Why Projects Fail...

These are the slides for our *Why Projects Fail* talk. We've given it many times, and it always gets a good response. If you'd like us to come and give it for your group or organization, contact us at http://www.stellman-greene.com.

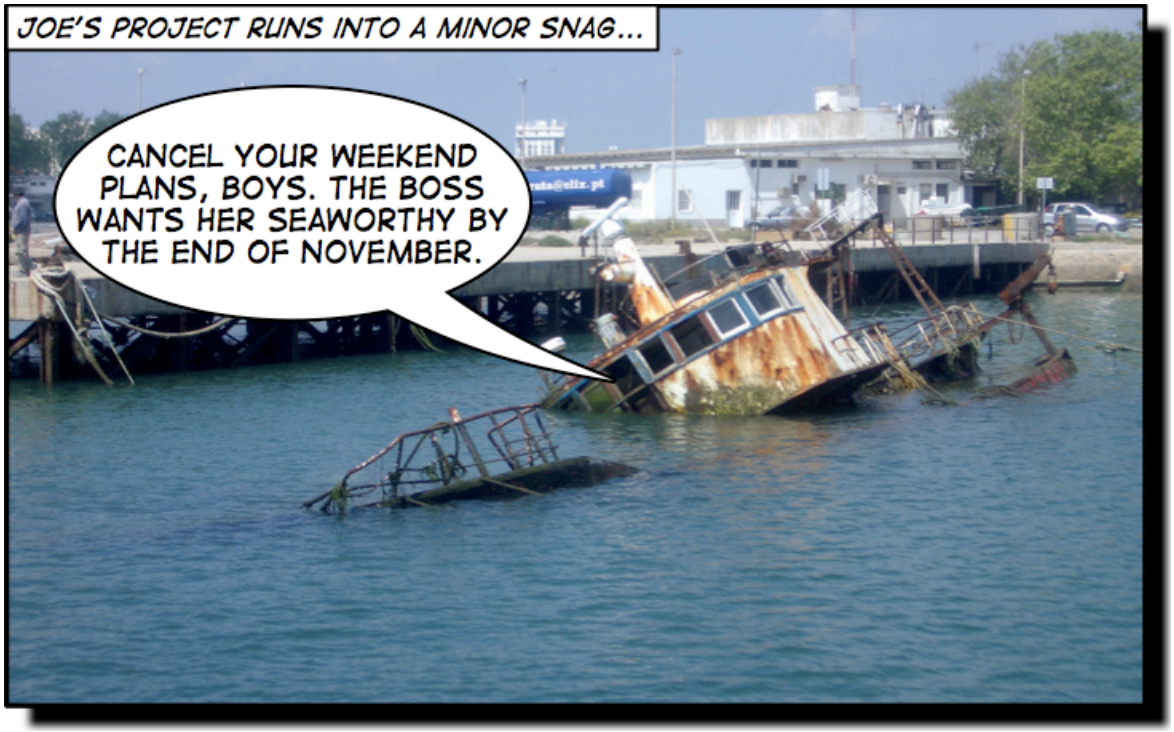


Photo by Dan Taylor (CC license)

... and what you can do about it

Even if it's not your fault!

A presentation by Jennifer Greene and Andrew Stellman

Who we are...

Andrew Stellman started programming in the 80s, and lost count of how many languages he's worked with.

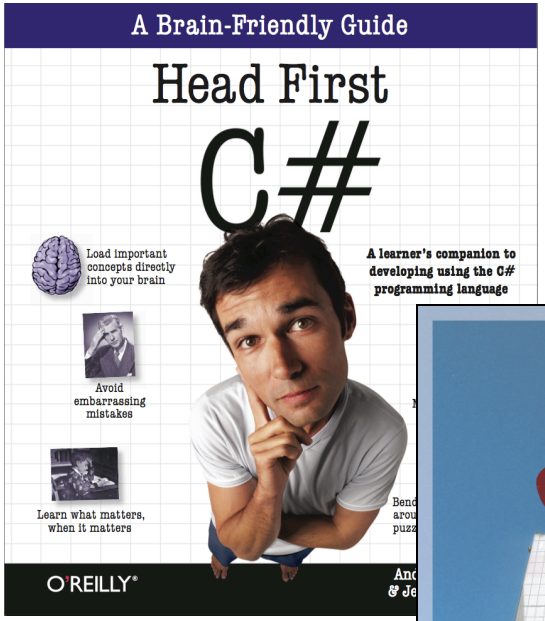
Jennifer Greene's spent the last 20 years or so managing development and testing teams.

