Contributors to SW Project Failure - Survey, Conjecture, and Preachy Advice

This presentation has been given to 46 companies and organizations, in 8 countries. No visible effect on the world economy has been noted.

How Do Software-based Projects Fail?

- Unhappy Customer – do not get what they expect or expect what they get
  - Ran out of time and $$
  - Unhappy user
- Safety compromised, including death
- Technically inadequate or over-adequate
- Does not contribute to the company business case
- Menace to Society – student favorite

Preachy advice to students #1

- Never lose the customer’s point of view
- Never lose the user’s point of view
- Never lose the business point of view
- Never lose the team-member point of view
**Sometimes we risk schedule**

- 85% of SW projects are either late or delivered under-spec.  
  *source: SEI Web Site*

- Microsoft no longer names their products for the release year. Um, why?  
  *source: my computer*

**Sometimes we risk budget**

- If the entire budget for the Denver Airport Automated Baggage System had been converted to cash, it could have paid wages for a manual system for 1000 years.  
  *source: Modern Materials Handling Magazine*

**... we lose money**

- ATMs in Mexico City were processing debits as credits, but only at the user account level (i.e. bank and machine settlements were accurate)
  - lines at the ATMs sometimes totaled 1200 people
  - ATM use increased 400%
  - users managed to keep it an underground secret for 2 weeks
  - misappropriated credits totaled $4 million, a small percentage was never recovered  
  *source: I was there*
A Classic Error

what they wanted:
if (A == B)
{
    B = C;
    return;
}

what they wrote:
if (A = B)
    B = C;
return;

what they got:
if (A == 1)
    { 
        B = C;
    }
return;

 sometimes we are too eager to technologize

The Puget Sound ferry system had been using mechanical and hydraulic controls since 1875.
After converting to computers and electronics, it began bumping docks, switching from forward
 to reverse by itself, injuring passengers, and dumping cars, and caused more damage in its
next 90 days than in its previous 100 years.

source: Software Woes
Sometimes we risk function

- Long distance services for every state east of the Mississippi went down for two days due to a software change made 1 day before shipping a switching computer, that had just passed 19 weeks of tests.
- The change was made to the machine code, in 1s and 0s, without benefit of source code and compilation.
- The manager had to appear before a Congressional committee (Committee on Common Sense?)

source: this guy was later my boss

---

Killer Apps

- The Therac 25 cancer radiation system killed 4 patients on the table, due to a software error that misjudged the appropriate dosage level based on the position of a mechanical filter.
  - The company first regarded it as a user issue, because it was precipitated by an odd start-up sequence
- A fuel-saving computer on the DC-10 attempted to save fuel by cutting the engines during landing.
  - The pilot restarted manually in time.

source: Software Woes
Computer and human interaction

- At Chernobyl, technicians tried to run an experiment to feed power to backup generators, as the system was shutting down (combining two operations into one sequence).
  - Technicians needed to send control rods into the reactor core (usually a “run” process), during shutdown.
  - The control system was not programmed to do anything during shutdown, except shut down.
  - Control rods did not respond, power spiraled, the core overheated.

  source: Inviting Disaster

Killer Apps

- An intravenous medication pump ran dry and injected air into the patient.
  - the design RELIED ON manual intervention

- A digital display combined the name of one patient with medication information from another. Staff thought they had mis-medicated the patients.

- A heart monitor was designed to sound an alarm when the patient’s heart rate dropped below a certain threshold, but shut off when the heart stopped, thereby missing entirely any cardiac arrest.

Technology Misapplied

- The first robot to kill a human happened 25 years ago, and 4 times since (in industry).
  - robots are primarily software

- GM’s first attempt at robotic assembly resulted in 100% robot failure within 40 hrs. The solution?
  - “As much of a failure as a bug.”

  source: Scientific American “The Mechanization of Work”
Survey of Working Software Developers

“Give the top ten reasons why software projects fail”
- 472 students, representing 104 companies.
- 28% management
- 61% contributing staff
- 11% full time students
- No item was included that did not have 5+ mentions in the survey.
- Categories include People & Management, Process, and Style
- Only slightly listed in order of impact.

References: "Code Complete", Steve McConnell, Microsoft Press
People / Management

- #1 - Territoriality, on the part of all participants, at all levels, in all categories.

  examples:
  "marketing was always interfering"
  "the customer kept asking for more"
  "my engineers kept trying new things"
  "everyone wanted it their own way"

preachy advice to students #2

“Disagree, concede, compromise, but build the best system possible.”

People / Management

- Weak Personnel and Problem Employees (see McConnell) - spoiling an otherwise jelled team - includes management above the team.

- Reducing a person’s contribution, by not managing the whole person – treating a person as “not a person” because he/she’s at work

  - 15% of injuries treated at a Bethlehem Steel plant were from kicking and punching time clocks.

  source: Trouble in Bethlehem
People / Management

- Team member burnout.
- Undermined Motivation (McConnell) -
  - Office Gossip
  - Office Politics
- Unrealistic Expectations and Wishful Thinking (McConnell) - planning to catch up later.
- Lack of project sponsorship from above and buy-in from below (McConnell)

preachy advice to students #3

“Why do we choose the jobs that we do?”

People / Management

- Inadequate, uninformed budgets and schedules –
  - There is a minimum, but no maximum, of what a project will take. Admit to that minimum. Set up realistic groundrules, track and control the project from beginning to end.
  - Stupid estimates
  - Knowingly omitting tasks from the budget and schedule
- No contingency plan - starting out with 60 hr. weeks.
People / Management

- Using unpaid overtime in the schedule baseline.
- Insufficient basic management - risk management, planning, reporting, and controls (McConnell) - cross your fingers and wait for surprises.
- Abandonment of planning under pressure (McConnell).

