AN ANALYSIS OF PROBLEMS WITHIN THE SOFTWARE DEVELOPMENT INDUSTRY WITH AN EMPHASIS ON THE MYTHICAL MAN-MONTH BY ROBERT BROOKS

Anthony Mwaniki

Dr. Gary Schmidt

Computer Information Sciences Department
Washburn University

TonyM@parod.com
gary.schmidt@washburn.edu

ABSTRACT

The system development industry has fallen short of meeting its objectives in the last decade. Reasons include factors such as systems not delivered on time to being functionally obsolete by the time the system is delivered. The end result can be huge cost overruns with no new functionality to justify the cost.
This paper covers a broad overview of the reasons behind this phenomena and possible solutions. This paper is based on *The Mythical Man-Month* by Dr. Robert Brooks, published in 1975. This book addresses many of these issues ranging from "conceptual integrity", which is a key factor to determine whether or not a system will function as required and expected; to the elemental organization of a project team.
This paper also addresses the evolution of software design and development from the old ad-hoc styles to the emergence of object oriented programming design as a solution to the issues that face the industry today.

In 1994, the Standish group, (an international group of information systems professionals and specialists who have many years of hands-on experience in assessing risk, cost, and return for investments in systems projects development.) conducted a survey on over 350 companies about software projects to find out how the systems were fairing. The results were sobering. More than half of all programs/systems being developed fail. The survey seemed to suggest projects were even cancelled before the system was completed. Some were never used after they were done or they, quite simple, failed to meet the projects objectives and benefits. This was further compounded because of the half that was delivered, each ran into its own unique problems. The survey published a year later said "On the success side, the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget. And, even when these projects are completed, many are no more than a mere shadow of their original specification requirements. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars". (1)

These results were nothing new to the industry. Fredrick Brooks had already surmised this in his best selling and luminary book '*The Mythical Man-Month'*, first published in 1975. Brooks worked for IBM in the sixties and seventies developing large application systems at a time when system development was globally confined to the large corporate business entities (such as IBM) that could afford the costs associated these endeavors. In his book, Brooks offers a myriad of reasons why failures occurs. Chief among them, he argues, is that some systems are actually faulty from the get go. He uses the term 'conceptual integrity', which means that end result is a system that is easy to use because the objective of its designers have been fully met. He says that the purpose of writing a program is to make a computer easy to use. He writes, "Because the ease of use is the purpose the ratio of function to conceptual integrity is the ultimate test of a system design." (3). To further illustrate this point, he compares two systems (OS/360 and the Time-Sharing system for PDL – 10). Function, he says, has always been a measure for excellence for the OS/360 designers because it had the most functions. On the other hand, simplicity and spareness was the measure for excellence of the Time-sharing system. He argues that when both were held up for comparison, each was found wanting, obtaining half of its true goal. This, he reasoned, was because they were not straightforward. This in turn, compromises 'conceptual integrity', the objectives of both groups' systems designers. The end result did not fully manifest itself in the final product of both teams. He concluded that ease of use of the product by the end user shows that the conceptual integrity has been maintained throughout development. This argument holds true today. The Standish report survey mentioned previously indicated this to be true. Half the projects lost failed to meet their objective and requirements.

Another factor for failure, Brooks argues, is a "lack of calendar time." (3) This is a euphemism he uses in chapter two of his book to broadly refer to the time constraints and the overall problems that a particular system project encounters throughout its development life cycle. In this chapter, he says that "All programmers are optimists." (3) This reason alone has a more profound effect on the development of systems projects than most people give it credit. Programmers and developers have a natural human

instinct to think that they can do a project faster than they actually able to do so. Many assume that the project will be smooth and they do account for *certain* aspects of project design like implementation the system but fail to so for others. What effect will the new system (To – Be) have on the current application (As - Is) residing in that domain? How long will debugging take? Fixing a defect, he says, has a substantial chance of propagating other errors and defects thus lengthening the project time estimates. Because of this, he argues, most project stakeholders cannot accurately estimate how the project will progress and therefore make poor assumptions regarding the whole project progress from the beginning to the completion date. He further argues that when projects are late for whatever reason, the natural inclination of most managers is to add more people. He strongly discourages against this in what he simply calls Brooks law, "Adding manpower to a late software project makes it later." (3) This happens because adding more manpower means that tasks already assigned have to be repartitioned and assigned to others. This means that "some work already done will be lost and the system testing (semantic and logical) will have to be lengthened." (3) Adding extra people to a project also means that they have to be trained in order to bring them up-to-date and familiarize them with the status and goals of the project. The system development life cycle (SDLC) is today defined "as the process of understanding and determining how an information system project can efficiently support a business need; be designed effectively, built in the same manner, and delivered to its users in a timely manner in order to satisfy those needs." (4). The late delivery invariably results in huge cost overruns that make the project much more expensive than what was originally budgeted. More over, this late delivery may render the whole project obsolete. This occurs when the overall business need change during the time period it takes to design, build/develop and deliver the system to its client/s, end-user etc. This will invariably make it hard for a client/end-user to satisfy their client/customer needs. This can and often causes bad customer relations, which in turn, can prompt a client to take their business elsewhere.

Another reason of failure, according to Brooks, is that the technology base on which one builds a system, is inexorably advancing. This means that by the time one is ready to implement the system, it is delivered with fewer features than the end-user has to have in order to run their business and/or endeavors competitively and in a cost effective manner. This holds true today, for example, because of the rapid development of faster chips in recent years. The causal effect of this is the availability of robust, efficient and cost effective hardware components (hence the explosion of the PC market) leading to the dramatic reduction of software development costs. This has created an intense and competitive information systems market as the number of service providers has increased exponentially. Brooks notes that "more and more vendors offer more and better software products for a dizzying variety of application." (3) Limiting a project size because of huge memory costs is not a major issue today as it once was. Project complexity today, is of more concern to developers when it comes to project size as opposed to memory allocation.

