The 7 Reasons Why Projects Fail

Technology White Paper

Why projects fail, a complete report inside

"The Roman bridges of antiquity were very inefficient structures. By modern standards, they used too much stone, and as a result, far too much labor to build. Over the years we have learned to build bridges more efficiently, using fewer materials and less labor to perform the same task."

Tom Clancy (The Sum of All Fears)

The Bridge and Software Analogy

In 1986, Alfred Spector, president of Transarc Corporation, co-authored a paper comparing bridge building to software development. The premise: Bridges are normally built on-time, on-budget, and do not fall down.

On the other hand, software seems to never come in on-time or on-budget. And this still holds true today.

In addition, it usually breaks down. Nevertheless, bridge building did not always have such a stellar record. Many bridge building projects overshot their estimates, time frames, and some even fell down.

One of the biggest reasons bridges come in on-time, on-budget and do not fall down is because of the extreme detail of design. The design is frozen and the contractor has little flexibility in changing the specifications.

However, in today's fast moving business environment, a frozen design does not accommodate changes in the business practices. Therefore a more flexible model must be used. This could be and has been used as a rationale for development failure.

But there is another difference between software failures and bridge failures, beside 3,000 years of experience. When a bridge falls down, it is investigated and a report is written on the cause of the failure.

Why Software Projects Succeed or Fail

This is not so in the computer industry where **failures are covered up, ignored, and/or rationalized.** As a result, we keep making the same mistakes over and over again.

The average cost of a development project for a large company is $2,322,000; for a medium company, it is $1,331,000; and for a small company, it is $434,000. A great many of these projects fail.

The Standish Group, a consulting and research firm, shows a staggering 31.1% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measured, but could easily be in the trillions of dollars

The most important aspect of the research is discovering why projects fail. To do this, The Standish Group surveyed IT executive managers for their opinions about why projects succeed. The three major reasons that a project will succeed are:

* user involvement,
* executive management support,
* and a clear statement of requirements.

There are other success criteria, but with these three elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically.

There is one final aspect to be considered in any degree of project failure. All success is rooted in either luck or failure. If you begin with luck, you learn nothing but arrogance. However, if you begin with failure and learn to evaluate it, you also learn to succeed. Failure begets knowledge. Out of knowledge you gain wisdom, and it is with wisdom that you can become truly successful.

Successes & Failures

So, what we want to find out is what makes a software development succeed, and what are the reasons for its failure. The real key we are exploring here is the requirements aspect of software development. So, let's first look at some of the different aspects of requirements before we discuss the reasons for failure.

Reason for Requirements

The initial aspect we want to explore is the reasons we even need requirements in the first place.

- Customers are good at problem statements, bad at solution statements.
- The product is initially approved for development based on the solution statements
- Requirements are the basis for project planning.
- Designers/coders are good at implementation, bad at mind-reading.
- Designers/coders need detailed statements from solution statements –that lead to requirements
- Use these requirements to test the product before delivery.

Now let's look at what we mean by a requirement.

What is a Requirement?

- A software requirement is a **capability** needed by the user, **to solve a problem or achieve an objective**. It must be met or possessed in the software in order to satisfy a contract, standard, or specification. [Dorfman]
- A **commitment** between the customer (user) and developer. It collectively represents an **agreement** between two (or more) parties.

    * The success of any project is dependent on the quality of its requirements to ensure the desired end product.
    * Requirements drive costs, schedule, skills, resources and operational procedures.

Next, let's define what we believe is a good requirement.

What is a Good Requirement?

- It Is a clear and unambiguous Statement of Need
- It faithfully reflects the source specification
- It Adds value for the user
- It Includes more depth than its source specification
- It makes sense
- It is traceable to a customer use / need
- It can be verified
- Describes what, not how

What are the Characteristics of a Requirement?

