

The Nine Deadly Sins of Project Planning

Steve McConnell

At a time when some software organizations have achieved close to perfect on-time delivery records,¹ others continue to suffer mediocre results. Surveys generally indicate that poor project planning is one of the top sources of problems.

How can you recognize a badly planned software project? Here are nine of the deadliest sins I've found in project planning.



1. Not planning at all

By far the most common planning problem is simply not planning at all, and this sin is easily avoided. A person need not be an expert planner to plan effectively. I've seen numerous instances of projects planned by rank amateurs that have run well simply because the people in charge

had carefully considered their project's specific needs. Forced to choose between an expert project planner who doesn't carefully think through his plan or an amateur who has thoroughly evaluated his project needs, I'll bet on the amateur every time.

2. Failing to account for all project activities

If Deadly Sin #1 is not planning at all, Deadly Sin #2 is not planning *enough*. Some project plans are created using the assumption that no one on the software team will get sick, attend training, go on vacation, or quit. Core activities are often underestimated to a great degree. Plans created using unrealistic assumptions like these set up a project for disaster.

There are numerous variations on this theme. Some projects neglect to account for ancillary activities such as the effort needed to

create setup programs, convert data from previous versions, perform cutover to new systems, perform compatibility testing, and other pesky kinds of work that take up more time than we would like to admit.

Some late projects propose to catch up by reducing their originally planned testing cycle; they reason that there probably won't be very many defects to detect or correct. (I leave as an exercise for the reader to determine why—if this is really the case—they didn't plan for a shorter testing cycle in the first place.)

3. Failure to plan for risk

In *Design Paradigms*,² Henry Petroski argues that the most spectacular failures in bridge design were generally preceded by periods of success that led to complacency in the creation of new designs. Designers of failed bridges were lulled into copying the attributes of successful bridges and didn't pay enough attention to each new bridge's potential failure modes.

For software projects, actively avoiding failure is as important as emulating success. In many business contexts, the word "risk" isn't mentioned unless a project is already in deep trouble. In software, a project planner who isn't using the word "risk" every day and incorporating risk management into his plans probably isn't doing his job. As Tom Gilb says, "If you do not actively attack the risks on your project, they will actively attack you."³

4. Using the same plan for every project

Some organizations grow familiar with a particular approach to running software projects, which is known as "the way we do things around here." When an organization uses this approach, it tends to do well as long as the new projects look like the old projects. When new

DEPARTMENT EDITORS

Bookshelf: Warren Keuffel,
wkeuffel@computer.org

Country Report: Deependra Moitra, Lucent Technologies
d.moitra@computer.org

Design: Martin Fowler, ThoughtWorks,
fowler@acm.org

Loyal Opposition: Robert Glass, Computing Trends,
rglass@indiana.edu

Manager: Don Reifer, Reifer Consultants,
dreifer@sprintmail.com

Quality Time: Jeffrey Voas, Cigital,
voas@cigital.com

STAFF

Senior Lead Editor
Dale C. Strok
dstrok@computer.org

Group Managing Editor
Crystal Chweh

Associate Editors
**Jenny Ferrero and
Dennis Taylor**

Staff Editors
**Shani Murray, Scott L. Andresen,
and Kathy Clark-Fisher**

Magazine Assistants
Dawn Craig
software@computer.org

Pauline Hosillos

Art Director
Toni Van Buskirk

Cover Illustration
Dirk Hagner

Technical Illustrator
Alex Torres

Production Artists
Carmen Flores-Garvey and Larry Bauer

Acting Executive Director
Anne Marie Kelly

Publisher
Angela Burgess

Assistant Publisher
Dick Price

Membership/Circulation
Marketing Manager
Georgann Carter

Advertising Assistant
Debbie Sims

CONTRIBUTING EDITORS

Greg Goth, Keri Schreiner, and Gil Shif

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Send 2 electronic versions (1 word-processed and 1 postscript or PDF) of articles to Magazine Assistant, *IEEE Software*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; software@computer.org. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

projects look different, however, reusing old plans can cause more harm than good.

Good plans address specific conditions of the project for which they are created. Many elements can be reused, but project planners should think carefully about the extent to which each element of a previous plan still applies to the new project context.

