World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:1, No:9, 2007

# The Challenge of Large-Scale IT Projects

Ahmet Denker



***Abstract*—**The trend in the world of Information Technology (IT) is getting increasingly large and difficult projects rather than smaller and easier. However, the data on large-scale IT project success rates provide cause for concern. This paper seeks to answer why large-scale IT projects are different from and more difficult than other typical engineering projects. Drawing on the industrial experience, a compilation of the conditions that influence failure is presented. With a view to improve success rates solutions are suggested.

***Keywords*—**Software engineering, software economics, project management, large-scale projects.

## I. Introduction

INFORMATION Technology (IT) is a much younger branch compared to the other branches of technology. It is 2005, and we are only a half-century into the history of the ITfield. At this stage, the most common problem is that the success rates of IT systems are disappointingly low. Industry reports show that the odds of successful completion drop noticeably with medium sized projects and disappear almost completely with large-scale projects. It has been common over the last decade to read stories of " IT failure" in both popular press and computing literature. Although these stories correspond to a tiny percentage of all the IT projects ever attempted they represent the seriousness of the problem. Unfortunately, failure is common, not the exception, in large-scale IT projects.

Why is there such a serious problem? Because the IT field has not made a conscientious effort to develop histories of past project experiences. Because the construction of large-scale software is very complex- it is the most complex task ever undertaken by human beings. Because it is invisible-it is all too common for both customers and developers not to sense its limitations. Lack of hisstory, complexity and invisibility make IT projects different from, and more difficult than other typical engineering projects.

As the IT industry continues to tackle problems that are forcing its constraints, it becomes even more important to analyze what go wrong in largge-scale projects.

## II. High Failure Risk High Price Tag

In his seminal book "The Mythical Man-Month" [1] Brooks correlated large-system programming with the mortal struggles of dinosaurs in the tar pits. Struggling to meet the goal, schedule and budget of a large- scale IT project, in fact, resembles the scenes of violently thrashing prehistoric beasts in the tar pits. Because of the lack of history, the amount of complexity and invisibility many large-scale projects share the unfotunate fate of ancient monsters.

The "Chaos" report of Standish Group in 1995 [2] showed that many large and powerful companies had become entangled in the stickiness of the problem. Only 9% of the large-scale IT projects were delivered on time and in budget. Large, medium or small, every company seemed to have been struggling against the grip of the tar. For medium and small sized companies the struggle was less fierce, however: the numbers for them improved to 16% and 28% respectively.

In 1999 and 2003, the Standish Group reported improvements. But the data on IT project rates continued to provide cause for concern. For example, a recent study carried out in the U.K. [3] showed that only 16% of the large-scale IT projects were considered successful.

Daily life experiences -of the " I am sorry I can't help you, the computer is down" sort – have been amplified by sensationalist journalism. Horror stories of collosal IT failures continued to hit the headlines. Failures are of course more sensational than successes, and the high profile failures significantly tarnished the reputation of IT industry:

- A Lockheed-Martin satellite went off course into space because there was a hyphen missing in one of the millions of lines of code.
- Boeing's Delta III rocket explosion was also caused by a faulty line of code.
- Ariane 5 prototype exploded due to software failure.
- Hong Kong's new Airport lost $600 million in its opening due to IT failure.
- Denver Airport Baggage Handling System turned out as a disaster classic.
- Libra IT system for the Magistrates Courts in the U.K. caused to embarrassment of the Government.

These are just a few of a number of sensational failure stories extracted from newspaper accounts. In fact, chairman of the U.K. Public Accounts Committee described Libra as "the worst IT project I have ever seen." What he was referring to was a large-scale IT project with a budget of over $500 million.

A recent review [4] estimated that a phenomenal $150 billion was wasted in 2003 due to IT project failures in the U.S., with a further $140 billion in the E.U. These survey results show that one thing common in large scale IT projects is failure, and the price tag attached to failure is unacceptably large.

