Why Projects Fail

and what you can do about it

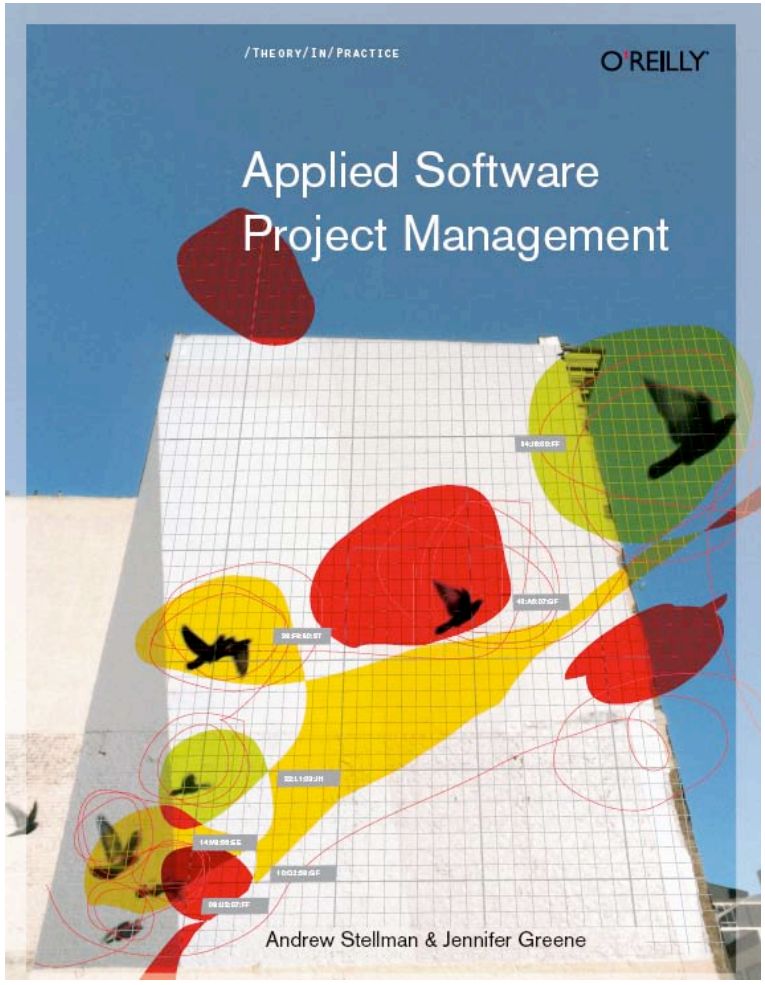
Photo by Dan Taylor (CC license)