He's led teams of programmers, requirements analysts and process engineers.

She's currently doing consulting and training for a group with a large (500 person) IT team.

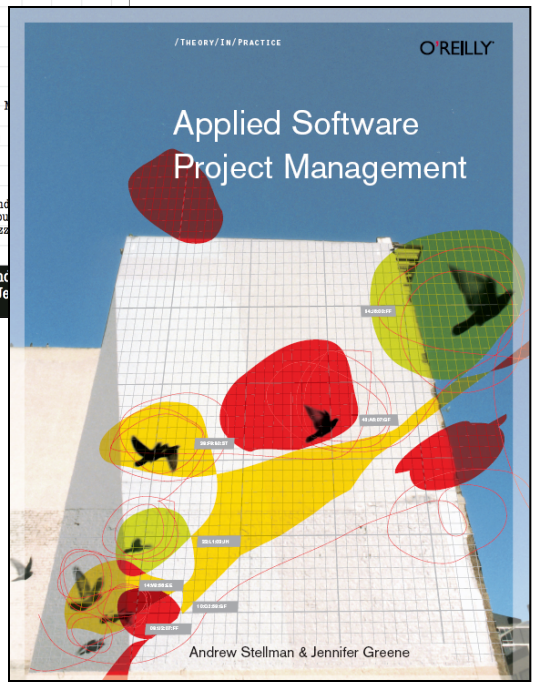
Jenny and Andrew truly believe that with better development practices and good programming habits, we can all build better software.

Our books...

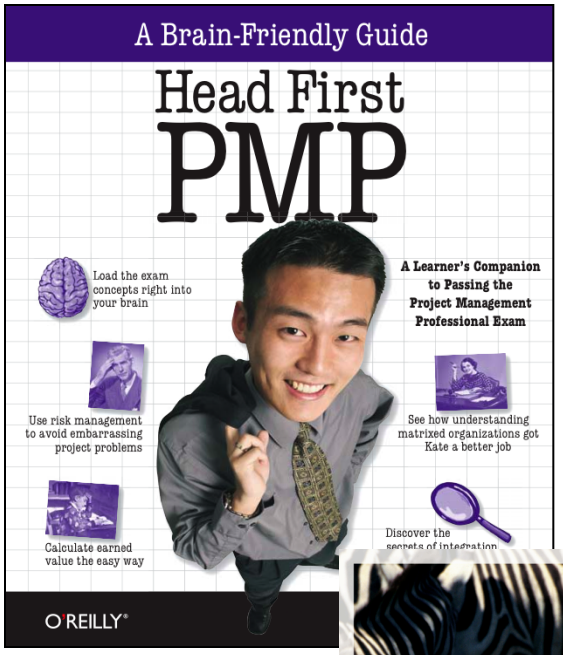


2008 (1st ed)

2010 (2nd ed)



2005



2007 (1st ed)

2009 (2nd ed)

2010

WARNING: This is NOT an academic presentation

The topics we are about to cover may be deadly serious, but we won't be

If you want academic slides, we've got 'em:
http://www.stellman-greene.com/slides

Not all failures are this easy to spot...

...but some projects do fail spectacularly.

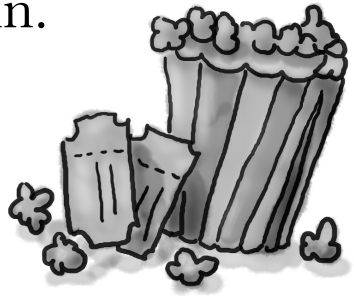
The Tacoma Narrows Bridge project failed before the first yard of concrete was poured.