People / Management

- Abandonment of common sense under pressure - testing, quality assurance, personal hygiene.
- Hero based projects –
  - Reliance on one person’s mania and drive.
  - The truck factor.
The **Wrong/Right List**

The survey included many anecdotes –

- Rank / Responsibilities
- My way / Our way
- Arbitrary Deadlines / Reality
- Threatening / Teaching
- Time Clocks / Autonomy
- Employee of the Month / Team-oriented Rewards

The **Wrong/Right List**

- Experts / Generalists
- Temporal bonuses / Bonuses based on performance
- E-mail / Face to face
- No personal calls / Home at work & Work at Home
- One big goal / Incremental goals
- Bureaucracy - don't use that credit card / Find a way
- Copied software / Legal software

The **Wrong/Right List**

- 1-person Jobs (Heroes) / Teams (Leaders)
- Casual Overtime / Compensation
- Sarcasm / Honesty
- Calculated Deception / Honesty
- Management by Intimidation / Delegation of Responsibility
- Annual Performance Reviews – ambush / Monthly Performance Reviews
The **Wrong/Right List**

- Mandatory Overtime / Goal Oriented - interesting work, realistic schedules, or else staff it correctly
- Status Meetings / Issues Meetings
- Hands-off / Hands-On
- Misplaced Recognition / Knowing your team
- 5-O’clock walk-aways / All-day walk arounds
- Management by Cliché / Management of Reality
Process

% of time spent

The Dream Curve

Spec Analysis Design Code Integration Test Maintenance

Process

- Omission of Lifecycle Steps -
  - Lack of a requirements specification.
  - Inadequate (or omission of) Design
  - Lack of a test plan.
  - Coding too quickly. Designing at the terminal.

- Overspecification and too-rigid specification and design.
  - Some details are better decided incrementally.
  - Designing for flexibility is different than designing for function.
**Process**

- Requirements creep, feature creep, developer gold-plating (McConnell).
- Lack of a Software Engineering subculture -
  - Engineering methods
  - Tracking, traceability, and control
  - Education
- Lack of non-execution-based tests.
- Miscommunication of intent - in requirements and design.

---

**Process**

- Lack of Coding Standards.
- Out of date, inaccurate documentation - requirements specifications and test plans.
- No Version Control or any enforced record keeping.
  - During development.
  - During maintenance.

---

**Process**

- No Configuration Management of delivered product.
- Same people design and code. Same people code and test.
- Over-application of technology
  - the Technology Scale (?) asks “Could an animal do this cheaper? Then how about a motor?”
Beige Cosmos Produces Red Faces
By JOHN NOBLE WILFORD

Few gave a thought to the color of the universe until two months ago, when astronomers at Johns Hopkins University ran calculations through a spectrum of color schemes and concluded that on average the universe is pale turquoise, or just a shade greener.

It is a pleasingly serene color, which made the front pages of newspapers and the TV news. But reality, it turns out, is not so vivid. The universe is really beige. Get used to it.

"We got it wrong," the astronomers, Dr. Karl Glazebrook and Dr. Ivan Baldry, announced yesterday. They said they had been led astray by a flaw in their computer software.
preachy advice to students (I’ve lost count)

“design and program for an audience, to agreed standards.”

“standards are guidelines”

A division of Grumman

“There will be no comments in the source code. Code should be self-documenting”

get a sense of perspective
Go-to’s, pointers, and issues of clarity
Go-tos are not to be "avoided at all costs". It is, instead, serpentine code that needs to be avoided. Simplicity and clarity should override most other design decisions. A go-to, in particular, is a powerful tool when used as a direct, no-nonsense jump under well-stated conditions, and can very closely follow problem-space behavior if used with some planning and forethought (Ada, a language designed from scratch by smart French people, contains a goto keyword). On the other hand, the indirection of a pointer tends to be a computer-space construct, that is often confusing and - if honesty should prevail - unnecessary (Java, the latest geek programming language, does not allow the use of pointers).

... Eastman Kodak’s coding standards
Style

- Not matching the "solution space" to the "problem space"
  - Reliance on computer-centric constructs to match real world events.
  - Synchronous modeling of asynchronous processes.

  e.g. Using the pattern name instead of what it’s modeling,... a program that tracked used car parts, had four Model-View-Controllers,

  all named Model_View_Controller.

Style

- Reliance on esoteric technologies and CASE tools. You should suffer through enough projects without them to understand their need and place.
- Extending a system’s design and architecture beyond its intent - know when to start over.
- Keeping prototypes as part of the end system

Style

- Comments that reflect what is done rather than what is intended.
- Residual, inaccurate comments.
- Leaving commenting until the end.

my favorite, actual comments:

```
#define ON 1
#define OFF 1 // set to the same as ON to // avoid confusion
```
**Style**
- Indirect Addressing - pointers, convoluted array indices.
- Nesting greater than 3 levels.
- Tightly coupled modules.
- Incohesive modules.

**Style**
- Freely written global variables.
- Weakly typed variables - operations and comparisons done on variables of incompatible type.
- Code not understood by the author - done by trial and error.
- Coding for speed and size to the exclusion of architectural design.

**Summary**
- People/Management: People will largely perform as expected, unless self-worth is damaged.
- Process: a Methodology is a necessity.
- Style: Technically complex solutions are not required and yield poor results.