## III. Why They Fail

Everyone seems to have been puzzled by the stickiness of the problem. It is hard to discern a way out of it. But we must try to solve this puzzle, because large-scale IT projects are central to the functioning of our societies in the Information Age. They are so pervasive that we can scarcely imagine life without them. They are also crucial to our economic growth. They create a large part of today's wealth and jobs. Why then are large-scale IT projects so susceptible to failure?

A large-scale IT project is a project with a large and complex software component, it consists of more than 100 people and the project schedule is over 3 years. It is the complex software component of these projects which make them different and more difficult than other engineering projects. "IT failure" generically refers to "software failure".

A comparison between large-scale projects and small-scale projects show how different can be success discriminators for the large project processes: large projects require substantial management overhead; performance is highly dependent on the skills of the management; project process maturity is essential; change management is necessary; maintaining consistency among the evolving artifacts is imperative.

### *A. Colossal Complexity*

From the beginning of a large-scale IT project, the odds for failure is higher than for success. Because extreme complexity is involved in it. Large-scale IT systems are among the most complex entities man ever built. Building a large-scale IT system can only be matched with the construction of Egyptian pyramids in complexity.

Take the IT system in a large-scale organization. Powerful workstations or personal computers sit on every desk. The computers are connected in a network. Teams of them are harnessed together to crunch away on a truly big problem. Mighty computers called servers support the network and manage the huge databases. Critical elements supplied by as many as 100 different vendors from Asia, America and Europe plug interchangeably into the system. Software component of such a system consists of several million lines of code which could fill thousands of books. And a little typo in one of these books would have the potential to crash the whole system. The vulnerability increases with size and makes failure easy.

There are other reasons for the complexity. Software construction is a people-intensive process. Consequently managing people has a profound leverage. The famous *Division of Labor Theory* does not lend itself easily. Division of a huge programming task among programmers does not necessarily mean sharing and reducing the load. Because software tasks cannot be easily partitioned and the required effort of intercommunication counteracts the benefits of labor division. Each part of the task must be separately coordinated with each other part, and the coordination effort increases as $n(n-1)/2$, where $n$ shows the number of parts. A task split into three parts require three times as much pairwise intercommunication as a task split into two. When $n$ icreases from 2 to 4 the required coordination effort is multiplied by 6.

The added complexity of intercommunation is worse when it comes to creating ties between talented software developers. They are as varied as they are smart and infamous for being *difficult people to manage*. They can be easily disillusioned. They would rather be told what to do than how to do it. This is a work style unsuitable to coordination. A talented software developer can be 10 times as productive as an average one, but the probability of retaining these exceptional people till the end of the project is very small.

### *B. Invisibility*

Sofware is invisible, the visualisation problem is what makes software management different and more difficult from any other engineering project management. This invisibility is the source of many IT project failures. Customer cannot have any underlying sense and may ask for functions that are impossible to deliver. Their inability to visualise the boundaries creates indifference to what is possible and what is not. This encourages people to change their minds more frequently than they might do for engineering projects where constraints are obvious.

Invisibility has the danger of creating a misperception that anything and everything is possible with IT. This makes both customers and developers susceptible to forgeting the limitations of IT. As a result, sofware engineers often take on risks far in excess of the limits accepted in other engineering disciplines.

### *C. Over-Optimism*

Software managers, especially the less experienced ones tend to assume that everything will go well, each task will take only as long as it should take. The act of taking on a large-scale IT project with such a tendency is, by very definition, an act of over-optimism. In a large programming effort with $n$ parts chained side-by-side or end-to-end, the probability that each will go well becomes vanishingly small. Murphy's law applies-"*if something can go wrong, it will go wrong.*"

Inexperienced managers neither have the experience nor the authority to negotiate with the upper management. Understandably, due to pressure from customer, upper management have a desire to shorten time and to meet external deadlines. It is almost impossible for the inexperienced manager to make a job-risking defence against such desires. The result is unrealistic and overly optimistic schedules.