There was nothing wrong with the construction. Poor design and badly planned cost cutting in materials led to an unfortunate end.

If this video doesn't play, try using a newer version of Acrobat – or you can download the video here:
http://www.stellman-greene.com/GallopingGertie

"This time it's different..."

There's an old saying about how there are a million ways to fail, but only one way to be right. When it comes to projects, nothing's further from the truth. Projects fail the same few ways over and over again.



Don't go in the basement!

Software projects are **a lot like cheesy horror movies**. After you've seen a few of them, you know that the first guy to leave the group is going to get an axe in his head. Projects are the same way. People keep making the same mistakes over and over, and it keeps getting their projects killed.

You know you're on a failed project when...

A judge in 1964 said, "I don't know how to define pornography, but I know it when I see it." And the same goes for failing projects - we all know when we're on one that's sinking.

What does a failing project look like?

You know your project failed if it got aborted and everyone was laid off. But there are other, less obvious kinds of failure:

- The project costs a lot more than it should.
- It takes a lot longer than anyone expected.
- The product doesn't do what it was supposed to.
- Nobody is happy about it.

Sometimes failure seems normal

Nobody sets out to fail, but for some reason people just accept that a lot of software projects won't deliver on time, under budget with the expected scope intact. But talking about what causes failure makes people **uncomfortable**, because nobody wants to give or take that kind of criticism.

A show of hands, please...

Jenny and I have never met a single professional developer, tester or project manager with more than a few years of experience working on software projects who hasn't been on at least one failed project.

Are there any here?

Four basic ways projects can fail

There are plenty of ways that you can categorize failed projects. I like to think of them like this:

- **Things the boss does**
  Classic management mistakes that can damage the project
- **Things the team should've done**
  Once in a while, it really *is* the team's fault
- **Things the software does (or doesn't do)**
  How your project doesn't quite meet the needs of the people you built it for
- **Things that could have been caught**
  …but weren't until it was way too late

Things the boss does

Some problems start at the top… and when we're managing teams (or when we're **leading our projects**), that means a lot of problems that affect other people start with us.

- Unmanaged changes
- Micromanagement
- Over-reliance on gut instincts
- Tunnel vision
- An artificial "wall" that the business puts up to disconnect from the engineering team

Like it or not, when you're leading a project team you're the boss!

Up close: A Vision & Scope Document keeps everyone on the same page

The *Vision and Scope document* is where you define **who** needs the product, **what** they need it for, and **how** it will fulfill those needs.

Vision and scope document outline

1. Problem Statement
   a. Project background
   b. Stakeholders
   c. Users
   d. Risks
   e. Assumptions
2. Vision of the Solution
   a. Vision statement
   b. List of features
   c. Scope of phased release (optional)
   d. Features that will not be developed