5. Applying prepackaged plans indiscriminately

A close cousin to Deadly Sin #3 is reusing a generic plan someone else created without applying your own critical thinking or considering your project's unique needs. "Someone else's plan" usually arrives in the form of a book or methodology that a project planner applies out of the box. Current examples include the *Rational Unified Process*,⁴ *Extreme Programming*,⁵ and to some extent (despite my best intentions to the contrary) my own *Software Project Survival Guide*⁶ and my company's Cx-One. These prepackaged plans can help avoid Deadly Sins #1 and #2, but they are not a substitute for thinking about and optimizing your plans to the unique demands of your project.

No outside expert can possibly understand a project's specific needs as well as the people directly involved. Project planners should always tailor the "expert's" plan to their specific circumstances. Fortunately, I've found that project planners who are aware enough of planning issues to read software engineering books usually also have enough common sense to be selective about the parts of the prepackaged plans that are likely to work for them.

6. Allowing a plan to diverge from project reality

One common approach to planning is to create a plan early in the project, then put it on the shelf and let it gather dust for the remainder of the project. As project conditions change, the plan becomes increasingly irrelevant, so by mid-project the project runs free-form, with no real relationship between the unchanging plan and project reality.

This Deadly Sin is exacerbated by Deadly Sin #5—project planners who

embrace prepackaged methodologies whole-hog are sometimes reluctant to change them midstream when they're not working. They think the problem is with their application of the plan when, in fact, the problem is with the plan. Good project planning should occur and recur incrementally throughout a project.

7. Planning in too much detail too soon

Some well-intentioned project planners try to map out a whole project's worth of activities early on. But a software project consists of a constantly unfolding set of decisions, and each project phase creates dependencies for future decisions. Since planners do not have crystal balls, attempting to plan distant activities in too much detail is an exercise in bureaucracy that is almost as bad as not planning at all.

The more work that goes into creating prematurely detailed plans, the higher the likelihood the plan will become shelfware (Deadly Sin #6). No one likes to throw away previous work, and project planners sometimes try to force-fit the project's reality into their earlier plans rather than laboriously revising their prematurely detailed plans.

I think of good project planning like driving at night with my car's headlights on. I might have a road map that tells me how to get from City A to City B, but the distance I can see in detail in my headlights is limited. On a medium-size or large project, macro-level project plans should be mapped out end-to-end early in the project. Detailed, micro-level planning should generally be conducted only a few weeks at a time and "just in time."

8. Planning to catch up later

For projects that get behind schedule, one common mistake is planning to make up lost time later. The typical rationalization is that, "The team was climbing a learning curve early in the project. We learned a lot of lessons the hard way. But now we understand what we're doing and should be able to finish the project quickly." Wrong answer! A 1991 survey of more than 300

projects found that projects hardly ever make up lost time—they tend to get further behind.⁷ The flaw in the rationalization is that software teams make their highest-leverage decisions earliest in the project—the time during which new technology, new business areas, and new methodologies are the least well understood. As the team works its way into the later phases of the project, it won't speed up; it will slow down as it encounters the consequences of mistakes it made earlier and invests time correcting those mistakes.

9. Not learning from past planning sins

The deadliest sin of all might be not learning from earlier deadly sins. Software projects can take a long time, and people's memories can be clouded by ego and the passage of time. By the end of a long project, it can be difficult to remember all the early decisions that affected the project's conclusion.

One easy way to counter these tendencies and prevent future deadly sins is to conduct a structured project post-

mortem review.⁸ A postmortem review might not erase the sins of projects past, but it can certainly help prevent sins on future projects. ☞

References

1. S. Ahuja, "Laying the Groundwork for Success," *IEEE Software*, vol. 16, no. 6, Nov.–Dec. 1999, pp. 72–75.
2. H. Petroski, *Design Paradigms*, Cambridge Univ. Press, Cambridge, U.K., 1994.
3. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Reading, Mass., 1988.
4. P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed., Addison-Wesley, Reading, Mass., 2000.
5. K. Beck, *Extreme Programming: Embrace Change*, Addison-Wesley, Reading, Mass., 2000.
6. S. McConnell, *Software Project Survival Guide*, Microsoft Press, Redmond, Wash., 1997.
7. M. van Genuchten, "Why Is Software Late? An Empirical Study of Reasons for Delay in Software Development," *IEEE Trans. Software Eng.*, vol. 17, no. 6, June 1991, pp. 582–590.
8. B. Collier, T. Demarco, and P. Fearey, "A Defined Process for Project Postmortem Review," *IEEE Software*, vol. 13, no. 4, July–Aug. 1996, pp. 65–72.

