A Note on Waste Management in Software Engineering (International Focus)

Dr. N.K. SRINATH
Professor and Dean
Department of Computer science & Engineering
R.V.College of Engineering, Mysore road, Bangalore-560 059

Abstract: - U.S. government and businesses spent approximately $81 billion on canceled software projects, and another $59 billion for budget overruns. Their survey claimed that in the United States, only about one-sixth of all projects were completed on time and within budget, nearly one third of a projects were canceled outright, and well over half were considered "challenged." Of the challenged or canceled projects, the average project was 189 percent over budget, 222 percent behind schedule, and contained only 61 percent of the originally specified features.[1] One reason for the varied conclusions is that most failed projects are never studied—even by the organization that experienced the failure. A failure is defined as any software project with severe cost or schedule overruns, quality problems, or that suffers outright cancellation. Some of the important points that are to be considered to avoid waste in software Engineering are discussed in this paper. This note is a kind of review article based on references considered.

I. INTRODUCTION

There are two important quotes - “A clever person solves a problem. A wise person avoids it.” Albert Einstein

“It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts”. —Sherlock Holmes

One needs to know where he is going before he set out; it’s surprising how many projects don’t have well-defined performance objectives. When we ask what the performance objectives are, we often get a response like: “As fast as possible.” An objective such as this simply isn’t useful. When defining performance objectives, one should not forget that needs may change over the product’s lifetime. For example, current performance objective may be to process 10,000 events per second. However, in two years, it may need to be able to process 30,000 events per second. It is a good idea to consider future uses of the software so that these can changes anticipated and built for the necessary scalability. This avoids waste of developed software.

As observed, waste management software is spoken with regard to software development to handle the waste data for various software product developments. The main goal of most of the waste management software is to give examples of how IT solutions developed under “best practices”; methodologies and have been successfully automated the waste management process. The basic waste of time energy, manpower and money in the software industry is less spoken.

Some of the most important examples regarding the waste of software are listed.

♦ In February 2003 the U.S. Treasury Department mailed 50,000 Social Security checks without a beneficiary name. A spokesperson said that the missing names were due to a software program maintenance error.

♦ In July 2001 a “serious flaw” was found in off-the-shelf software that had long been used in systems for tracking U.S. nuclear materials. The software had recently been donated to another country and scientists in that country discovered the problem and told U.S. officials about it.

♦ In October 1999 the $125 million NASA Mars Climate Orbiter—an interplanetary weather satellite—was lost in space due to a data conversion error. Investigators discovered that software on the spacecraft performed certain calculations in English units (yards) when it should have used metric units (meters).

♦ In June 1996 the first flight of the European Space Agency's Ariane 5 rocket failed shortly after launching, resulting in an uninsured loss of $500,000,000. The disaster was traced to the lack of exception handling for a floating-point error when a 64-bit integer was converted to a 16-bit signed integer.
A failure is defined as any software project with severe cost or schedule overruns, quality problems, or that suffers outright cancellation. Some of the important points that are to be considered to avoid waste in software Engineering are –

**Poor User Input**

Project failed because the system did not meet user needs. Failed project is waste of software, time, energy and money. Very common problem is acquirers the requirements from the client. The developers of the system receive most of their requirements from higher-level supervisors and so-called "users" who were not regularly using the existing system. Although "not invented here" syndrome has contributed to the system which eventual has lack of acceptance. The bottom line is that the system was inadequate for its environment.

**Stakeholder Conflicts**

It is observed that when there are deep incompatibilities among business partners, the project building of a system is cancelled at any expense, which may be litigious failure of the entire enterprise. Stakeholder conflicts can play many different roles in project failures. Some projects fail because the developers do not know who the "real" stakeholders are. Other projects, especially smaller projects within larger projects, never go anywhere because the internal stakeholders never agree on priorities.

**Vague Requirements**

Until the requirements are stable the design process cannot proceed. Projects could be headed for trouble if architectures and processes are not change-friendly, or if there are poorly established guidelines that determine how and when requirements can be added, removed, and implemented—and who will shoulder the cost of the changes.

**Poor Cost and Schedule Estimation**

Cutting a corner that undermines the entire foundation of the project should be avoided. If it is done, the project will cost more, take longer, and worsen the quality than would have been possible if a realistic schedule and budget are followed.

**Skills that Do Not Match the Job**

The technologists had to endure what is considered avoidable delays and mistakes because, decisions were being made by people with no technical expertise in the area but had all the authority. The larger the project, there is need for people with best knowledge, planning, organization, and communications skills; excellent technologists do not necessarily have these abilities.