A presentation by Jenny and Andrew

for NYC CitySPIN / PMINYC

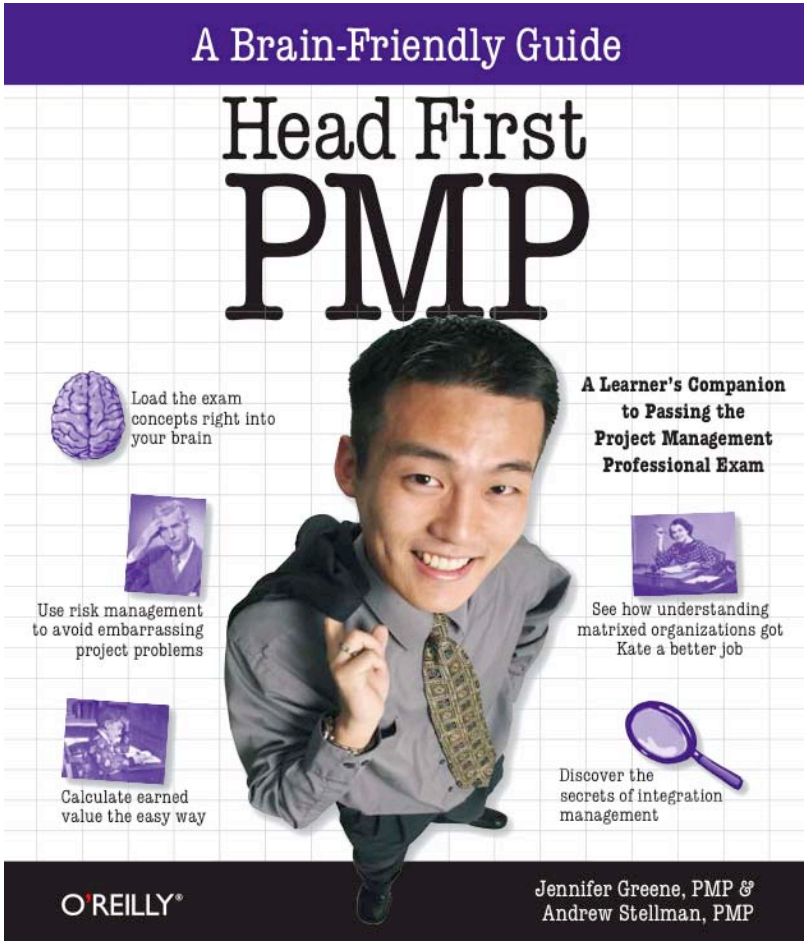
October 9, 2007

Applied Software Project Management



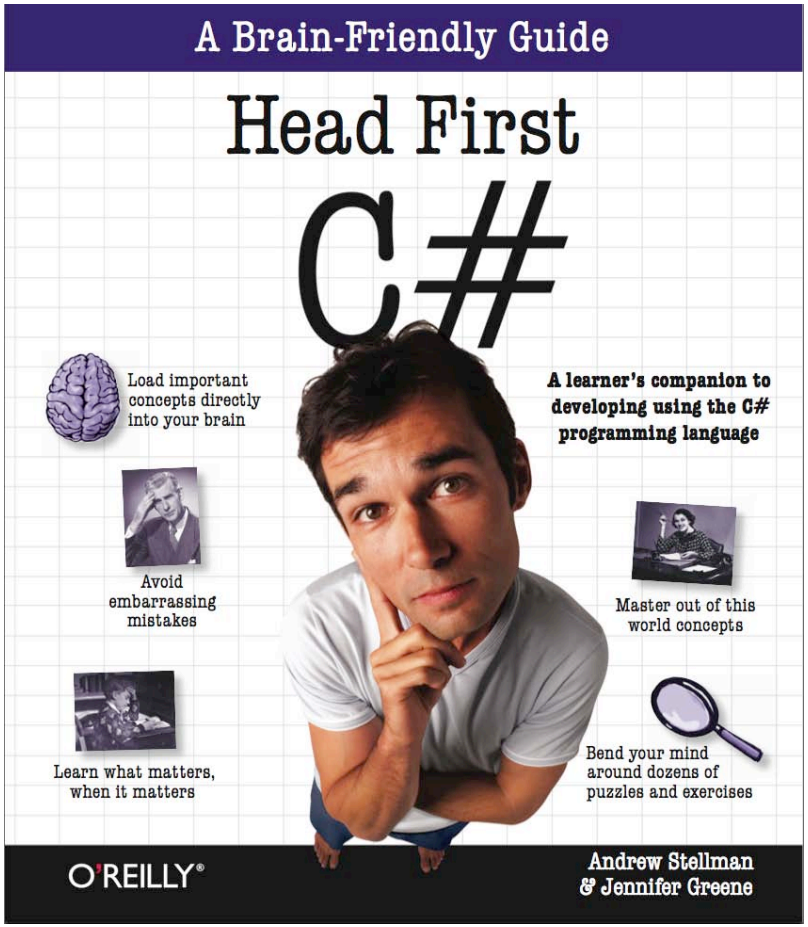
- ★Textbook on software project management
- ★Used as a textbook in university graduate software engineering courses
- ★Focuses on practices aimed at solving specific project problems
- ★(like the ones we'll talk about tonight)
- ★Published in 2005 (322 pages)