Upcoming Topics

November/December '01:
Extreme Programming from a CMM Perspective

January/February '02:
Building Systems Securely from the Ground Up

March/April '02:
Building Internet Software

May/June '02:
Knowledge Management in Software Engineering

EDITOR IN CHIEF:
Steve McConnell
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314
software@construx.com

EDITOR IN CHIEF EMERITUS:
Alan M. Davis, Omni-Vista

ASSOCIATE EDITORS IN CHIEF

Design: Maarten Boasson, Quaerendo Invenietis
boasson@quaerendo.com

Construction: Terry Bollinger, Mitre Corp.
terry@mitre.org

Requirements: Christof Ebert, Alcatel Telecom
christof.ebert@alcatel.be

Management: Ann Miller, University of Missouri, Rolla
miller@ece.umar.edu

Quality: Jeffrey Voas, Cigital
voas@cigital.com

Experience Reports: Wolfgang Strigel,
Software Productivity Center; strigel@spc.ca

EDITORIAL BOARD

Don Bagert, Texas Tech University
Richard Fairley, Oregon Graduate Institute
Martin Fowler, ThoughtWorks
Robert Glass, Computing Trends
Natalia Juristo, Universidad Politécnica de Madrid
Warren Keuffel, independent consultant
Brian Lawrence, Coyote Valley Software
Karen Mackey, Cisco Systems
Deependra Moitra, Lucent Technologies, India
Don Reifer, Reifer Consultants
Suzanne Robertson, Atlantic Systems Guild
Wolfgang Strigel, Software Productivity Center
Karl Wieggers, Process Impact

INDUSTRY ADVISORY BOARD

Robert Cochran, Catalyst Software (chair)
Annie Kuntzmann-Combelles, Q-Labs
Enrique Draier, PSINet
Eric Horvitz, Microsoft Research
David Hsiao, Cisco Systems
Takaya Ishida, Mitsubishi Electric Corp.
Dehua Ju, ASTI Shanghai
Donna Kasperson, Science Applications International
Pavle Knaflic, Hermes SoftLab
Günter Koch, Austrian Research Centers
Wojtek Kozaczynski, Rational Software Corp.
Tomoo Matsubara, Matsubara Consulting
Masao Matsumoto, Univ. of Tsukuba
Dorothy McKinney, Lockheed Martin Space Systems
Nancy Mead, Software Engineering Institute
Stephen Mellor, Project Technology
Susan Mickel, AgileTV
Dave Moore, Vulcan Northwest
Melissa Murphy, Sandia National Laboratories
Kiyoh Nakamura, Fujitsu
Grant Rule, Software Measurement Services
Girish Seshagiri, Advanced Information Services
Chandra Shekaran, Microsoft
Martyn Thomas, Praxis
Rob Thomsett, The Thomsett Company
John Vu, The Boeing Company
Simon Wright, Integrated Chipware
Tsuneo Yamaura, Hitachi Software Engineering

MAGAZINE OPERATIONS COMMITTEE

Sorel Reisman (chair), James H. Aylor, Jean Bacon,
Thomas J. Bergin, Wushow Chou, William I.
Grosky, Steve McConnell, Ken Sakamura, Nigel
Shadbolt, Munindar P. Singh, Francis Sullivan,
James J. Thomas, Yervant Zorian

PUBLICATIONS BOARD

Rangachar Kasturi (chair), Angela Burgess (pub-
lisher), Jake Aggarwal, Laxmi Bhuyan, Mark Chris-
tensen, Lori Clarke, Mike T. Liu, Sorel Reisman,
Gabiella Sannitti di Baja, Sallie Sheppard, Mike
Williams, Zhiwei Xu