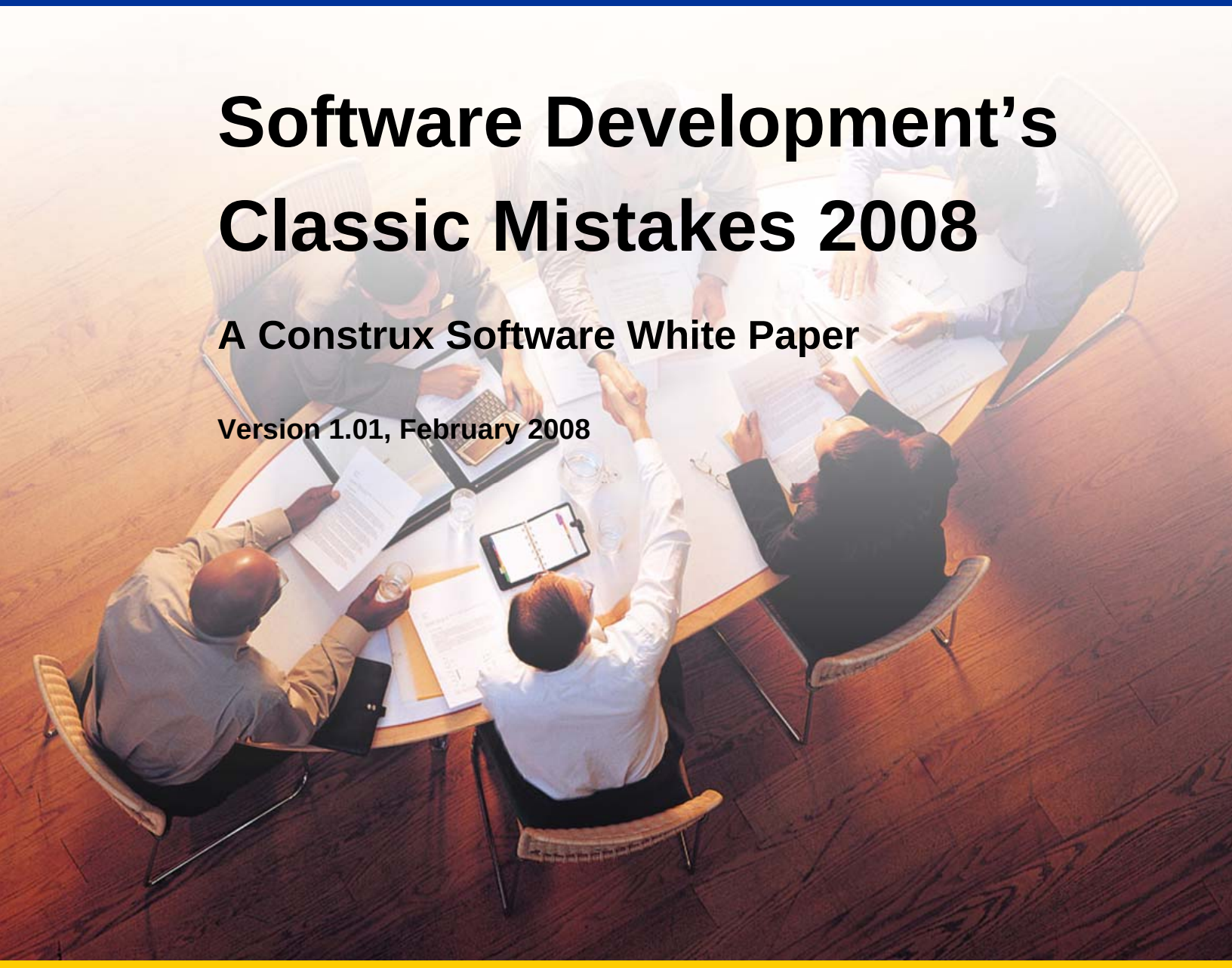
Software Development's Classic Mistakes 2008



A Construx Software White Paper

Version 1.01, February 2008

Construx®

SOFTWARE DEVELOPMENT BEST PRACTICES

For more information about Construx's products and services, please see our website at www.construx.com.