Head First PMP



- ★ Second-best selling PMP preparation guide, after only six months on the market
- ★ Thousands of copies sold in the last quarter alone
- ★ Rave reviews from members of the PMBOK® Guide leadership team ("by far the best PMP exam preparation book I have reviewed" - Dennis Bolles, PMP)
- ★ Uses a unique style with humor, graphics and an emphasis on cognitive learning principles to help people understand the concepts without rote memorization
- ★ Published in 2007 (692 pages)

Coming Soon - Head First C#



- ★A brain-friendly guide to learning the C# programming language
- ★Aimed at novices and intermediate programmers unfamiliar with C#
- ★Manuscript will be complete and tech reviewed within the next three weeks
- ★Which is why we've been so slow to answer e-mail lately
- ★To be published late 2007 (650 pages)

Who we are...

Andrew and Jenny each have over 15 years of experience in software engineering, and have been working together since 1998

★ Andrew is an independent consultant hired by software engineering contracting companies to manage large-scale, globally distributed software development teams

★ He is a graduate of the School of Computer Science at Carnegie Mellon University and is a PMP-certified project manager

★ Jennifer has managed large software teams distributed over multiple continents for billion-dollar companies

★ She is a PMP-certified project manager with a strong background in quality management and software testing

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

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.

"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...

We've never met a single professional software engineer with more than a few years of experience 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. We like to think of them like this:

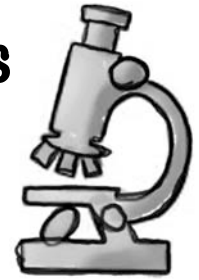
- ★**Things the software does (or doesn't do)**
  How your project doesn't quite meet the needs of the people you built it for
- ★**Things the team should've done**
  Once in a while, it really *is* the team's fault
- ★**Things that could have been caught**
  …but weren't until it was way too late.
-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

