
Chapter 1

Introduction

Objectives

- To explain why it is so important to improve the success rate of software development projects
- To outline the scope of this report and list the issues which are not covered therein
- To briefly discuss some additional aspects which are related to the subject of this report
- To introduce a common definition of projects and to define the differences between plain work, projects and programs
- To discuss the uniqueness of software engineering compared to other engineering disciplines such as civil engineering and ship-building

Contents

1.1	Problem Statement	12
1.2	Scope of this report	13
1.3	Additional Aspects	14
1.4	Definition of a Project	15
1.5	What makes IT Projects Unique	16
1.6	Key Points	19
1.7	Further Reading	19

1.1 Problem Statement

“Who would have thought that a new airport would be unable to open because of software defects? Yet, that is exactly what happened at the new Denver International Airport. Software is everywhere these days. Cellular phones, automobiles, dishwashers, telephone switches, credit card transactions, stock trading, power plants, medical equipment, packaged software—the world relies on software. And the quality of everything around us depends upon software quality. The time-to-market for everything around us depends on timely delivery of software components.”

- Fujimura A. (2000) -

This quotation demonstrates, that software development is no longer only an Information Technology issue; software and its related problems have found their way into daily life. The problems related to software and its quality also causes a huge economic impact. *The Standish Group (1995)* discovered that American companies and government agencies alone cancelled 80.000 projects, wasting about 81 billion dollars on those cancellations and incurred an additional 59 billion dollars due to overruns for completed projects.

Software development and maintenance projects have been troublesome for more than 50 years¹. The majority of software projects exceeded their original budgets, were delivered late or with reduced functionality. Many projects were even cancelled prior to completion. Often, software does not meet the real business needs of the customer. Frequent technology changes, the convergence of information technology and telecommunication systems and other influences make it even more complicated to succeed.

On the other hand software is very valuable and important. Modern business and daily life strongly depend upon computers and software. For example the daily airline reservation, air traffic control, trading transactions etc. would be impos-

1 1939: John Atanasoff produced the first prototype of an electronic binary computer
1941: Konrad Zuse introduced the first computing device, called “Z1”
1943: Colossus, an entirely electronic deciphering device, is operational
1944: Howard Aiken and IBM present the Mark I at Harvard University
1945: John von Neumann’s report on EDVAC describes stored-program computing
1946: Enivac is unveiled at the University of Pennsylvania
1951: Univac is delivered to the US Bureau of the census
1954: John Backus invented FORTRAN

sible without them. The development costs of a large software project are no less than the cost of building a skyscraper. However, in spite of its value, software projects are extremely troublesome as information technology is one of the most labour-intensive and error-prone fields in history.

The main question behind these facts should be: Are these bad records “acts of god” or are we able to improve significantly? Martin Cobb, a member of the Treasury Board of Canada Secretariat Ottawa, outlined the following famous paradox: *“We know why projects fail, we know how to prevent their failure -- so why do they still fail?”*

1.2 Scope of this Report

The chosen topic opens a broad field of possible investigation areas and related sub-topics. I therefore took the decision to focus on the primary concerns regarding information technology projects, which is software development projects for commercial use. No distinction between software for the mass market (e.g. Microsoft Office™) and bespoke application development regarding needs, strategies and recommendations was made. The topic “system integration” is only briefly discussed within this report, because a separate report might be written to cover this topic in detail. The following list of topics also belong to the area of IT projects but have different key issues and are therefore deliberately not covered within this report to keep the focus on the chosen issues:

- Software developed for the gaming industry
- IT projects which deal with hardware development
- Development of components for real-time environments
- Differences and similarities of software development projects performed in different countries and cultures
- Differences and similarities of software development projects, which are dependent on their size (small projects have other needs and problems than large ones)
- Influences and difficulties caused by the convergence of the IT industry and other industries such as telecommunication and automobile manufacturing

1.3 Additional Aspects

There are some additional aspects which will intentionally be excluded from this report although they are somehow related to the chosen topic. Discussing these issues in detail would be beyond the intended scope of this report. However, the most important surrounding aspects are listed in alphabetical order in the table below:

Topic	Description
Advanced Risk Management	There are special methodologies for advanced Risk Management. Details may be found in related books.
eXtreme Programming	eXtreme Programming is a new approach for software engineering and was developed by Chrysler. The main idea emphasises simplicity, testing, continuously reviewing the progress, peer development and so on. Some excellent results could be reached with this new methodology.
Joint application design	<i>Jones C. (2000): "Joint application design or JAD is a method for developing software requirements under which user representatives and development representatives work together with a facilitator to produce a joint requirement specification which both sides agree to."</i>
Unified Modelling Language and use cases	According to <i>Priestley M. (2000)</i> use cases provide a structured view of the system functionality. The main idea is to define a number of actors, which describe the different roles that users can play interacting with the system and use cases - a definition of the whole functionality of the system, seen from the user's view. Use cases are a central issue to the Unified Modelling Language approach.