Impossibly tight schedules have high stress levels attached to themselves. Survey results [5] show that stress is the cause of more than 40% of all software errors. These mistakes mean enormous amount of rework. Optimism eventually leaves its place to disbelief, and stress level spirals even higher.

### *D. Extreme Uncertainties from the Kickoff*

An upstream-downstream waterfall image is often employed as the benchmark of conventional software process. Like any other creative activity [6], the stages of programming can be collected into three groups: the idea, the implementation, and the interaction.

A program comes into existence as an idea, at the project outset, during the "upstream phase" . However, uncertainty is high at this phase due to vague customer requirements. In fact, 50 % of the requirements defined at this stage were found to be useless at the "downstream phase" of projects. Many projects fail due to flaws in requirement definitions.

Customers are usually poor in saying what they want, but very talented at saying what they do not want. Software developers suffer from changing requirements during the course of the project. " Nobody would force a builder to build the basement after having put on the roof, but in the software industry, that is common practice" [7]. This is a phenomenon known as "feature creep".

Requirements management is a concern throughout the project life-cycle. If customers do not understand the implications of changing requirements and asking for highly complex systems then the project is likely to be in red. Failure to strike a balance can and will lead to major slippages of time and money.

### *E. The Gulf between Best Practice and Common Practice*

Following best practices is a an exception rather than rule in IT industry. It appears that no other engineering discipline has such a gulf between best practice and typical practice.

IT professionals are required to be competent in appropriate areas. However, there is widespread complaint about lack of professionalism in IT industry. Projects are often poorly defined, codes of best practices are frequently ignored.

The breakneck speed of technological change and ferocity of commerial competition led to the triumph of short cut solutions over best practices. Extremely rapid progress of technological progress in IT emerges as an obstacle to professionalism. This pace makes it difficult for expertise in a particular technique or language to mature. A culture is estalished where the use of tools or solutions that are not yet proven is not only aceptable but also commonplace.

### *F. Rework*

The creation is not complete until the customer runs the program. Then occurs the interaction of customer with the mind of the creators. It is at this stage that the incompleteness and inconsistencies of the developers' ideas become clear. In every piece of a complex software there are embedded a number of assumptions. The large the number of these assumptions means some of them will prove incorrect at the interacion phase.

Thus "rework" becomes an essential and inescapable part of the process. Rework then takes time, money and sweat. The optimistic schedules are thorougly affected by rework. Furthermore the time required depends on the number and subtlety of the discrepancies encountered.

In the conventional waterfall model testing comes at the end of the schedule and with an unrealistically short time. Because of optimism managers usually expect the numbers of bugs to be smaller than they turn out to be. That's why, testing is usually the most mis-scheduled part of the process. Failure to allow enough time for system test is particularly disastrous. Since it comes at the end of the schedule. The delay at this stage is late and unsettling with severe financial and psycological repercussions. The project is fully staffed and cost-per-day is maximum.

Examination of IT project schedules show that few have allowed time for rework, but that most indeed spend half of the actual schedule for that purpose.

## IV. How to Solve IT

Many of the reasons of failure seem to adhere to already known reasons. Thus, one would think that a significant percentage of IT failures could have been avoided using techniques we already know. As stated in the Cobb's Paradox [8]*: "We know why projects fail, we know how to prevent their failure-so why do they still fail?"*

As a possible explanation for Cobb's paradox we can say that unfortunately practice in the management of large-scale IT systems does not seem to have kept up with the exponential rate of human ambition. There is a well documented good practice but it is all too rarely used. Commercial pressures mean that more and more complex systems need to be delivered in ever decreasing time-frames. As a consequence, success loses out in a trade-off against speed and complexity. Hence, there is a major software engineering challenge to deal with the irresistable rise in the demand for speedy delivery of increasingly complex IT systems.

Complexity in large-scale IT systems remains an area which is insufficiently understood. The degree of complexity of a particular project can be very difficult to estimate at the project kick-off. Projects may involve much more complexity than esimated at the outset. This makes large-scale projects extremely prone to failure. Research into better understanding and estimation of complexity is required.