Requirements need to have the following characteristics to be effective and understandable. They must be;

- Unambiguous/Specific
  - For each requirement there exists only one interpretation
- Free from design detail
  - The requirements do not constrain the design unless necessary.
  - The requirements do not impose the "voice of the engineer" on the "voice of the customer"
- Verifiable
  - For each requirement, there exists a finite cost-effective process for checking if the system and/or software meets it.
- Traceable
  - Each requirement should trace backward to its source and forward to descendant documents.
- Complete
  - Nothing is implied or assumed
- Workable/Feasible
  - The requirement can be implemented given current technology and constraints of time and budget.
- Consistent
  - No requirement contradicts another (or an external interface requirement).
- Correct
  - The requirement reflects the user's and sponsor's needs.

If there is any doubt about the necessity of a requirement, then ask: **What is the worst thing that could happen if this requirement were not included?** If you do not find an answer of any consequence, then you probably do not need the requirement.

Types of Requirements

And finally, before we examine the seven reasons projects fail, we want to look at the different types of requirements. They are;

- Functional
  - transformations (inputs, process, outputs)
  - The software shall calculate the drive command based on the input position error of the selected sensor.
- Behavioral
  - dynamic aspects of the software system including control, timing states, modes, how the software reacts to internal and external stimuli
  - The system shall enter the ready mode after initialization is complete.
    - May require internal knowledge to verify
- Performance
  - Numeric values for measurable attributes of a system timing (execution time, response time, etc.)
  - The application shall return the response to the screen within 3 seconds after the enter key has been depressed.
- Operational
  - Specifies how the software will run and communicate with its human users (number/qualification of user, built-in assistance, number and size of external messages)
  - The software shall provide a help facility that describes the commands.
- User Interface
  - Specifies characteristics of the human/computer interaction (screen formats, menus, page layouts, etc.)
  - The terminal screen shall display error messages starting in row 1 character position 1.
- Interface
  - Hardware interfaces, software interfaces, communications
  - The system shall use Microsoft Windows XP or 2000 operating system.
- Adaptation
  - Requirements for adapting to site specific conditions and changes in the systems environment.
  - The system shall operate in batch or interactive mode.
- Data Requirement
  - Validity, accuracy, range, units, size
  - The software shall verify that the patient number exists in the customer database.
- Design Constraint
  - Design/development, environmental restrictions, etc.
  - The application shall execute in a partition of 64K bytes.
- Quality
  - Reliability, availability, usability, maintainability, portability
  - The system shall allow at most 3 errors during 1,000 CPU hours of operation.
- Safety
  - Safety of equipment, people or property
  - The software shall power down any externally connected equipment prior to powering down the system.
- Security
  - Controlling access to data and/or program execution
  - The system shall lock out the user after the user enters the wrong password three consecutive times.

Now that we have looked at every aspect of requirements, let's explore the seven reasons projects fail, in the requirements stage of development.

The 7 Main Requirements Problems

1) Poor Requirements Quality

Problem
In practice, the actual quality of many specified requirements is poor.

Poor requirements quality can mean many things, but here are some of the most obvious traits: ambiguous, incomplete, inconsistent, incorrect, out-of-date, not mandatory, irrelevant to the system being built, untraced, and in a form that is foreign to many of the stakeholders.

Many times this problem arises because the requirements engineers may be inadequately trained, have inadequate access to stakeholders and other sources of the requirements, and who are given inadequate resources or authority to properly engineer the requirements.

Consequences
Requirements engineering is the first engineering activity during which major mistakes can be made, and the negative consequences of these mistakes are felt during all downstream activities such as architecting, design, implementation, and testing. Poor-quality requirements greatly increase development and sustainment costs and often cause major schedule overruns

As long ago as the early 1990s, it was well known that defects discovered once a system is deployed cost 50 to 200 times as much to correct, had they been found during requirements evaluations.

2) Inappropriate Constraints

Problem
In practice, many requirements are not actually mandatory. Instead, too many of them are architecture, design, implementation, and installation/configuration constraints that are unnecessarily specified as requirements.

Because stakeholders, and/or requirements engineers, sometimes incorrectly assume that a common way to implement a requirement is actually the only way to implement the requirement, they confuse the implementation with the requirement and inappropriately specify *how* to build the system, rather than *what* the system should do or *how well* the system should do it.

Consequences

By unnecessarily specifying constraints, the requirements needlessly tie the hands of the architects, designers, implementers, and installers. This often prevents a better solution to the problem from being selected.