Table 1.2 Surrounding aspects

1.4 Definition of a Project

The dictionary states that a project is something proposed or mapped out in the mind, as a course of action; a plan. *Kerzner H. (1997)* provides a more detailed view: “A project can be considered to be any series of activities and tasks that

- *have a specific objective to be completed within certain specifications*
- *have defined start and end dates*
- *have funding limits (if applicable)*
- *consume resources (i.e. money, people, equipment)*

Baker S. (2000) adds the following properties:

- *Every project produces a unique outcome*
- *Projects should follow a planned and organised approach to meet their objectives*
- *Projects usually involve a team of people to get it done*
- *Projects always have a unique set of stakeholders*

On the other hand

- responding to a customer’s requests
- meeting with colleagues to discuss a new procedure
- writing a report etc.

are not projects but might rather be defined as “plain and ordinary work”.

A program is another name for a recurring project. For example the development of the Windows™ operating system is rather a program, that is divided into several projects for every release (e.g. Windows95™, Windows98™ etc.). This recurrence happens predictably, but each cycle involves a new plan and a unique end product.

The above given compilation of projects, programs as well as the demarcation line between plain work, projects and programs might also be found in other relevant sources and is today common understanding. Therefore this definition is accepted and hence further used in this report without changes.

1.5 What makes IT Projects Unique

“Bridges are normally built on-time, on-budget, and do not fall down. On the other hand, software never comes in on-time or on-budget. In addition, it always breaks down. Nevertheless, bridge building did not always have such a stellar record. Many bridge building projects over-shot their estimates, time frames, and some even fell down.”

- Spector A. (1986) -

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

- The Standish Group (1995) -

Sommerville I. (1996) raises three further issues which distinguish software engineering from other engineering disciplines:

1. Software is intangible: Software cannot be seen or touched; project managers cannot see the progress.
2. There is no standard process: The engineering process for civil engineering or shipbuilding is well understood and has matured over the centuries; unfortunately this is not true for software engineering. *“We do not have a clear understanding of the relationship between the software process and product types”*
3. *Large software projects are often “one-off” projects: Rapid technological changes in computers outdate previous experience. Lessons learnt from that experience may not be transferable to new projects.*

Unfortunately there are some difficulties in addition to the above mentioned issues:

Problem area	Description	Comments
Complexity	Large software projects have often more than 10 million lines of source code. Dave Cutler, chief architect of the VMS and WindowsNT™ operating system, mentioned, that no single person is able to fully understand today's complex operating systems.	Details see chapter 2.4, "Theoretical Aspects" on page 51.
Missing common overall architecture	There is no common overall IT system architecture. Many different approaches were introduced within the last decades such as host, client-server, three tier model, Internet-based architecture etc. Non of them had enough time to mature.	Recommendations might be found in chapter 4.3, "System Integration Issues" on page 100.
Layered technology and open interfaces	<p>Today's IT systems consist of many different layers and sub-layers including hardware, firmware, middleware, communication layers, application layer, data access layer and so on. Many layers have to communicate via complex and open interfaces between each other.</p> <p>It is common practice for companies and end user to assemble those parts from different vendors and combine them to an overall system on their own. This seems to be unique to the industry. Could you possibly imagine buying a car from one vendor and then change the fixed configuration (e.g. the engine, the gear box, etc.) on your own, choosing any part you like from any other vendors?</p>	It is common practice in the PC industry that end users assemble their "individual personal computer" with hardware and software components from different vendors, which have never been tested or certified before.
Rapid technology changes	Rapid technology changes in nearly all components (hardware, software, development tools etc.) are usual nowadays. This leads to pre-mature components and conflicting release versions.	<i>Paulk M. (1995), Jones C. (1996), KPMG (1994)</i> and some other sources detect this situation as a special problem area for software projects.