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

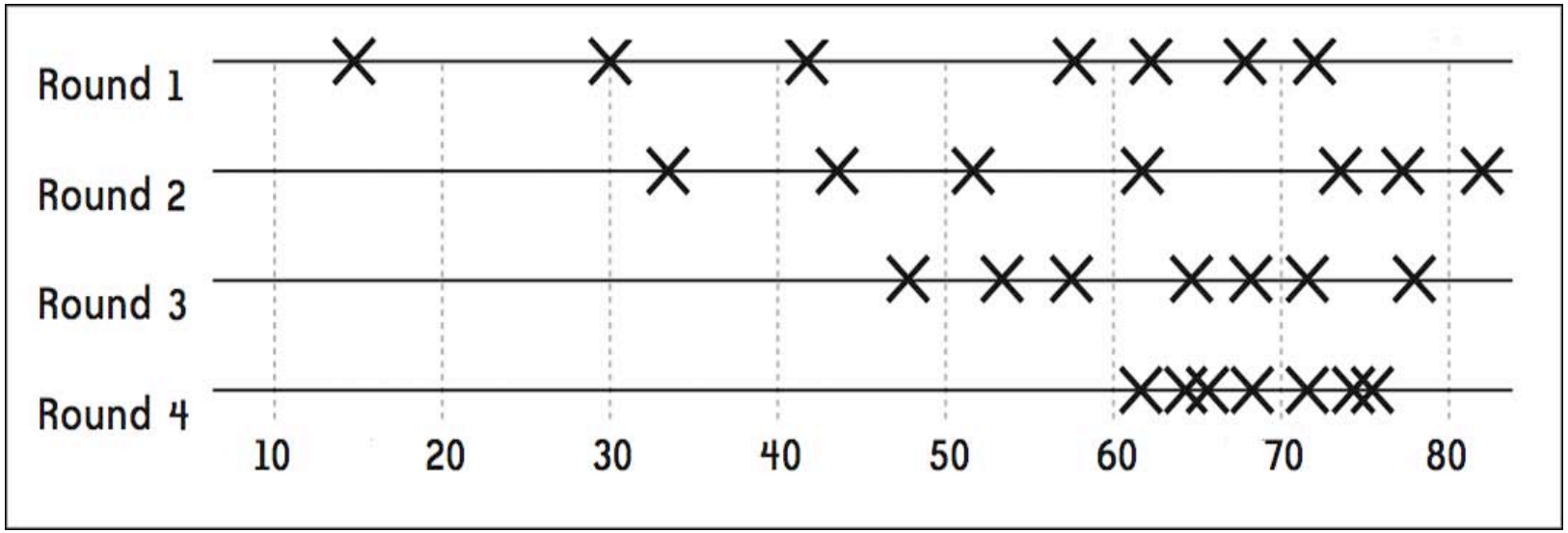
Things the team should've done

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

- ★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 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.

Things the boss does

Some problems start with senior managers, others start with ***us PMs***. But they can all sink the project.

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

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

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 us: "Practices, practices, practices."

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

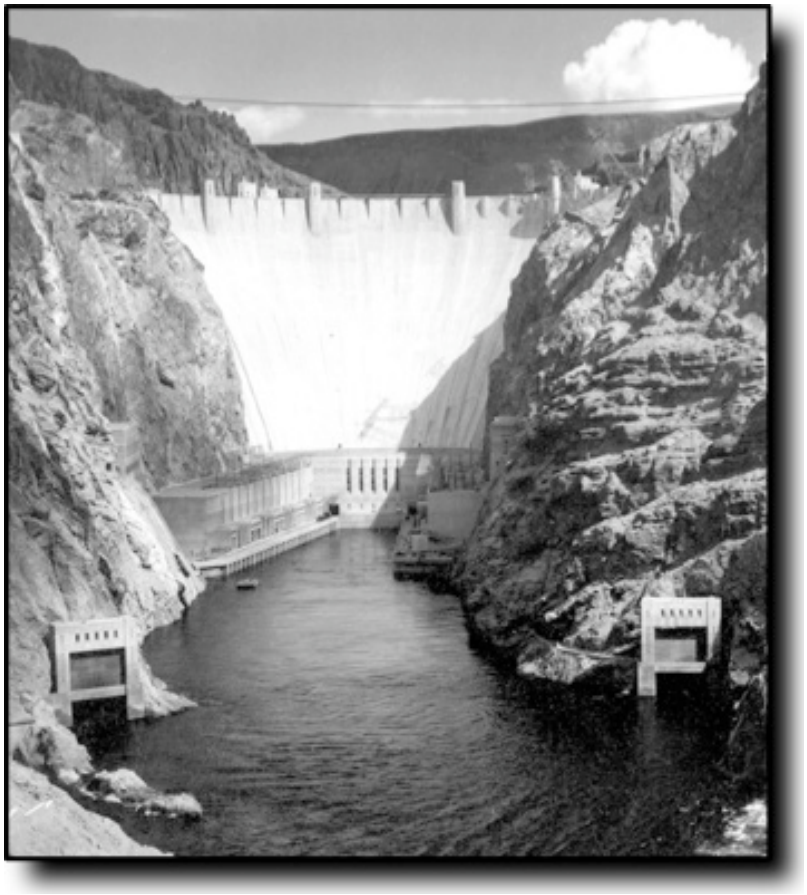
- ★ Process improvement starts with setting **concrete goals** and making incremental improvements.
- ★ They're good solutions to **specific** problems, but they might not solve ***your*** problems.
- ★ There are lots more solutions where those came from. And the ones we chose were the ones that we could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

The talent is there... the project management's not.

