under two heads:

Management

(i) Work and Progress: It comprises defining estimates and tracking progress and can be assessed by completion of use-cases, counting lines of code, evaluating test cases, and checking against milestones.
(ii) Budgeted Costs and Expenditures: Comparing costs and expenditures and this can be completed very objectively and these can also be compared to estimated cost and projections can be made.
(iii) Staffing and team dynamics: Starting from a small team growth progresses as risks are better understood. Employee turnover shows poorly on the project and is an important measurement.

Table 1: Metrics for Software Maintenance [DOE 2002]							
	Problem Identification, Modification Stage	Analysis Stage	Design Stage	Programming Stage	System Test Stage	Acceptance Stage	Delivery Stage
Factors	Correctness Maintainability	Flexibility Traceability Usability Reusability Maintainability Comprehensibility	Flexibility Traceability Reusability Testability Maintainability Comprehensibility Reliability	Flexibility Traceability Maintainability Comprehensibility Reliability	Flexibility Traceability Verifiability Testability Interoperability Comprehensibility Reliability	Flexibility Traceability Interoperability Testability Comprehensibility Reliability	Completeness Reliability
Metrics	No. of omissions on Modification Request (MR) No. of MR submittals No. of duplicate MRs Time expended for problem validation	Requirement changes Documentation error rates Effort per function area (e.g., SQA) Elapsed time (schedule) Error rates, by priority and type	S/W complexity Design changes Effort per function area Elapsed time Test plans and procedure changes Error rates, by priority and type Number of lines of code, added, deleted, modified, tested	Volume/ functionality (function points or lines of code) Error rates, by priority and type	Error rates, by priority and type Generated Corrected	Error rates, by priority and type Generated Corrected	Documentation changes (i.e. version description documents, training manuals, operation guidelines)

Quality

(iv) Change traffic and stability: These qualities are inter-related and with decrease in change requests the software converges towards stability and predictability
(v) Breakage and Modularity: Modularity is average breakage (extent of change) and over time these should decrease for greater stability
(vi) Rework and Adaptability: This aspect measures the time required for cost of change
(vii) Mean time between failures (MTBF) and maturity: Maturity is gained as the MTBF decreases over time and these are important quality characteristics

Software Maintenance metrics according to [DOE 2002] are defined across each of its stage and summarized in Table 1.

Risk Management

Risk is defined as "a potential problem; a problem is a risk whose likelihood has reached 100%" [Charette 1997] or "Risk is the possibility of suffering loss" [SEI]. Risk is essentially lack of information, time or control [CS5984] and failure to take action can result in greater risk and even failure. The basic [SEI] paradigm is to:

(i) Identify (locate risks before they become problems)
(ii) Analyze (evaluate impact and probability)
(iii) Plan (mitigate risk by present and future action)
(iv) Track (monitor) and
(v) Control (correct deviations) the risk

Figure 4: Risk Management Paradigm [SEI]

This is undertaken using efficient communication. Following this paradigm Risk Management is the process of assessing risk, taking steps to reduce it to an acceptable level and maintaining that level of risk.

Two types of risk management have been suggested in [CS5984]:

(i) Reactive Risk Management: As the name suggests its reacting to problems as they occur and there is no future planning to *prevent* failures
(ii) Proactive Risk Management: This involves formal risk analysis and mitigation plans to ensure that projects are not disrupted.

Effective Risk Management can be done using the framework of the 7 principles.

Table 2: Risk Management Principles [SEI]

Global perspective	• Viewing software development within the context of the larger systems-level view • Recognizing the opportunity in risk
Forward-looking view	• Thinking for the future • Managing resources while anticipating uncertainties
Open communication	• Encouraging free-flowing information at and between all project levels • Enabling all kinds of communication • Using processes that value the individual voice (bringing unique knowledge and insight to identifying and managing risk)
Integrated management	• Making risk management an integral and vital part of project management • Adapting risk management methods and tools to a project's infrastructure and culture
Continuous process	• Constant vigilance • Identifying and managing risks routinely through all phases of the project's life-cycle
Shared product vision	• Product vision based on common purpose, shared ownership, and collective communication
Teamwork	• Working cooperatively to achieve common goal • Pooling talents, skills, and knowledge

Human Resource Management (HRM)

Human Resources are the most important asset of any organization and HRM tasks include [Thayer]:

(i) Recruiting: filling organizational positions by hiring new personnel or promoting qualified people
(ii) Assimilating: Orientate and familiarize new people with the organization culture and practices
(iii) Training: Provide help to improve knowledge, attitude and skills of the people
(iv) Appraising: Evaluate personnel and set performance goals
(v) Compensating: Provide wages, bonuses, benefits, and other remuneration
(vi) Terminating: Remove people if necessary

The quality and motivation of personnel is very important and can determine project success or failure. However it is imperative to recognize that [Bryan] "Not all programmers are created equal." This needs to kept in mind when allocating different tasks of development and maintenance in a team.