Market pressure	The pressure to come up with totally new product features and functionality within a few months and with a competitive price is normal. The vendor, which introduces new software releases must first achieve a significant market share; governmental organisations often have to choose the cheapest offer. This often leads to “beta versioning” and error-prone systems.	Typical examples are the “Bid to win approach” according to <i>Vigder M. (1994)</i> , business pressure as described by <i>Boehm B. (2000) (2)</i> and excessive schedule pressure <i>Dillard J. (1997)</i> .
No common methodologies	Because of the relatively new engineering discipline, the frequent technology changes and some certain factors, there does not exist a common methodology of how to produce software. Different software development models – [SDLC] exist; all with advantages and disadvantages. Different methodologies, different education focus and conflicting acronyms make this situation even worse.	<i>Glass R. (1998)</i> argues, that the failure risk increases with the “buzzword quotient”, caused by different models and methodologies.
Mass market phenomenon	After introducing the “IBM compatible PC” in the early 80’s, computing devices became more and more popular. This led to a dramatic decrease of prices and margins causing cost saving programs on the vendor’s side – often correlated with declining quality and maturity.	See comments regarding “market pressure”
Convergence	The former “data processing” focused software industry is nowadays converging with many other technologies such as the telecommunication industry, automobile industry etc. This introduces additional complexity levels.	<i>Jones C. (1996)</i> , <i>Sommerville I. (1996)</i> , <i>Fujimura A. (2000)</i> and many other sources raise this issue.

Table 1.3 Software-specific problem areas

1.6 Key Points

- Unique objectives and outcomes, a defined start and end date, funding limits and specific resource requirements are the most important factors that distinguish projects from plain work and programs. These properties are common to software projects and for projects in other disciplines.
- Information technology and software engineering is present nearly everywhere in our technical driven society. Obvious examples are flight reservation and trading systems, but also cellular phones, dishwashers etc. are more and more dependent on software.
- Software development projects are often troublesome and very cost intensive. Cancelled and challenged projects have a huge negative impact on our economy.
- The relatively new technology, ignoring the experience from comparable software projects in the past, rapid changing environment, the complexity and intangibility of software, missing common methodologies and huge market pressure are the most important factors, which make software development projects unique and risky.

1.7 Further Reading

No special recommendations regarding further reading for this chapter. The most important information sources are already covered with the mentioned references.

Chapter 2

Project Failure

Investigation

Objectives

- To define the term “failure” associated with IT-related projects
- To describe the differences between successful, challenged and cancelled projects
- To provide an overview of the most important surveys and statistics related to project failures and their root causes
- To categorise the most important causes of failures
- To provide some background material regarding theoretical problems associated with software development

Contents

2.1	Project failure definition	22
2.2	Project failure surveys & statistics	25
2.3	Failure reason categorisation	48
2.4	Some theoretical aspects	51
2.5	Key points	53
2.6	Further reading	54

2.1 Project Failure Definition

2.1.1. Overview

The dictionary states “*Failure is a cessation of supply, becoming bankrupt or proving unsuccessful in business or in any profession or trade, neglect or non-performance; one who fails conspicuously or in some specific effort. Synonyms: loss, misfortune, neglect*”.

The above definition covers interesting aspects of project failure and related reasons such as unsuccessfulness, neglect, non-performance etc. The term suggests that failure prevention is possible, presuming that projects are monitored.

Let’s now move forward with a more project-related focus. A good explanation is supported by *Khafre (1998): In simple terms, a failure can be defined as “not meeting the stated project goals”*. We can see that the determination of failure hinges on the concept of a goal. In systems design, goals, which carry the connotation of desirable, are translated into requirements, which carry the meaning of essential. These requirements specify what must be implemented in order for the goals to be met. The success or failure of a software system often depends on whether key requirements are met.

These stated goals lead typically to a list of common project success criteria’s, which are: Complete the project

- within the defined time
- within the given budget
- with the specified content and functionality

A more comprehensive approach also includes aspects which lie beyond the project implementation phases such as

- Are the business goals met? (e.g. return on investment, market advantages etc.)
- Are the operational and maintenance costs within the desired range?

These aspects will be covered in more detail in chapter 3.1, “Project Success Definition” on page 56.

Mahaney R. (1999) introduces the idea of the degree of failure: “The more over budget a project is the more of a failure it is. Likewise, the less functionality delivered relative to the original scope, the more of a failure it is.”

Unfortunately the above described failure definition is somewhat ambiguous. However, it reflects today’s situation and will be therefore further used in this report without changes.