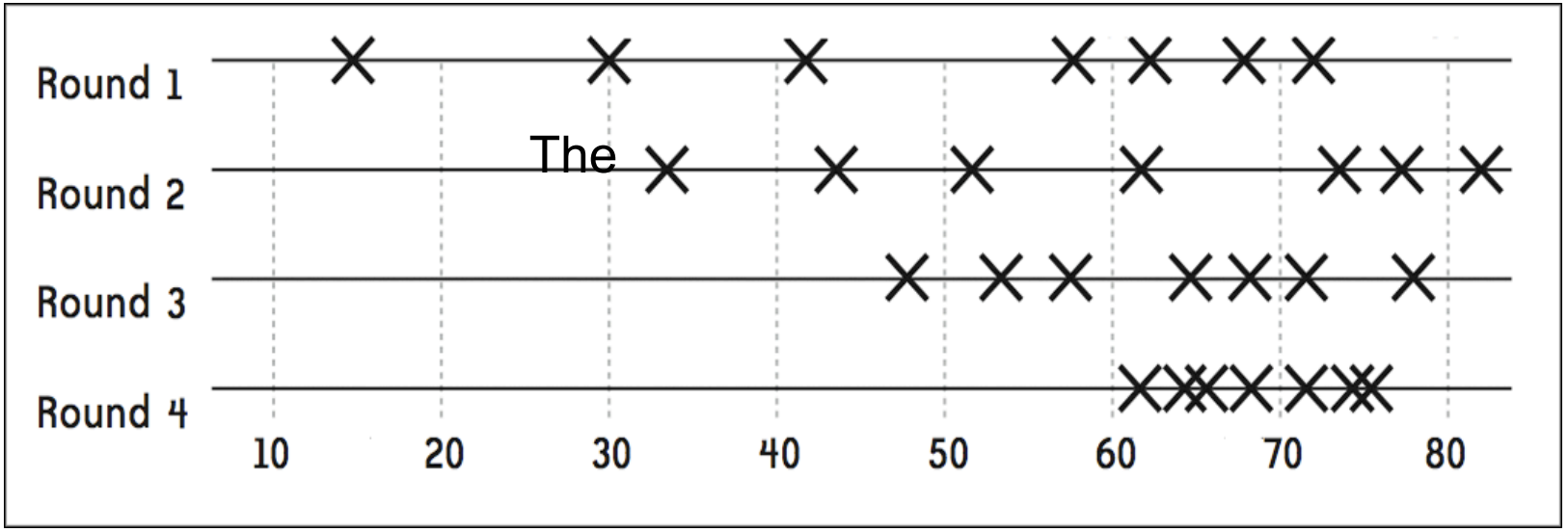
Things the team should've done

The team could have done the work more efficiently, if only we'd taken the time to think it through.

- "It'll take about three weeks…"
- Padded estimates compensate for unknowns.
- Project teams will just pick a deadline and stick to it, no matter what basic reason and common sense tell them.
- Somehow non-programming tasks always seem to get cut when the deadline gets closer.
- Misunderstood predecessors lead to cascading delays.

Up close: Wideband Delphi keeps estimates honest

Wideband Delphi is a repeatable estimation process that guides your experts and team members so their estimates converge accurately.



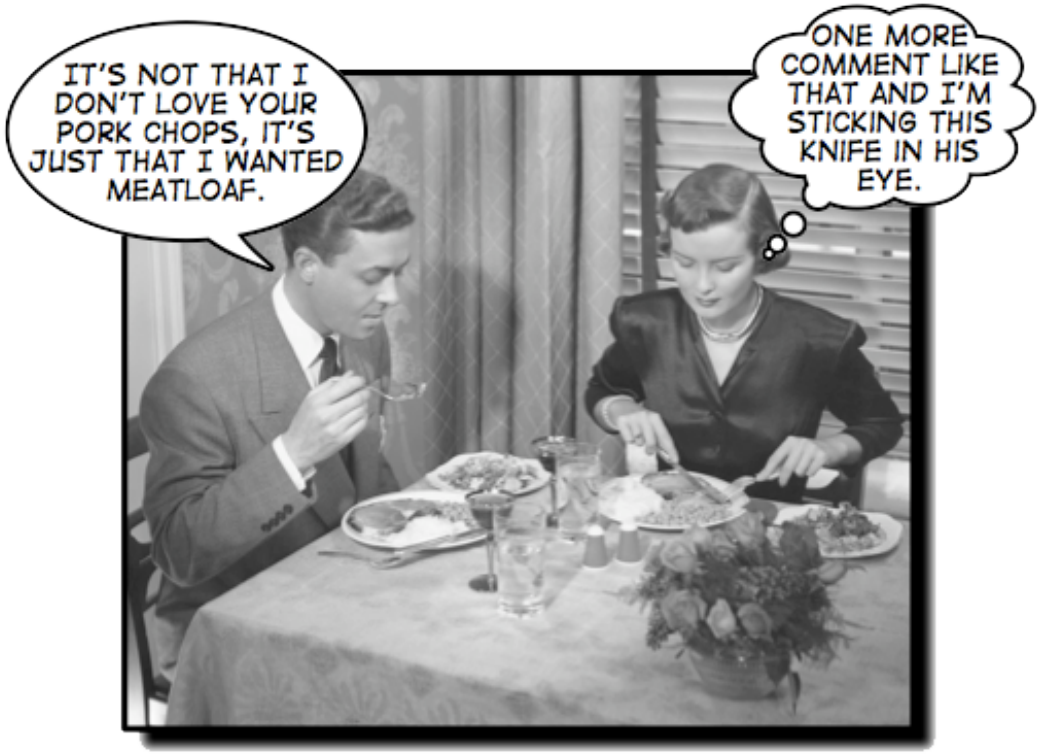
We cover Wideband Delphi in detail in the Estimation chapter of Applied Software Project Management - download the PDF here:

http://www.stellman-greene.com/chapter3

Things the software does (or doesn't do)

It seems pretty obvious that you should know what the software's supposed to do before you start building it... not that that stops us.

- We only find serious problems after we've built them into the software
- We have big, useless meetings that fail to figure out what the software's supposed to do
- Scope creep
- 90% done, with 90% left to go.



Up close: Use cases can help you avoid requirements problems

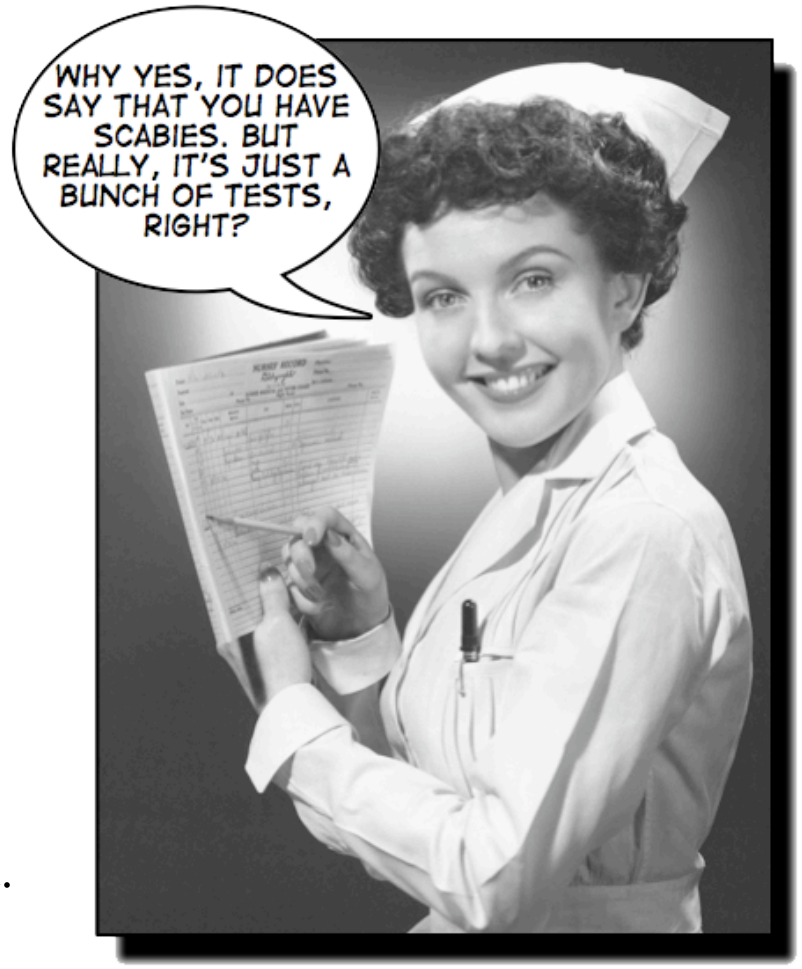
Use cases are a deceptively simple way to document every planned interaction between the users (and other actors) and the software.

Name	**UC-8: Search**
Summary	All occurrences of a search term are replaced with replacement text.
Rationale	While editing a document, many users find that there is text somewhere in the file being edited that needs to be replaced, but searching for it manually by looking through the entire document is time-consuming and ineffective. The search-and-replace function allows the user to find it automatically and replace it with specified text. Sometimes this term is repeated in many places and needs to be replaced. At other times, only the first occurrence should be replaced. The user may also wish to simply find the location of that text without replacing it.
Users	All users
Preconditions	A document is loaded and being edited.
Basic course of events	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Postconditions	All occurrences of the search term have been replaced with the replacement text.