3) Requirements Not Traced

Problem
Although the value of requirements tracing is widely recognized, and it is often mandated in contracts, and it is included in many requirements engineering methods and training classes, many requirements are still not properly traced in practice.

The sources of requirements (e.g., higher level requirements, other documents, and stakeholders) are not documented. Similarly, requirements are often neither allocated to architecture and design elements nor to the test sets that verify them.

On many projects, the very large number of requirements makes requirements tracing impossible to perform manually, and difficult and resource-intensive to perform, even with a modern tool support.

Consequences
By unnecessarily specifying constraints, the requirements needlessly tie the hands of the architects, designers, implementers, and installers. This often prevents a better solution to the problem from being selected.

4) Missing Requirements

Problem
Midsized systems often have hundreds of requirements and many large systems can end up with several thousand separate requirements, especially when one considers the derived requirements that are allocated to subsystems and their subsystems.

In fact, many information systems often are specified to have numerous features that are not used by almost all users, and possibly not needed at all. However, overlooking such requirements is not what this problem is about.

The real problem is that many architecturally-significant requirements are accidentally overlooked. These are usually nonfunctional requirements, most commonly quality requirements specifying minimum acceptable amounts of some type of quality such as availability, interoperability, performance, portability, reliability, robustness, safety, security, stability, and usability.

Consequences
Because missing requirements are much harder to spot during requirements evaluations than incorrect or poorly-specified requirements, their absence is often missed until the system is integrated, undergoing operational testing, being manufactured, or being deployed.

5) Inadequate Requirements Management

Problem
Many projects do not adequately manage their requirements. They store their requirements in paper documents or in simple spreadsheets. Different kinds of requirements are also stored separately in different media, controlled by different teams. such as the marketing team, the management team, the requirements team, and specialty engineering teams.

Consequences

Requirements stored in paper form, rather than in a requirements repository, are difficult if not impossible to create, manipulate, and maintain. And scattered requirements are also hard to find, sort, query, and maintain.

6) Inadequate Requirements Process

Problem

On many projects, the actual requirements method used is largely undocumented. It is often incomplete in terms of either missing or inadequately documenting important tasks, techniques, roles, and work products. The as-followed requirements engineering process is often inconsistently followed, and significantly different from the as-documented requirements engineering method.

Consequences

Therefore, a poor documented method is enacted as poor processes produce poor products, which in this case are poor-quality requirements and requirements specification documents. Inappropriate methods are inefficient and ineffective.

When different requirements engineers and requirements engineering teams use poorly documented methods, they produce inconsistently specified requirements, which are difficult for architects, designers, implementers, and testers to use.

7) Inadequate Tool Support

Problem

Many requirements engineers do not have or do not use adequate tool support when engineering their requirements. For example, many requirements engineers still use a requirements specification document as their combined requirements specification and requirements repository, while others use a simple spreadsheet or relational database table.

Consequences

Many requirements models are not properly documented and stored to back up the actual requirements. It is extremely labor-intensive to manually produce and maintain a non-trivial amount of requirements. Without tool support, inconsistencies significantly increase and the documented requirements easily get out-of-date.

Conclusion & Recommendations

There are many things you can do to lessen the effects of these reasons for project failure. You can inject a strict methodology into your SDLC. This will put everyone on the same page, and bring some discipline to your project development.

You can also train your people, and ensure you have the best people possible in each SDLC job function. In particular, it is important the your Business Analysts (BA'S) understand how to take requirements and as importantly, understand the business side of the project they are working on.

However a very important aspect, of setting up, recording and managing requirements, is to have the right tool set that can allow this to happen. With the right tool set, you will be guided through the right steps to ensure that your projects do not fail. You can also cut weeks and months off the length of your projects by incorporating the best tools.

eDev Technologies has a solution that allows you to correct all of the above problems. Out tools – IdEA and inteGREAT – will ensure that your business users understand what they will be getting, right in the requirements stage. And you will capture and retain all of the knowledge necessary to ensure a successful project and its implementation.

For more information, please contact us at the listings below, or visit our web site.

Please contact Asif Sharif, our founder at;
www.edev.ca
asif.sharif@edev.ca
(416) 469-3131