2.1.2. Failure Impacts

The next step is to define the possible impact of failures on projects. The following figure illustrates the main failure types:

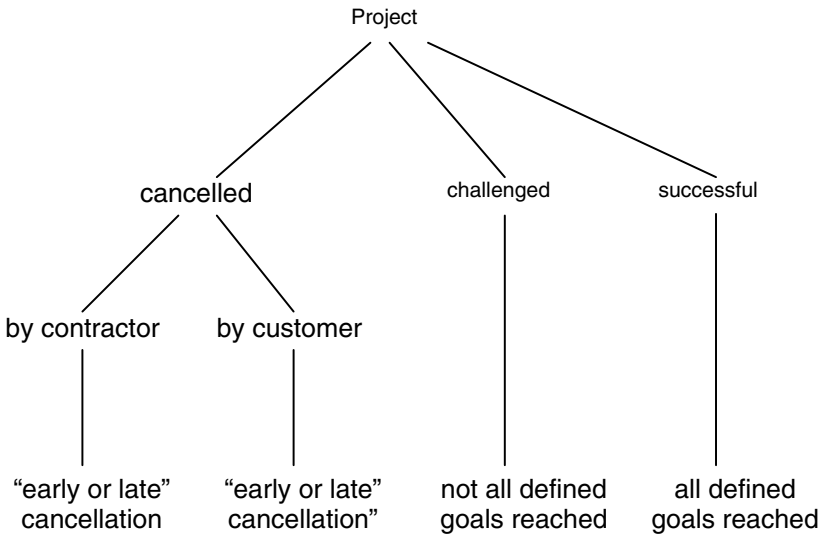


Figure 2.1 Project Failure types

Khafre (1998) distinguishes between “partial failures and total failures”; The Standish Group (1995) differentiates “challenged and impaired projects” and Jones C. (1996) uses the terms “relative and absolute failures” to formulate a distinction between challenged and cancelled projects.

Cancelled projects are terminated prior to completion; most often caused by extreme cost and/or schedule overruns, poor quality, delivered in an inoperable condition or significant litigation between the customer and the contractor.

Challenged projects are those, that are completed, but significantly¹ exceed the defined cost and/or schedule targets and/or the planned functionality has to be reduced.

Jones C. (1996) introduces an additional interesting aspect, the concept of recovery: “*Some projects that are heading toward potential disaster or failure can be turned around and can achieve acceptable or at least survivable results*”. This aspect will be covered in more detail in chapter 4.1, “Recovery options” on page 74.

2.1.3. Cancellation does not equal failure

An interesting issue is “early or late cancellation”. In most cases, cancellation is treated as a failure, but this must not always be the case. The main question should be “when is a project cancelled?”. *Moore J. (2000)* mentions, that one reason for unsatisfying success rates within the IT industry is, because projects are often cancelled too late: “*Building construction projects often fail early – they are cancelled in the proposal, design or planning stages – and therefore the failures are not as visible. Software projects, on the other hand, tend to fail conspicuously late.*” Also *Boehm B. (2000) (1)* stresses this point in his article “*Project Termination Doesn’t Equal Project Failure*”.

2.1.4. Failure Severity Levels

Unfortunately there is no common understanding or definition of how to define a failed project and no precise borderline between success and failure. However, there are some common terms, derived from *Glass R. (1998)* and *Jones C. (1996)*:

¹ Unfortunately there is no common definition regarding “challenged projects”. Further explanations see chapter 2.1.4, “Failure Severity Levels” on page 24.

Term	Definition
Relative failure	<i>Jones C. (1996)</i> provides examples for relative failures: “A software project that is delivered but exceeds anticipated cost and schedules by more than 50 percent or with a delay of more than six months”
Crunch mode	Crunch mode describes a project that has an extremely tight schedule. It speaks of the pressure being felt by the project participants. This term is often used by John Boddie in his book of the same name.
Death march	Death march is used to describe a project that has an almost impossible schedule. It hints at the oppressive smell of potential failure surrounding the project participants. This term is often used by <i>Ed Yourdon</i> in his book of the same name
Runaway project	Runaway is used to describe a project nearing or after its termination. The two most frequently used definitions for runaway projects are from <i>KPMG (1994)</i> - “A runaway project is one which failed significantly to achieve its objectives and/or has exceeded its original budget by at least 30 percent” and from <i>Glass R. (1998)</i> - “A project consumes close to double of its allotted estimated cost and/or time or more”
Disaster	The term disaster is often used, when a challenged or cancelled project causes serious threats to the particular company such as tremendous cost overruns, losing market share or even the risk of bankruptcy.

Table 2.1 Terms related to failure severity levels

2.2 Project Failure Surveys and Statistics

“Off the coast of the eastern seaboard in 1906 there were 177 shipwrecks. These shipwrecks were caused by accidents, bad management, carelessness, errors in judgement, and pilot errors - all of which can be attributed to human (as opposed to technological) failings. So too were the results of The Standish Group report on development project failures.”