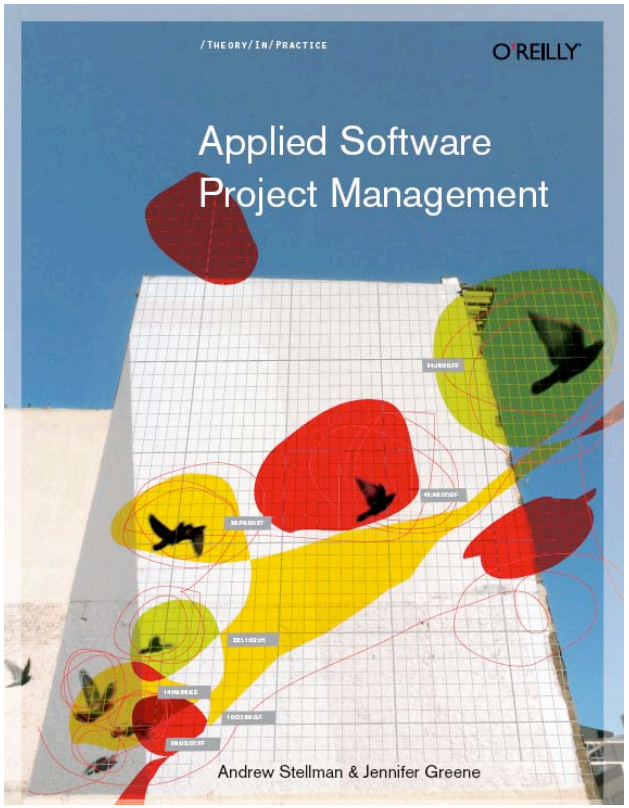
Hoover Dam was finished two years early, and under budget. Software's not so different that we can't engineer it just as well.

- ★Our problems have, for the most part, been solved.
- ★Over and over and over again. Seriously.
- ★**We just have to stop ignoring the solutions.**

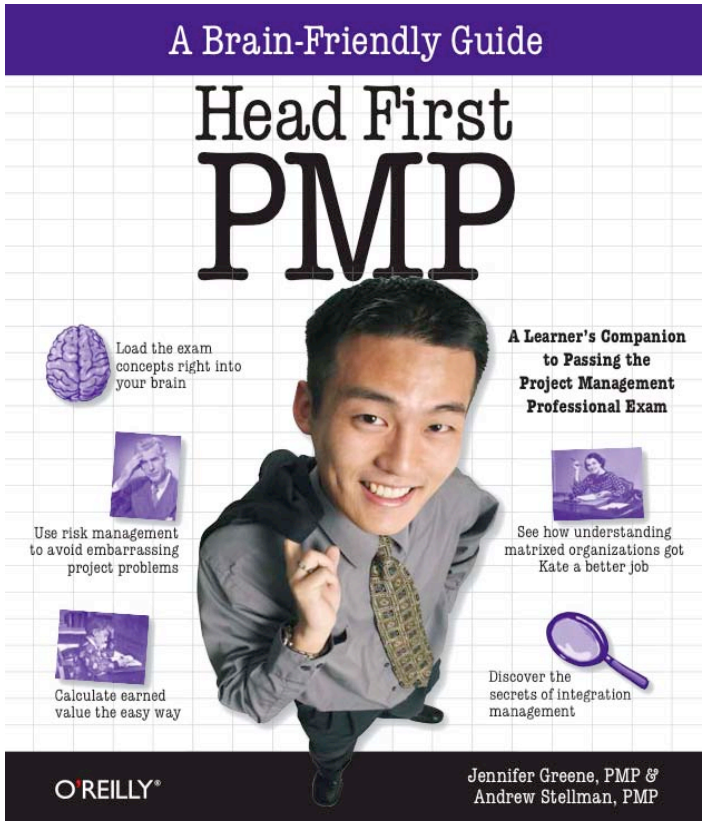
Do you think your project will take more effort than this one?



One last quick note from the O'Reilly marketing department



Buy these books



And check out our blog, "Building Better Software"

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

We'll post these slides in the next few days.