Construx®

SOFTWARE DEVELOPMENT BEST PRACTICES

Contents

List of Tables

Introduction

Construx's Chief Software Engineer/CEO, Steve McConnell, introduced the concept of software development's classic mistakes in his book *Rapid Development*. He defined "classic mistakes" as mistakes that have been made so often, by so many people, that the consequences of making these mistakes should be predictable and the mistakes themselves should be avoidable. His original list contained 36 classic mistakes.

In our work with clients since Construx was founded in 1996, we have found the concept of classic mistakes to be useful. Simply having a list of mistakes is useful. Being able to identify certain mistakes as so common that they are classic, and therefore should be avoidable, is also useful.

To make the list of classic mistakes even more useful, in 2007 Construx consultants updated the list of Classic Mistakes based on Construx's work with hundreds of clients since 1996. The following mistakes were added:

- Confusing estimates with targets
- Excessive multi-tasking
- Assuming global development has a negligible impact on total effort
- Unclear project vision
- Trusting the map more than the terrain
- Outsourcing to reduce cost
- Letting a team go dark (replaces the previous "lack of management controls")

These additions and changes produced a total of 42 classic mistakes. For explanations of these mistakes, see "Appendix G. Classic Mistake Descriptions."

Construx then conducted a survey to determine how frequent and how serious these classic mistakes are.

Summary of Survey Methodology

The survey was conducted from June-July 2007. More than 500 people responded to the survey. As Figure 1 illustrates, the roles of people who responded to the survey mirrors the roles of people in the software industry overall.

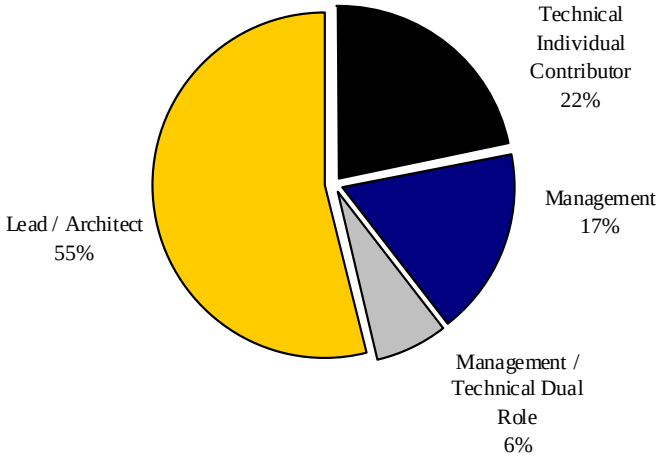


Figure 1. Respondents to Classic Mistakes Survey

Numerous types of software were represented in the survey. Figure 2 shows survey respondents by type of software developed.

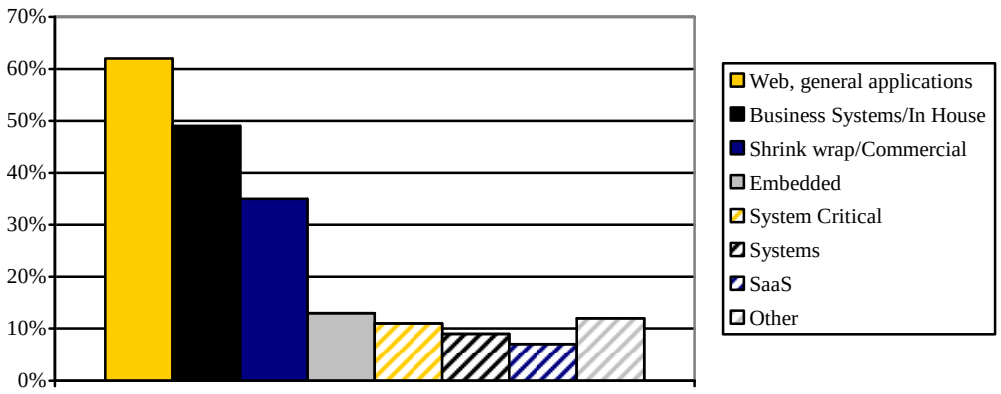


Figure 2. Type of Software Developed

The most common kind of software developed by survey respondents was web software, followed by business systems/in house software and shrink wrap/commercial software. Respondents were allowed to choose more than one kind of software, and there were many respondents who indicated that they are working on both web software and business systems as well as web software and shrink wrap.

For more on the survey methodology, see Appendix A, “Survey Methodology.”