- The Standish Group (1996) -

Because of the severity of the obvious problems associated with IT-projects, there exist a lot of different statistics regarding project failures. Unfortunately most of them are either performed by large software development companies for internal purposes and therefore under “non-disclosure” or do not represent the present situation.

However, there are three comprehensive statistics, based on either large surveys or extensive experience:

- A survey performed by *KPMG (1994)*
- Research performed by *The Standish Group (1995)*
- The book “Patterns of Software Systems Failure and Success” from *Jones C. (1996)*

These sources are referenced frequently on the Internet and in related books and reach the same conclusion. The table below provides an overview regarding the research methods, sample sizes and main focus.

Research and year	Method	Sample size and target audience	Main focus
KPMG research 1994	Telephone market research	120 respondents in the UK; large organisations from several industries	<ul style="list-style-type: none"> • Find out reasons for runaways • Discover how to avoid runaway situations in the future
Research by Standish group 1995	Research survey and personal interviews	365 IT executive managers from small, medium and large companies across major industry segments	<ul style="list-style-type: none"> • The scope of software project failures • The major factors that cause projects to fail • The key ingredients that can reduce project failures
Book by Capers Jones 1996	Assessment and benchmark consulting studies	Data from 500 enterprises and about 6700 world-wide software projects	<ul style="list-style-type: none"> • Find patterns of failure and success factors • Introduce a “6-stage improvement model”

Table 2.2 Method and relevance of the discussed failure statistics

Because of the different approaches and the importance of all the sources, each will be discussed in its own chapter. The data below are completely derived from the original research results. Additional surveys and statistics are introduced to provide a broader picture.

2.2.1. KPMG Report

KPMG (1994) performed two surveys; one in 1989 and a second in 1994. The latter one is discussed here, the first one is only used to watch for trends. The most important results are summarised by *Glass R. (1998)*:

Key Findings

- Many of the runaway projects were “overly ambitious”
- Most of the projects failed from a multiplicity of causes
- Management problems were more frequent than technical problems
- Schedule overruns were more common (89%) than cost overruns (62%)
- The use of packaged software did not help in reducing the incidence of runaways (24% were custom software; 22% were packaged software)
- Runaway projects showed the problems early in the project (25% during the initial planning and more than 50% during development)
- 55% of the runaway projects had not performed any risk management; half of those who performed risk management did not use the risk findings

Trends:

- Companies were much more reluctant to discuss runaway projects in 1994 than they were in 1989
- Technology is an increasing major cause of runaways (7% in 1989 versus 45 % in 1994)

Extract from the most important diagrams:

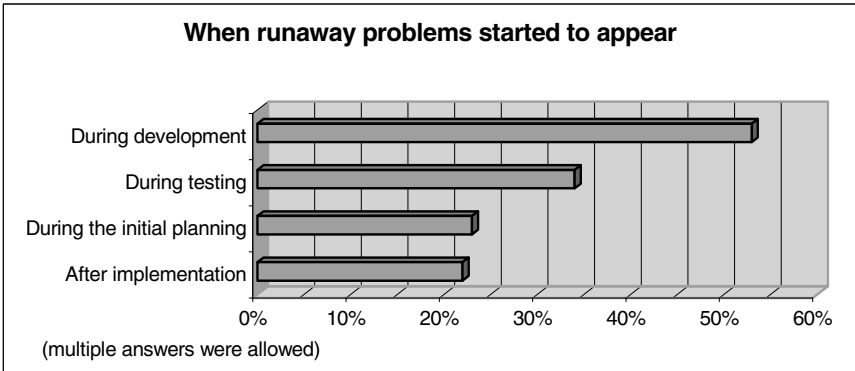


Diagram 2.1 When runaway problems started to appear

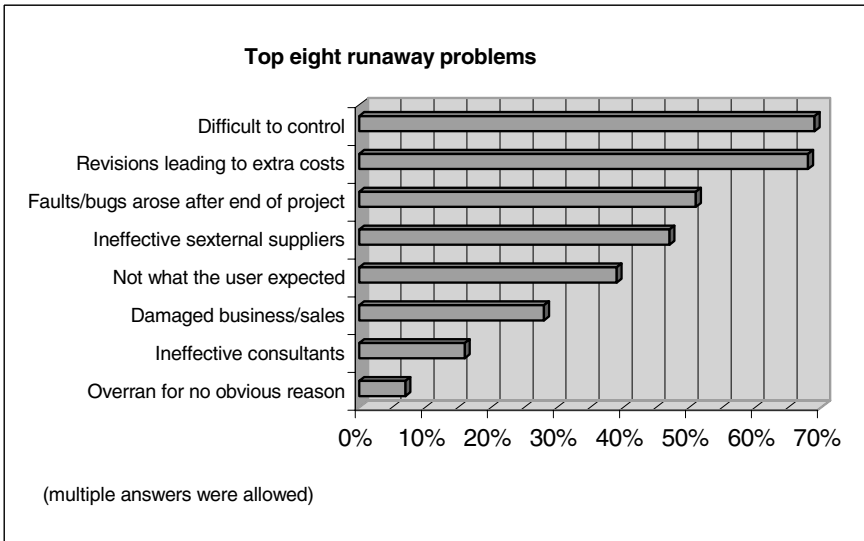


Diagram 2.2 Top eight runaway problems

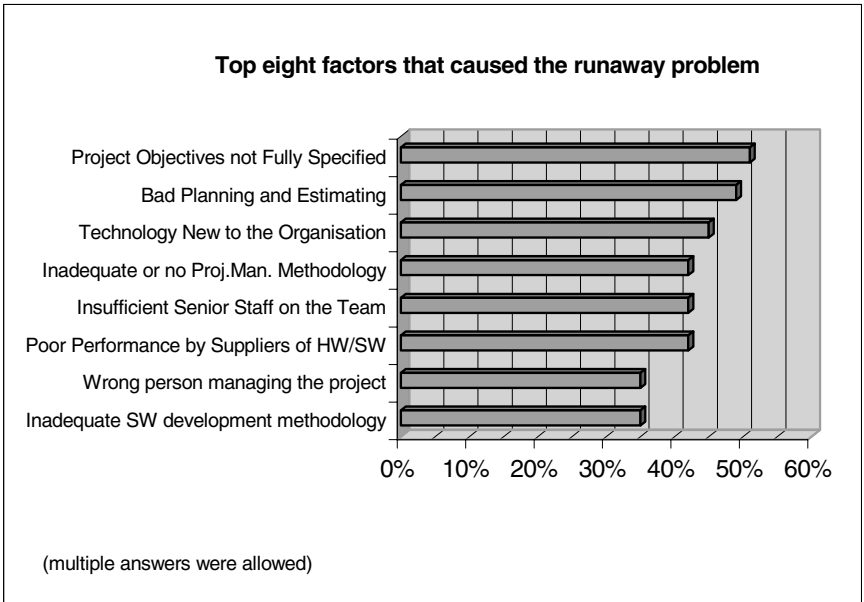


Diagram 2.3 Top eight factors that caused the runaway problem

2.2.2. The CHAOS Report from The Standish Group

The key findings, directly derived from the research, are:

- 31% of projects will be cancelled before they ever get completed
- 53% of projects will cost 189% of their original estimates; not taking subsequent cost and business damages into account
- The average success rate of projects is only 16%; in larger companies even worse with only 9%
- Projects completed by the largest American companies have only approximately 42% of their originally-proposed features and functions
- Smaller companies do much better; a total of 79% of their software projects will get deployed with at least 74% of their original proposed features and functions
- A high percentage of executive managers believe that there are more project failures now than five years ago and ten years ago. This despite the fact that technology has had time to mature.

