Software Disasters

Brian Hallesy

Introduction

- What makes a project fail? How often do project failures happen? And, how can they be avoided?
- This presentation will begin by discussing the first two questions, with a coverage of project failure statistics and an analysis of some examples of high-profile software disasters.
- It will then proceed to the root causes of software failure, and discuss what might be done to address the critical third question.

Outline

- Software project statistics and discussion
- High-profile software disasters
- Common causes of failures
- Possible solutions
- Conclusions

Software Project Statistics

- The Standish reports find that 15-30% of projects ultimately fail, with as many as 50% becoming challenged and only about a quarter finishing in time and within their budgets.
- Costs of software project failures can be massive. About $70 billion per year is wasted on failed IT projects.
- Why does this happen?

What causes project failure?

- Just as there are countless examples of failed projects, there are countless potential causes for project failure.
- Following are a number of high-profile project failures and the reasons they failed.
- After this, the common causes for failure will be analyzed and potential solutions will be discussed.

McDonalds' "Innovate" Project

- Project goals:
  - Create a real-time global network linking over 30,000 stores in 121 countries.
  - Enable executives to effectively and consistently manage and operate stores moment by moment.
- Project budget:
  - Innovate's budget was $1 billion
  - Innovate began in 1999 and was cancelled in 2002, having used $170 million of the budget.

Innovate: What went wrong?

- Not enough scope management
  - A real-time global network was impossible at the time due to countries having differing IT infrastructures.
- Lack of customer involvement
  - The store managers who would use the system were not asked for sufficient feedback, which would have allowed project team to reevaluate the scope.
- Unrealistic expectations and lack of expertise
  - Project was conceived by executives who had a good vision, but who lacked understanding of IT.
  - There was no project lead with sufficient technical expertise to realize that the goals could not be reasonably met.

FBI’s Virtual Case File

- VCF was contracted by the FBI to Science Applications International Corp.
- Project goals:
  - Modernize the FBI’s suite of investigative software applications.
  - VCF was intended to replace the FBI’s Automated Case Support system.
- Project budget:
  - VCF was worked on for 5 years between 2000 and 2005, and ultimately cost the government nearly $170 million.

VCF: What went wrong?

- Changes in requirements and specifications
  - Even as the project was falling behind schedule, requirements were added and specifications were changed; this was aggravated by the high turnover rate of managers.
- Lack of skills
  - Many FBI personnel with little experience with computer science were included as managers or even engineers.
- Lack of planning and testing
  - The FBI initially adopted a faulty “trial and error” approach to development.
  - They also neglected testing in the system, leading to a horribly buggy and ultimately unusable finished product.

Cigna's CRM system

- A Customer Relationship Management system upgrade planned by the Cigna health insurance corporation.
- Project goals:
  - Consolidate and upgrade Cigna's antiquated IT systems.
  - Provide a better CRM system to save costs on customer service.
- Project budget:
  - The initiative cost $1 billion. While the new system did not get scrapped, it had huge problems with glitches and lack of customer support, leading to a 6% decrease in membership and a 40% decrease in stock value for Cigna.

Cigna: What went wrong?

- Hurried execution
  - Due to legal issues and promises that had been made, Cigna rushed the development schedule and sacrificed quality.
- Lack of testing
  - Due to the hurried schedule, testing was also neglected and the new system had a number of issues which had not been noticed during development.

Cigna: What went wrong? (2)

- Poor personnel management
  - In anticipation of the new system cutting costs, Cigna cut off a number of customer service reps, while hiring some newer ones at lower pay.
  - When customers called about issues with the system, they were put on hold for long periods of time, and many of the reps were not helpful as they had little experience with the system.

Advanced Aviation System

- An IBM project, commissioned by the Federal Aviation Authority
- Project goals:
  - Modernize the system used by the FAA to keep track of what planes are in the air
- Project budget:
  - The AAS project began in 1981 and ended in 1994, spending 14 years and $3.7 billion without a single piece of software to show for it
  - At its peak, the project employed over 2,000 people

AAS: What went wrong?

- Too much conceptual development
  - Far too much time was spent in the first two areas of traditional life cycles, Requirements and Design.
  - This can give the false impression that more progress is being made than is actually happening.
- Changing requirements
  - The project's budget and fear of failure led to a great deal of watchfulness by management, endless meetings and constantly going back to the requirements phase to fine-tune things.
  - Ironically, the drive to get everything right caused previous work to be thrown out and was a large part of the reason that there was nothing to show after over a decade.

FoxMeyer's Delta III Project

- An ERP (Enterprise Resource Planning) project worked on by FoxMeyer Drugs.
- Project goals:
  - The goal was simply to use technology to increase efficiency.
- Project budget:
  - The Delta III project ran from 1993 to 1995 at an expected budget of $35 million.
  - It ended up costing over $100 million and driving the company to declare bankruptcy.

Delta III: What went wrong?

- Reliance on consultants
  - When there is a lack of skills, outside consultants can be useful to bring necessary knowledge to the team. However, FoxMeyer did not have the consultants train their existing employees.
  - When the consultants left, there was not enough knowledge remaining to maintain the system.