Having come to these conclusions among others, Brooks puts forth a series of suggestions that have played a large role in the system development world and organization management in general. In chapter three, he suggests that one has to

assemble a team of expert professionals each, proficient in their field of expertise, in order to get a systems project done efficiently and on time to meet the users need. He endorses a proposal put forth by Harlan Mills (a fellow developer and authority in the software engineering field) that suggests assembling a project team with the same precision and preparation as 'a surgical team rather than a hog butchering team.' (3) Mills proposal supports the idea that instead of having many trying to solve a project in an ad-hoc manner, they should be organized in a structured manner from the chief programmer (referred to in this analogy as the head surgeon) to an administrator (who oversees the project budget, people contractual obligations etc). This is also true of any business entity that wants to successful and profitable. One has to have capable staff, experienced and well trained in their respective fields, to do so.

Having a capable staff, however, does not guarantee success if there is no communication among the project members. Brooks uses the Tower of Babel analogy to emphasize the importance of communication in any given environment. It is known that the building of tower (Babel) to Heaven project that man attempted, failed when God made Man speak in different languages. Brooks used this scripture analogy to illustrate the dire consequences of a lack of communication and organization in a software development project specifically and in a business entity in general. As the project proceeds, critical changes that are vital to the success of the project are not communicated clearly and effectively or none at all to the rest of the project team/organization. This, the author argues, causes disastrous delays in project development "because the right hand doesn't know what the left hand is doing.' (3) Brooks's assertions are true and are embraced by experts in the business world today. Stephen Robbins, an authority in organizational management, writes "no organization (in this case, project team) can exist without communication i.e. the transfer of meaning among its members. It is through the transmission of meaning from one person to another that information and ideas are conveyed." (2)

All these reasons have played an important role in the advancement of software design and the approaches that one uses to create systems. Different methodologies have been put forth to assist programmers and designers deliver their projects on time. Use of high level programming languages has led to improved productivity and more robust applications. Brooks endorses structured programming because "it frees up the program from much of its accidental complexity" by replacing the previous ad hoc and undisciplined approaches. Structured programming introduces the use of formal step-by-step processes that move the project logically from one phase to the next. The key advantage to the structured design is that "it identifies system requirements long before programming begins and it minimizes changes to the requirements as the project proceeds". (4) This approach, however, does have a key disadvantage. The design specifications must be completely specified on paper before the project begins and this can be very confusing for large projects because the information flow can often muddled.

The constraints in structured programming led to the popularity of rapid application development or RAD methodologies. This approach continues to be popular today. It attempts to deliver some parts of the system to its end user quickly. The initial system is

developed using the important and fundamental requirements (version 1). This is refined extensively and the issues identified are incorporated with new ideas to develop version 2. The process continues until the system is complete or no longer needed. This is called Phased development. In other cases, a prototype of the first part of the system that the user will use is quickly developed and implemented. The user and the development team then get together to refine it iteratively until both are satisfied. This methodology drastically reduces the project time and gets clients much more involved in the development of the system. Changes are cost effective and are much easier to make. This gives the clients a chance to better understand the system and enabling the client to suggest revisions that bring the system closer to what they expect. These methodologies take advantage of computer tools such as code generators and CASE tools. Their purpose is to improve the speed and quality of the systems being developed. However, the collective advantages of RAD methodologies are also their achilles heel. Their effectiveness can led to high user/clients expectations which can obscure what is actually possible.

As effective as the RAD methodologies are, the solutions they provide are constrained because they focus on a set of processes, how to do them and on the entities or objects in the user environment. These are hard to distinguish in a real world environment. Hence the increasing popularity of object oriented programming methodologies (OOP) that have been around since the sixties when an OOP language Simila was developed but were considered cost ineffective because of a lack of computational power in the computers of that era. Objects in OOP contain classes that define all data and process of each object contained in that class. This methodology allows relationships to exits between different classes, by encouraging encapsulation and information hiding (which Brooks was once against). Encapsulation combines the process and data into a single object. Information hiding requires that only the information needed to use an object should be made available to want to use it. These concepts are critical because changes made in a particular part of the system are not propagated throughout the system. OOP allows for inheritance, which means that classes can reuse attributes and methods defined in other classes. Since there are many similarities among the many classes in a system, inheritance allows the classes that are in a hieratical order to be used more efficiently. The degree of flexibility and tolerance that OOP methodologies provide, suggests that the future of system development is good. These methodologies are found in most of the innovations taking place in all sectors of the information system world. They are an integral part in the development of artificial intelligence. Their ability to decompose complex project into efficient and manageable modules allows the IS practitioners to think ‘outside the box’.

In conclusion, one can agree with Brooks the mythical man-month is a fact. Time and people are not interchangeable. Humans are prone to errors as they progress through their endeavors. With new technology, one can be much more optimistic than Brooks. With every new methodology, the industry inexorably continues to make progress in systems development.

References

1. Standish report (1996) www.standish.com

2. Robbins, S. (1993); "Organizational behavior", 6th Edition; Prentice Hall, New Jersey.

3. Brooks, R (1997); "The Mythical Man-Month", 7th Printing; Addison Wesley Longman, Inc., Reading Massachusetts.

4. Dennis, A., Haley-Wixom, B., Tegarden, D. (2002); "System Analysis And Design" 4th Edition; McGraw-Hill, New York.