Most Frequent Classic Mistakes

For each mistake, respondents were asked to judge, based on their personal experience, how frequently each mistake occurred. Respondents were able to select from the following responses:

- Almost Always (75%+)
- Often (50-74%)
- Sometimes (25-49%)
- Rarely (<25%)
- Don't know / N/A

Respondents were instructed to focus on the "frequency of occurrence in the last three years or last five projects, whichever is shorter."

Table 1 lists the 10 mistakes that respondents listed as occurring *Almost Always* or *Often*.

Table 1. Mistakes That are Most Frequently Reported to Occur *Almost Always* or *Often*

Rank	Classic Mistake	Frequency of Response
1	Overly optimistic schedules	77%
2	Unrealistic expectations	73%
3	Excessive multi-tasking	71%
4	Shortchanged quality assurance	70%
5	Noisy, crowded offices	69%
6	Feature creep	69%
7	Wishful thinking	68%
8	Insufficient risk management	68%
9	Confusing estimates with targets	65%
10	Omitting necessary tasks from estimates	61%

For a complete table of mistakes and the frequencies with which they are reported, see Appendix B, "Classic Mistakes by Reported Frequency." For descriptions of the mistakes, see Appendix G, "Classic Mistake Descriptions."

Note that the percentages in Table 1 are not the frequency of occurrence of the mistake. Our survey methodology did not make an exact determination of frequency of occurrence possible. However, by using the midpoints of the survey ranges and computing a weighted average, we were able to model an approximate frequency of occurrence of each mistake. Modeled frequencies for the 10 mistakes in Table 1 are listed in Table 2.

Table 2. Approximate Frequency of Occurrence of the Most Common Classic Mistakes

Rank	Classic Mistake	Approximate Frequency of Occurrence
1	Overly optimistic schedules	60%-70%
2	Unrealistic expectations	60%-70%
3	Shortchanged quality assurance	60%-70%
4	Noisy, crowded offices	60%-70%
5	Excessive multi-tasking	55%-65%
6	Feature creep	55%-65%
7	Insufficient risk management	55%-65%
8	Confusing estimates with targets	55%-65%
9	Wishful thinking	55%-65%
10	Omitting necessary tasks from estimates	50%-60%

For details on the modeling technique used to approximate the frequency of occurrence of these mistakes, see "How Average Frequency of Occurrence is Modeled" in Appendix A. For a complete table of mistakes and their modeled frequencies, see Appendix C, "Classic Mistakes by Modeled Average Frequency."

Some mistakes were found to occur much less often. The least frequently occurring mistakes are listed in Table 3.

Table 3. Mistakes That are Least Frequently Reported to Occur *Almost Always* or *Often*

Rank	Classic Mistake	Frequency of Response
1	Switching tools in mid-project	3%
2	Lack of automated source control	14%
3	Research-oriented development	19%
4	Premature or too frequent convergence	24%
5	Overestimating savings from tools/methods	24%
6	Push me, pull me negotiation	26%
7	Silver-bullet syndrome	26%
8	Subcontractor failure	27%
9	Letting a team go dark	28%
10	Uncontrolled problem employees	29%

Table 4 shows the approximate frequency of occurrence of these mistakes.

Table 4. Approximate Frequency of Occurrence of the Least Common Classic Mistakes

Rank	Classic Mistake	Frequency of Response
1	Switching tools in mid-project	15%-25%
2	Lack of automated source control	20%-30%
3	Research-oriented development	25%-35%
4	Premature or too frequent convergence	30%-40%
5	Overestimating savings from tools/methods	30%-40%
6	Push me, pull me negotiation	30%-40%
7	Silver-bullet syndrome	30%-40%
8	Subcontractor failure	30%-40%
9	Letting a team go dark	35%-45%
10	Uncontrolled problem employees	35%-45%

The mistakes listed in Tables 3-4 occur infrequently enough that it could be argued that they do not qualify as "classic" mistakes. If the impact of a mistake is high enough, however, it should still be considered to be a classic mistake.

Most Severe Classic Mistakes

For each mistake, respondents were also asked to judge, again based on their personal experience, how serious each mistake is when it occurs. Respondents were able to select from among the following responses:

- Catastrophic Impact
- Serious Impact
- Moderate Impact
- Hardly any Impact
- Don't know / N/A

These categories were further clarified to apply to the "impact this mistake has on a project team's ability to deliver the project on-time, within budget, with the expected features, and quality."