Delta III: What went wrong? (2)

- Hurried execution
  - Foxmeyer had a goal of implementing the system in 18 months, which was unrealistic for the size of the project.
  - As goes without saying, rushed development hurts quality.
- User involvement issues
  - Possibly an unavoidable consequence of the project; since the users of the system would be warehouse employees whose jobs were threatened by it, there were serious morale problems.
  - Disgruntled employees would make mistakes or damage inventory.

Causes of Software Disasters

- Lack of planning/Unrealistic estimates
  - Insufficient planning can lead to a number of problems; if the problem is not researched enough, the scope is likely to be underestimated.
  - Without a realistic appraisal of a project's scope, the project is likely to run past deadlines and over budget.
- Unskilled or inappropriately skilled team members
  - Team members are not familiar with the tools or the environment of the project. This slows the project down and can cause it to be developed in the wrong direction (resulting in a product which does not fulfill the client's needs).

Causes of Software Disasters (2)

- Communication within team
  - If team members work on different areas of code and make different assumptions, things will break during integration testing. If integration testing was not performed, the assumptions can cause the product to fail spectacularly.
  - Furthermore, ‘code ownership’ makes it difficult for team members to help each other and can cause confrontations, both slowing down development.

Causes of Software Disasters (3)

- Communication with customer
  - Cause of numerous problems. As mentioned before, lack of communication at the planning phase could lead to wrong ideas about scope.
  - Lack of communication during development can lead to the completely wrong product being developed if the wrong assumptions were made about the customer's needs.

Causes of Software Disasters (4)

- Lack of testing
  - Self-evident. If the project is not tested comprehensively, the released software will be buggy and likely to fail.
- Changing objectives
  - As time passes, the environment and the customer's needs will change. Changes and additions to the team's objectives can set development back significantly.

Causes of Software Disasters (5)

- High turnover rate
  - Projects going through many different team members will be slowed down significantly as new members will need to be trained before they can help on the project.
- Running out of time/budget
  - Failed projects are always those which have run past deadlines and went over budget.
  - When deadlines are approaching, it is easy to dig yourself deeper into the hole by:
    - Working overtime
    - Throwing more team members at the problem
  - Working overtime exhausts the team and impacts performance. Adding manpower sounds like a good idea, but will actually slow down the project for the same reason high turnover rate is an issue.

Possible Solutions

- Lack of planning/Unrealistic estimates
  - Spend time at the beginning of a project meeting with the customer and collecting user stories. Make sure that the time and budget given will be sufficient for the scope of the project.
- Unskilled or inappropriately skilled team members
  - Evaluate your team beforehand with a method like the inception deck. Make sure that the team contains members with the skills necessary to make the project succeed. A customer team member would also help to bring expertise to the project.

Possible Solutions (2)

- Communication within team
  - The agile principles of shared ownership and an open workspace can both help with this.
  - An open workspace encourages communication among the development team.
  - Shared ownership reduces fights among developers (your code broke my code), and increases overall knowledge among the team.

Possible Solutions (3)

- Communication with customers
  - Careful collection of user stories during the planning phase will help create reasonable estimates of deadlines later on.
  - A short iteration development cycle will allow the team to present customers with working code.
  - The customers can offer feedback on the features which have been implemented, redirecting the team if incorrect assumptions had been made.

Possible Solutions (4)

- Lack of testing
  - Develop a comprehensive test suite, with both customers and developers contributing.
  - System tests to determine you are building the thing right, acceptance tests to determine you are building the right thing.
- Changing objectives
  - A flexible, agile approach to development helps considerably here.
  - As opposed to a traditional life cycle which would have to deal with overhead updating documentation and re-evaluating, an agile team already works with short cycles that each tackle small decomposed objectives and will thus be much less impacted by changing objectives.

Possible Solutions (5)

- High turnover rate
  - The obvious solution is to keep turnover rate low. Assemble all of the team members you are going to need before the work begins, and maintain good working conditions.
- Running out of time/budget
  - Agile methodology suggests that at this point it is best to compromise on scope.
  - Meet with the customer and determine which functionality is most important and what could be expendable.

Conclusions/Avoiding disasters

- As mentioned before, there are countless different ways for projects to fail.
- The most common causes of failure are probably these:
  - Lack of communication
  - Incorrect estimates (budget/time)
  - Inflexibility
- Any one of these issues can cause a software disaster by themselves, depending on the circumstances.

Conclusions (2)

- These problems are all endemic of traditional life cycles.
- Indeed, almost every high profile software failure I researched was a project which had employed traditional development.
- As well, most of the possible solutions I brought up to potential causes of failure are policies present in agile methodology.
- Is agile the solution then?

Conclusions (3)

- Agile development is not a shield which will make your project immune to failure.
  - Many agile projects also fail, especially when the agile principles are not followed correctly.
- Agile life cycles do not prevent software failure altogether, but they do prevent software disasters!
  - In particular, the agile method of delivering working code immediately lets us detect troubled projects very early on.
  - The projects can then be reevaluated or scrapped.
- An agile project will fail at the beginning, not at the end. This crucial difference will help us avoid those high-profile budget consuming disasters that have plagued companies.

Executive Summary

- Software failures occur frequently, and there have been many cases of extremely expensive high-profile disasters.
- There are countless possible causes, a number of these rooted in problems found in traditional life cycles.
- Agile methodology cannot absolutely prevent failure, but it can reduce failure rates by addressing many of the possible causes of failure.
- Also, agile development can prevent occurrence of high-profile software disasters by providing early detection of software failures.