Learn more about use cases here: http://www.stellman-greene.com/usecase

Things that could have been caught

Which would you choose: a well-built program that doesn't do what you need or a crappy one that's irritating to use and does?



- Getting a few tech support people to "bang on the software" is ***not*** testing.
- Maybe we could've caught that design problem *before* the code was built.
- Maybe we could've caught that code problem *before* we went to test.
- "Beta" does not mean "use at your own risk."

Up close: Don't overlook your acceptance criteria!

It's short-circuited far too often in favor of user acceptance testing, but, acceptance testing is about **more than just user acceptance**.

1. Successful completion of all tasks as documented in the test schedule.
2. Quantity of medium- and low-level defects must be at an acceptable level as determined by the software testing project team lead.
3. User interfaces for all features are functionally complete.
4. Installation documentation and scripts are complete and tested.
5. Development code reviews are complete and all issues addressed. All high-priority issues have been resolved.
6. All outstanding issues pertinent to this release are resolved and closed.
7. All current code must be under source control, must build cleanly, the build process must be automated, and the software components must be labeled with correct version numbers in the version control system.
8. All high-priority defects are corrected and fully tested prior to release.
9. All defects that have not been fixed before release have been reviewed by project stakeholders to confirm that they are acceptable.
10. The end user experience is at an agreed acceptable level.
11. Operational procedures have been written for installation, set up, error recovery, and escalation.
12. There must be no adverse effects on already deployed systems.

What you can do about it

Some easy ways to make sure your project **doesn't fail**:

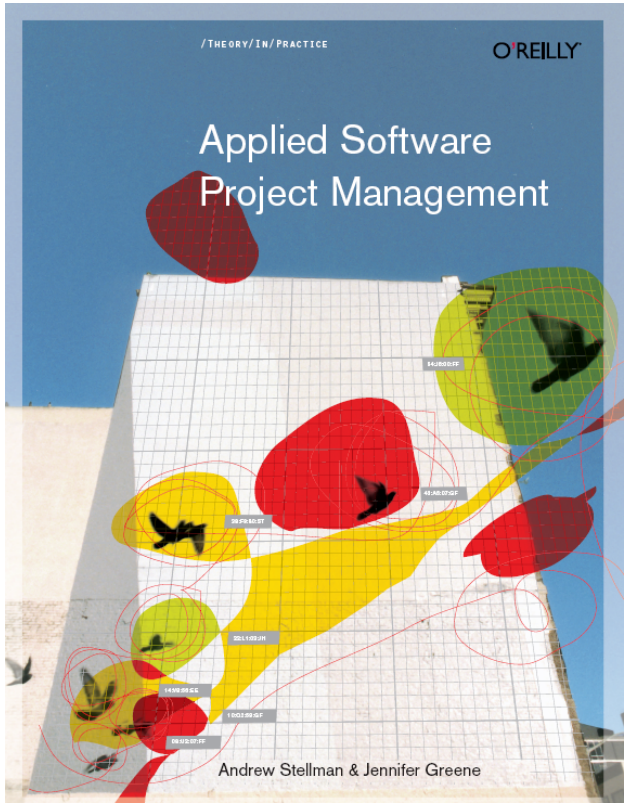
- Tell the truth all the time
- Trust your team
- Review everything, test everything
- Check your ego at the door
- The fastest way through the project is the right way through the project

Repeat after me: "Practices, practices, practices."

The solutions I talked about are only a few small steps towards a **better software process**.

- Process improvement starts with setting **concrete goals** and making incremental improvements.
- They're good solutions to **specific** problems, but they might not solve ***your*** problems.
- There are lots more solutions where those came from. And the ones we chose were the ones that we could explain quickly.
- Make sure the solutions you choose address the problems that **hurt** the most.

One last quick note from the marketing department...



Buy these books!

And check out our blog, "Building Better Software"

http://www.stellman-greene.com/

Thanks for coming! Any questions?