Visualisation of the most important survey results

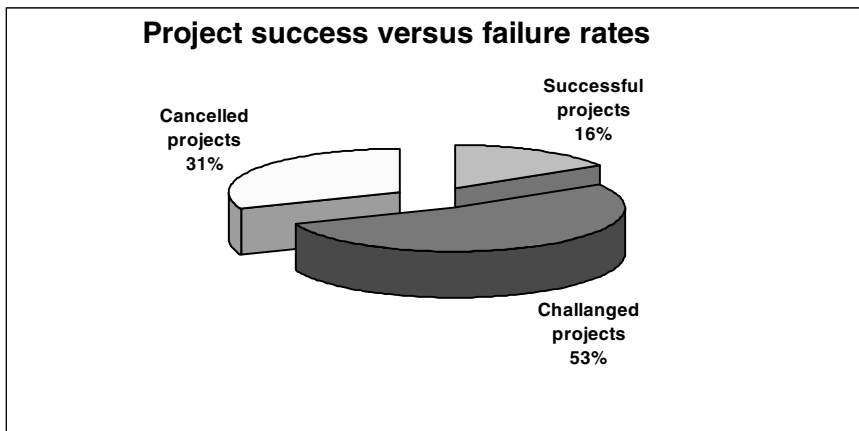


Diagram 2.5 Project success versus failure rates

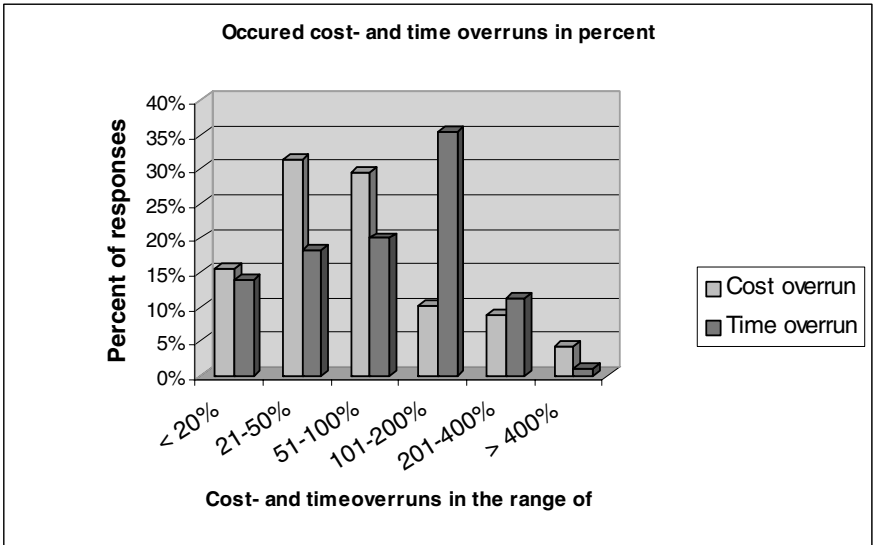


Diagram 2.6 Cost- and time overruns in percent

Explanation: This diagram shows how frequent and in what magnitude cost- and schedule-overruns occurred – e.g. 30 percent of the respondents (Y-axis) claimed, that they had to face a cost overrun of between 21 and 50% (X-axis).

Comment: One of the major causes of cost and schedule overruns are project restarts. For every 100 projects that start, there are 94% restarts; some of them have even more than one restart. The average time overrun is 222%; the average cost overrun is 189% of the original estimate.

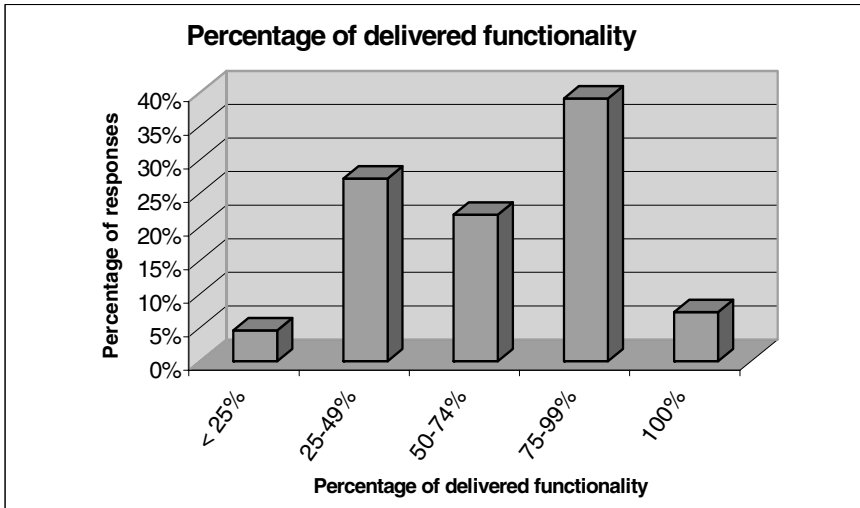


Diagram 2.7 Percentage of delivered functionality

Comment: For challenged projects, more than a quarter were completed with only 25% to 49% of originally-specified features and functions. On average, only 61% of originally specified features and functions were available on these projects.