Change management is critical in ensuring that excessive requirement alterations do not lead to lead cost and time runaways. Slippages of time and money associated with change proposals must be conveyed to the customer. Judicious use of freeze dates can help to control feature creep.

Due to over-optimism, many projects are undertaken on the assumption that the software will be or can be made to be perfect. In reality, rework often accounts for half of the software development budgets. Despite this, all too often testing and rework are left to the very end of the schedule. In fact, what will make a positive difference is to use the good practice that ensures code is developed and tested iteratively.

Most software experts agree that " if software development moves from a 'build everything yourself' to a 'assemble reusable products' model that would mean a major boost to success rate.

## V. Conclusion

"The Mythical Man Month," 20 years after its publication contains many perspective observations that remain disconcertingly relevant today. It is clear that lack of history, complexity and invisibility make large-scale IT projects prone to be stuck in tar-pits. Uncertainty of requirements, over-optimistic management, unrealistic schedules correlate inversely with performance.

The trend is getting increasingly large and difficult projects rather than smaller and easier. The demand is for building even more ambitious systems. There is some element of more complexity, more change, more uncertainty, greater difficulty everywhere. It is clear that we have to improve professionalism not only to improve existing perfomance levels but even to maintain them. Further research into sofware engineering methods and development of project management is in demand more than ever. Developments in understanding complexity adhered to scale will be needed to

meet the challenge of emerging large-system requirements on a scale never encountered before.

Our findings show that most of IT project failures could have been avoided using the best practices. In fact, a significant percentage of IT industry fails to implement the known best practices. In the light of increasing complexity of the IT sytems, the roles and responsibilities of IT professionals are becoming more and more critical and the need for adherence to best practice is of ever greater importance.

## REFERENCES

[1] Brooks, F. P., The Myhical Man Month, Addison Wesley1995.
[2] Standish Group, Chaos, 1995
[3] Sauer, C. and Cuthbertson, C., The State of IT Project Management in the U.K., Oxford, 2003.
[4] Darren, D. And Genus, A., Avoiding IS/IT Implementation Failure, TASM, vol. 15, No. 4, pp. 403-407, 2003.
[5] Glass, R. "IS Field: Stress Up, Satisfaction Down," Software Practitioner, 1994.
[6] Soyers, D., The Mind of the Maker, Harper San Francisco
[7] Hoch, D. J., Secrets of Software Success, Harvard Press, 2000.
[8] Cobb, M., Unfinished Voyages, A Follow-up to the CHAOS Report, The Standish Group, 1996.

**Ahmet DENKER.** B.S in EE, Boğaziçi Üniversity(1977); M.S. and Ph.D Sussex University, England (1978-1981). Dr. Denker worked as a full time professor at Boğaziçi University between years 1982 and 1999, gave lectures on Control and Robotics and supervised several theses. He was entitled "Chartered Engineer" (C.Eng.) by the Council of Engineering of U.K.. His particular areas of interest are applications of industrial control and Information Technology. He was a project manager at the Robotic Science Division of TUBİTAK (Turkish Scientific and Engineering Research Council) Marmara Research Center between 1992 and 1996 whereby he lead a team working on an industrial robot with visional capabilities. He held visiting positions in Sussex University (U.K.), Open University (U.K.), Eastern Mediterranean University (Cyprus) and Keio University (Japan). In 1996 he was awarded a medal by Matsumae International Organization of Japan for his contributions to science and peace. He acted as the General Manager of HAVELSAN Inc. between years 1996 and 2003. Through his leadership, Havelsan had undertaken critical responsibilities in initiating various e-government, IT and defense industry projects which elevated HAVELSAN to the top 100 defense companies world-wide as well as to number one IT company in Turkey. Dr. Denker is a full time professor at Ankara University since 2003 and giving lectures in Information Systems Engineering and Management.