3. Comparison of Development and Maintenance

The importance and high cost of maintenance has been emphasized throughout this discussion. However on most commercial projects the development and maintenance contracts are awarded to separate contractors. This adds to the maintenance woes since no matter how well documented there is no way to express the developers mind on paper. Also the lack of qualified personnel and relegation of maintenance creates a disparity between development and maintenance.

Setting it apart from development, in addition to the direct costs mentioned so far maintenance also incurs indirect costs in the form of deterioration of software and customer displeasure. The latter can be serious for

Table 3: Summary comparison of Software Development and Software Maintenance			
	Software Development	**Software Maintenance**	**Remark**
Life-Cycle	• Multiple approaches – linear, iterative, incremental, evolutionary	• Essentially based on customer feedback and "build-and-fix" approach • Adaptive • Preventive • Corrective • Perfective • Evolutive	• Quick solutions are preferred [Favre] therefore elaborate life-cycles not available for maintenance
Planning and Cost Estimation	• WBS • Automation • Cost estimation models	• Automation • Ad hoc cost estimation	• NIL
Artifacts and Milestones	• Management and Engineering artifacts • LCO, LCA, IOC milestones	• NIL	• NIL
Project Metrics	• Cost • Schedule • Risk • Quality	• Correctness • Maintainability • Traceability • Reusability • Reliability • Flexibility • Testability • Usability • Comprehensibility • Completeness	• Maintenance metrics listed include development metrics as well
Risk Management	• Risk Management Paradigm [SEI]	• Incorporating Risk Management into the organization [Charette 1997]	• NIL
Human Resource Management	• Well trained personnel • All Universities offer courses/degrees • Certifications (e.g. Microsoft (MCSA, MCSD), Sun Certifications etc)	• NIL	• Poor training and resource availability

continued for business survival. Of the desired management functions [CS5984, Thayer] while all are applied to development to a large extent, they are hardly applied to maintenance and seriously lack in staffing and direction.

Development processes need to be modified to build software modules with high cohesion and low coupling, thus enabling easier maintenance. With emergence of commercial off the shelf software

(COTS) modularity among software is increasing. This is a welcome change though we need to be wary of the reliability of component vendors.

Based on the Software Management aspects studied in Section 2, Table 3 summarizes the differences between Development and Maintenance with regard to Project Management.

4. Conclusion

Software Project Management has become more complex with the evolution of technology and as we move ahead with more complex and service oriented software products new paradigms will emerge. Both Development and Maintenance play equally important roles in the success of a project and both get will become more complex and harder to manage.

To get better return on investment on our maintenance dollars we need to build software with maintenance in mind. While this shall be an ongoing process the importance of project management cannot be undermined and it will continue to determine project success or failure.

5. References

[Boehm 1996] Boehm, B., Anchoring the Software Process, *IEEE Software*, July 1996

[Bryan] Bryan, GE, Not all Programmers are created Equal, University of California, Irvine

[Charette 1997] Charette, R. et al, Managing Risk in Software Maintenance, *IEEE Software*, Many/June 1997

[CS5984] Bohner, S., CS5984 Lecture Notes, Virginia Tech, Fall 2004

[DOE 2002] System Engineering Methodology Version 3, US Department of Energy, Directive number DOE 200.1-1A, September 2002, Chapter 10

[Favre] Software Maintenance http://www-adele.imag.fr/~jmfavre/ENSEIGNEMENT/TRANSPARENTS/SoftwareMaintenance/7/SoftwareMaintenance-7.pdf

[Gwartney] Gwartney et al, *Economics: Private and Public Choice*, Thomson South-Western, Tenth Edition, Chapter 1.

[Kemerer CF] Kemerer, C. F., *Software Maintenance*, University of Pittsburg, Chapter 11

[Royce 1998] Royce W, *Software Project Management A Unified Framework*, Addison Wesley 1998

[RUP] Kroll, Per, The Spirit of RUP, Rational Software 2001

[Schach 1999] Schach, R., *Software Engineering*, Fourth Edition, McGraw-Hill, Boston, MA 1999, pp. 11

[SE] Software Engineering http://manta.cs.vt.edu/cs5704/SEmodule/SE/Lessons/index.html

[SEI] http://www.sei.cmu.edu/programs/sepm/risk/risk.mgmt.overview.html

[Thayer] Thayer, R. H., *Software Engineering Project Management*, IEEE Computer Society, Second Edition, Chapter 3, 7

[Zagal 2002] Zagal, JP, Ahues, RS, Voehl, MN, Maintenance-Oriented Design and Development: A Case Study, *IEEE Software* July/August 2002