Notably, none of the mistakes had a modal (most frequent) response of *Catastrophic*. Nearly all of the mistakes had modal impacts of *Serious* (35 of 42). Seven mistakes had modal impacts of *Moderate*.

Table 5 lists the classic mistakes that survey respondents most frequently reported have a *Catastrophic* or *Serious* impact.

Table 5. Mistakes That are Most Frequently Reported to Produce *Catastrophic* or *Serious* Consequences When They Occur

Rank	Classic Mistake	Frequency of Response
1	Unrealistic expectations	83%
2	Weak personnel	78%
3	Overly optimistic schedules	78%
4	Wishful thinking	76%
5	Shortchanged quality assurance	72%
6	Inadequate design	72%
7	Lack of project sponsorship	71%
8	Confusing estimates with targets	71%
9	Excessive multi-tasking	71%
10	Lack of user involvement	70%

Note that the percentages reported in Table 5 are not the frequency of occurrence of these mistakes. They are the frequency with which survey respondents replied that the mistakes had *Catastrophic* or *Serious* consequences *when they occur*.

For a complete table of mistakes and the severities reported for them, see Appendix D, "Classic Mistakes by Reported Severity."

We also computed the average impact of each Classic Mistake when it occurs. Table 6 lists the average impact from the survey data.

Table 6. Average Impact of Classic Mistakes When They Occur

Rank	Classic Mistake	Average Impact
1	Unrealistic expectations	Serious
2	Weak personnel	Serious
3	Wishful thinking	Serious
4	Overly optimistic schedules	Serious
5	Lack of project sponsorship	Serious
6	Shortchanged quality assurance	Serious
7	Inadequate design	Serious
8	Lack of user involvement	Serious
9	Confusing estimates with targets	Serious
10	Excessive multi-tasking	Serious

The 10 most severe classic mistakes all had average impacts of *Serious*. For a complete table of mistakes and their average impacts, see Appendix E, "Classic Mistakes by Average Severity."

Most mistakes were found to have an average impact of *Serious*, but a few mistakes were found to have notably lower impact than the rest. Table 7 lists the Classic Mistakes that were reported by fewer than 50% of survey respondents to have *Catastrophic* or *Serious* impact.

Table 7. Mistakes That are Least Frequently Reported to Produce *Catastrophic* or *Serious* Consequences

Rank	Classic Mistake	Percent *Catastrophic* and *Serious* Responses
1	Premature or too frequent convergence	34%
2	Overestimating savings from tools/methods	39%
3	Developer gold-plating	41%
4	Wasted time in the fuzzy front end	48%
5	Adding people to a late project	48%
6	Switching tools in mid-project	49%
7	Omitting necessary tasks from estimates	49%

The first two Classic Mistakes had average impacts of *Moderate*. The other five had average impacts of *Moderate-Serious*.

Most Damaging Classic Mistakes Overall

The risk management field's concept of risk exposure helps to prioritize the Classic Mistakes based on this survey data. In risk management, Risk Exposure, also known as RE, is calculated by multiplying the likelihood of a risk by its severity. Statistically speaking, the result is the Expected Value of the risk. Sorting risks by their Risk Exposure provides a rough prioritization of the risks.

We applied a similar concept to the results of this Classic Mistakes survey to assess which Classic Mistakes are most problematic overall. We multiplied the approximate average frequency of each Classic Mistake times its average severity to arrive at a *Mistake Exposure Index* (MEI). This index ranges from 0 to 10, with 0 being the lowest exposure and 10 being the highest. Of the Classic Mistakes covered by this survey, the actual range was 2.6 to 9.9.

Table 8 shows the 10 mistakes that were found to have the highest MEIs.

Table 8. Highest Mistake Exposure Indices (MEI)

Rank	Classic Mistake	MEI
1	Unrealistic expectations	9.9
2	Overly optimistic schedules	9.6
3	Shortchanged quality assurance	9.0
4	Wishful thinking	8.9
5	Confusing estimates with targets	8.8
6	Excessive multi-tasking	8.7
7	Feature creep	8.1
8	Noisy, crowded offices	7.8
9	Abandoning planning under pressure	7.8
10	Insufficient risk management	7.8

These are essentially the worst of the Classic Mistakes overall. Table 9 summarizes the average frequencies and severities of the risks with the highest MEIs.