**Hidden Costs of Going "Lean and Mean"**

To meet the requirement of the company, cutting jobs and expecting the same work with fewer people for less money is an impossible task. Often-intentional "dishonest pricing" of projects is often off by a factor of two or four or more, would call for never-before-seen levels of performance. Hiring unskilled people will mean that Software people are very unskilled clerks. It's an enormous productivity issue.

**Failure to Plan**

Software goes out of control if it is not well planned. Even though people are working hard, and have no plans in place, the project will not be completed in time with the same requirement. The overall performance is a waste at the end of the day.

**Communication Breakdowns**

Co-ordination with the project leader is required to materialize the overall performance of the software. When a big project is being tested by two different teams with no co-ordination, there would result in break down. Such problems are common on large projects, especially if people are working on the same project but at different sites. In many failed projects, "there is not even one person who has an overview of the whole project. The people working need to have a complete architectural view that fits one piece into another to obtain complete solution.

**Poor Architecture**

In any project if the system is working right, no one will ever realize it. If the system fails then the problem starts. Bad choices show up as long-term limitations aggravation, and costs. Architecture must allow for organization and mission changes. To give an example, a project completed with particular operating system work satisfactorily. The decision of this architecture was taking during the design phase. When the operating system is upgraded without concurrence with the software developer, the system fails. This calls for poor architectural design.

**Late Failure Warning Signals**

Lower level people are not approachable to their upper-level managers to convey that their edicts are unworkable before the project goes underway. Development cycles need to be adopted that allows the leader to know at the earliest possible moment to provide evidence that the project is not working.

It is also required to understand how not to work with software engineering to avoid - waste of software developers time and expertise tips are as follows:
1. **Absorb praise:** Expect your successes to be recognized even though executives will often attempt to spray awards across the entire team. Therefore take the spotlight and bathe in the attention.

2. **Deflect blame:** In software development, the only thing that goes wrong and is noticed is usually software. When software fails, a software developer is to blame. That's just logical. Make sure to redirect the accusations when they’re aimed at you, and to preemptively sow blame whenever possible.

3. **Don’t bother with the details:** Avoid minute details at all costs.

4. **Involve them late:** Software engineers write code, that’s what they do. They’re always fretting about how stuff is distracting them from their hacking. So why would you waste their time involving them in a project before it’s ready for coding? Bring them in once all of the strategizing and synergizing is done and all that’s left is the programming.

5. **Add process:** The best way to demonstrate value to the team is by introducing the right process. Rules will grease the wheels of progress. It is required to schedule update meetings, daily briefings, and all-day reviews. The process of keeping the engineers productive is by making them fill the status reports and cross-functional executive updates. The best mechanism is to report through emails.

6. **Never tell the reasons:** Engineers are highly analytical, and hence they need to be kept in the dark. Which means they take a less-sophisticated approach to decision-making? This maintains a distance between the actual facts and the process in progress.

7. **Commit for them:** The main job of the product manager to make commitments on behalf of the team. It is required for the team to adhere to the commitments made by the leader. This makes the leadership go high and would be more challenging. Commitment by the leader without consulting the team would put pressure on the team member to put more effort and get the result.

8. **Interrupt at any time:** The work that is provided by the leader is more important than any busy schedule that is mentioned by the worker. The leader need to take this point and interrupt the worker/programmer and force the required job to be completed in stipulated time. It is obvious that there are many ways to interrupt. Some such calls are by Chats, phone calls. But the best means is by personally intruding the worker by tapping on the shoulder. This process will have higher priority for the job that is given.

9. **Be ambiguous:** Being ambiguous will make the subordinate more concisions and hard working. It is better to be vague and imprecise as possible. You avoid any type of blame make sure that the information is conveyed orally without being written anywhere. Change you made base on the situation to make the work complete.

10. **They’re always lying:** The most important point to know is that in Engineering there is nothing called “impossible.” If anyone says so, they’re lying. Force the subordinates to work to achieve the goal by hook or crook. The Wright Brothers never thought that flying across the Atlantic was impossible! Assume a software engineer is always deceiving you and act accordingly. So when you hear terms like “technical debt” or “working from home,” you’ll be ready to call their bluff.

### II. CONCLUSION

The causes of failure are many. There are only a very few ways to succeed. The factors of successful project management have been documented for years. Following the norms of software engineering and avoiding petty means of reaching the goal causes failure. Failure of project calls for waste of time, energy and money. The best way to manage the system is to follow the software engineering approach.

### III. ACKNOWLEDGEMENTS

The author thanks Dr.S.Sridhar, Professor and Dean, Cognitive & Central Computing Facility of R.V.College of Engineering, Bangalore, India for communicating this note for publication.
REFERENCES