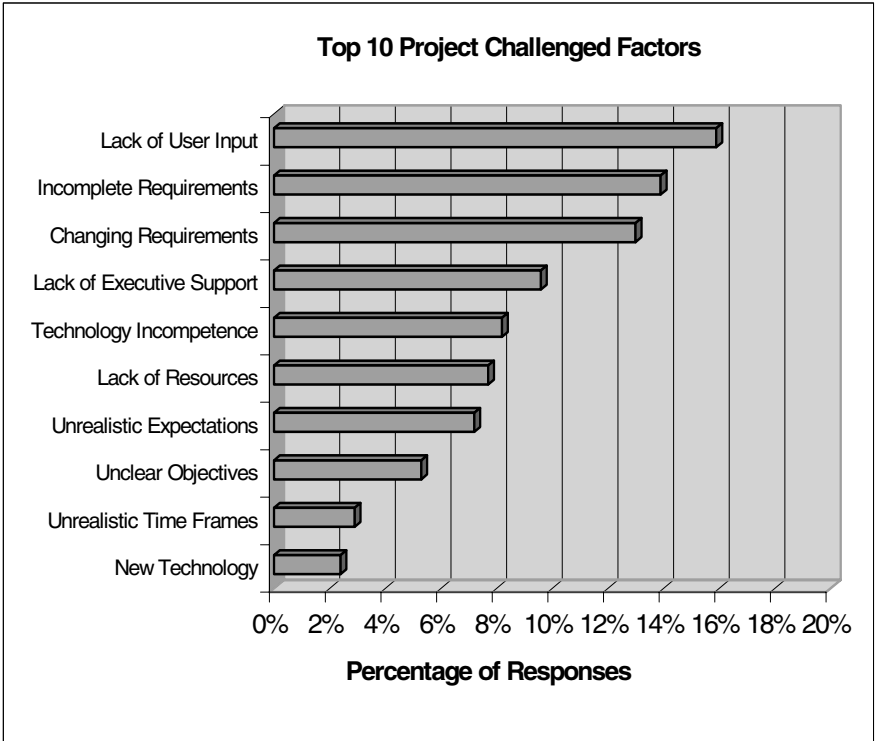


Diagram 2.8 Top 10 Project Challenged Factors

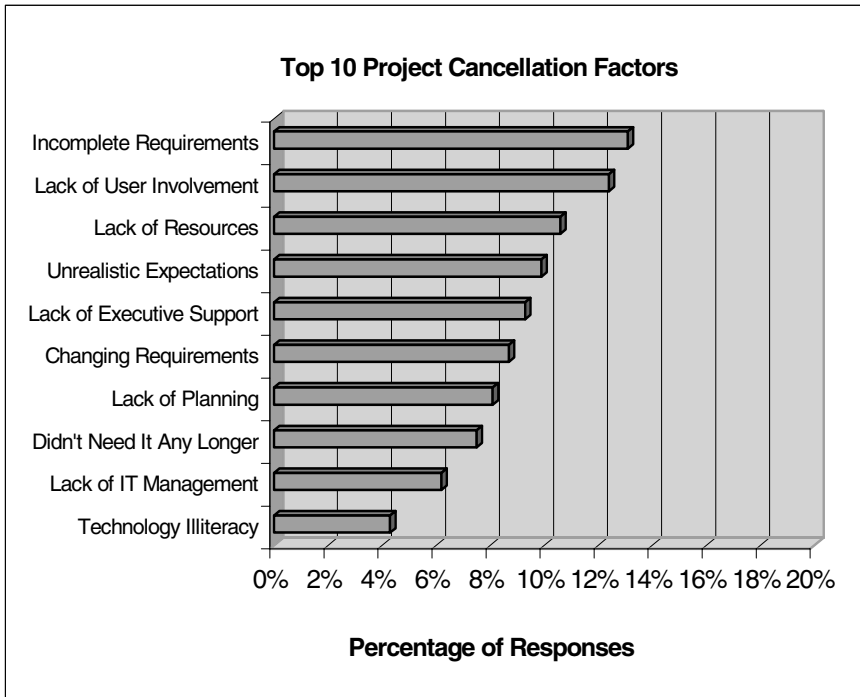


Diagram 2.9 Top 10 Project Cancellation Factors

2.2.3. Research performed by Capers Jones

The research performed by *Jones C. (1996)* regarding project failure and success patterns is the most detailed and most comprehensive. Capers Jones' research is mainly based on two principles:

1. The project size & complexity is measured using the [FUNCTION POINT] metric
2. Jones divides the software development into six sub industries

Function point metric

One [FUNCTION POINT] corresponds to about 105 Cobol or Fortran statements and equals about 125 C statements.

Explanation of the six sub industries

Subindustry	Description
Systems Software	Software that facilitates applications software, such as the operating system
Military Software	Software built to military procurement standards
MIS - Management Information systems	Business systems, the most common of Software applications
Outsource Software	Software built for, rather than by, the enterprise that needs it
Commercially Software	Packaged Software, such as spreadsheets and word processors
End-User Software	Software built by the intended user rather than professional software producers

Table 2.3 Six major classes of software projects