Table 9. Classic Mistakes with the Highest Mistake Exposure

Rank	Classic Mistake	Average Frequency	Average Severity
1	Unrealistic expectations	60%-70%	Serious
2	Overly optimistic schedules	60%-70%	Serious
3	Shortchanged quality assurance	60%-70%	Serious
4	Wishful thinking	55%-65%	Serious
5	Confusing estimates with targets	55%-65%	Serious
6	Excessive multi-tasking	55%-65%	Serious
7	Feature creep	55%-65%	Moderate-Serious
8	Noisy, crowded offices	60%-70%	Moderate-Serious
9	Abandoning planning under pressure	50%-60%	Serious
10	Insufficient risk management	55%-65%	Moderate-Serious

A few mistakes were found to occur relatively infrequently and to have relatively low severity when they do occur. It would therefore seem to be less important to guard against these mistakes than to guard against the mistakes with higher MEIs. Table 10 lists these mistakes.

Table 10. Classic Mistakes with the Lowest Mistake Exposure

Rank	Classic Mistake	Average Frequency	Average Severity
1	Switching tools in mid-project	15%-25%	Moderate-Serious
2	Lack of automated source control	20%-30%	Serious
3	Premature or too frequent convergence	30%-40%	Moderate
4	Overestimating savings from tools/methods	30%-40%	Moderate
5	Research-oriented development	25%-35%	Moderate-Serious

For a complete list of Classic Mistakes organized by MEI, see Appendix F, “Classic Mistakes by Mistake Exposure Index (MEI).”

Conclusions

Several conclusions can be drawn from the results of this survey.

- **Some mistakes are made frequently enough to be considered "classic" mistakes**. Our survey identified 20 mistakes that appear to made at least half the time.
- **The mistakes identified in this survey generally have significant impacts.** The least severe mistake in our survey had an average impact of *Moderate*, and only two mistakes were rated that low. Twenty four mistakes were rated as having *Moderate-Serious* impact, and thirteen were rated as having *Serious* impact.
- **2008's new additions to the list are significant.** Two of the ten mistakes with the highest Mistake Exposure Indexes (MEIs) were newly identified in 2007: *Confusing estimates with targets* and *Excessive multi-tasking*. These are not new phenomenon, but rather indicate that our understanding of software's classic mistakes is continuing to improve.
- **It is reasonable to characterize the mistakes surveyed as "Classic Mistakes."** Most of the mistakes included in the survey were reported to occur fairly frequently and to have significant adverse impact when they do occur. Thus it is accurate to refer to these mistakes as "mistakes that have been made so often, by so many people, that the consequences of making these mistakes should be predictable and the mistakes themselves should be avoidable."
- **A few mistakes can probably be removed from the list.** A few mistakes had both low frequency and low impact. While it is still desirable to avoid making those mistakes—just as it is desirable to avoid making any mistakes whatsoever—in the interest of restricting the list of mistakes to a manageable number, the bottom-tier mistakes should receive less attention than the top-tier mistakes.

Appendices

A. Survey Methodology

Solicitation

Survey respondents were recruited via direct e-mail solicitation using Construx's in-house email list, a blog entry by Steve McConnell, and a posting on Construx's Software Development Best Practices discussion forum. The survey itself can be found at www.construx.com/classicmistakes.

Time Frame

Survey data was collected from May 30, 2007 through July 19, 2007.

Respondents

The survey was completed by 558 respondents. Table A-1 lists the roles that respondents reported.

Table A-1. Survey Respondent Roles

Role	Percentage
Lead developer	40%
Technical lead / architect	35%
Individual contributor / developer	34%
Manager	14%
Director	5%
Individual contributor / tester	4%
Lead tester	4%
Vice President	3%

Respondents were allowed to identify more than one role, so the survey responses sum to more than 100%. The proportions of respondents in this survey approximate the proportion of workers playing the roles industry wide. We did not find any significant differences in survey responses that were correlated with different roles.

Table A-2 lists the types of software with which survey respondents are working.

Table A-2. Survey Respondent Software Types

Software Type	Percentage
Web, general (e-Commerce, web front ends, etc.)	62%
Business Systems/In house	49%
Shrink wrap/commercial (desktop applications, vertical market applications, etc.)	35%
Embedded	13%
Other	12%
System Critical	11%
Systems (OS, device drivers, etc.)	9%
SaaS	7%

Respondents were allowed to identify more than one type of software, so the survey responses sum to more than 100%. Based on these responses, these survey results are most applicable to Web software, business systems software, and shrink wrap/commercial software. We did not analyze whether there were any differences in the responses correlated with type of software.

How Average Frequency of Occurrence is Modeled

Respondents were able to select from among the following responses:

- Almost Always (75%+)
- Often (50-74%)
- Sometimes (25-49%)
- Rarely (<25%)
- Don’t know / N/A

To compute the average frequencies of occurrence, midpoints of the range for each category were used. For example, *Often* goes from 50%-74.9%, the midpoint of which is 62.5%. So for a classic mistake that a survey respondent said occurred *Often*, the number 62.5% was used as the estimated frequency of occurrence for that response. Table A-3 lists the frequencies used to calculate the approximate average frequencies.

Table A-3. Category Modeling for Frequency of Occurrence Data

Survey Category	Frequency Used to Calculate Approximate Average Frequencies
Almost Always (75%+)	87.5%
Often (50-74%)	62.5%
Sometimes (25-49%)	37.5%
Rarely (<25%)	12.5%
Don’t know / N/A	Not included

B. Classic Mistakes by Reported Frequency

Table B-1. Frequency with Which Mistakes are Reported to Occur *Almost Always* or *Often*

Mistake	Frequency of Response
Overly optimistic schedules	77%
Unrealistic expectations	73%
Excessive multi-tasking	71%
Shortchanged quality assurance	70%
Noisy, crowded offices	69%
Feature creep	69%
Wishful thinking	68%
Insufficient risk management	68%
Confusing estimates with targets	65%
Omitting necessary tasks from estimates	61%
Abandoning planning under pressure	59%
Shortchanged upstream activities	58%
Heroics	58%
Lack of user involvement	57%
Inadequate design	54%
Insufficient planning	54%
Wasted time in the fuzzy front end	52%
Planning to catch up later	51%
Weak personnel	49%
Undermined motivation	45%
Unclear project vision	44%
Requirements gold-plating	44%
Code-like-hell programming	44%
Lack of project sponsorship	42%
Politics placed over substance	37%
Adding people to a late project	36%
Friction between dev & customers	36%

Software Development's Classic Mistakes 2008

Mistake	Frequency of Response
Developer gold-plating	35%
Lack of stakeholder buy-in	33%
Trusting the map more than the terrain	32%
Assuming global development has little impact	30%
Outsourcing to reduce cost	29%
Uncontrolled problem employees	29%
Letting a team go dark	28%
Subcontractor failure	27%
Silver-bullet syndrome	26%
Push me, pull me negotiation	26%
Overestimating savings from tools/methods	24%
Premature or too frequent convergence	24%
Research-oriented development	19%
Lack of automated source control	14%
Switching tools in mid-project	3%

Note: The percentages in this table refer to the percentage of respondents who provided certain answers, not to the frequency with which each mistake occurs.

C. Classic Mistakes by Modeled Average Frequency

Table C-1. Approximate Average Frequency of Mistakes

Mistake	Approximate Average Reported Frequency
Overly optimistic schedules	60%-70%
Unrealistic expectations	60%-70%
Shortchanged quality assurance	60%-70%
Noisy, crowded offices	60%-70%
Excessive multi-tasking	55%-65%
Feature creep	55%-65%
Insufficient risk management	55%-65%
Confusing estimates with targets	55%-65%
Wishful thinking	55%-65%
Omitting necessary tasks from estimates	50%-60%
Shortchanged upstream activities	50%-60%
Heroics	50%-60%
Abandoning planning under pressure	50%-60%
Lack of user involvement	50%-60%
Inadequate design	50%-60%
Insufficient planning	50%-60%
Wasted time in the fuzzy front end	45%-55%
Planning to catch up later	45%-55%
Weak personnel	45%-55%
Unclear project vision	45%-55%
Undermined motivation	45%-55%
Requirements gold-plating	40%-50%
Code-like-hell programming	40%-50%
Lack of project sponsorship	40%-50%
Politics placed over substance	40%-50%
Developer gold-plating	40%-50%
Friction between dev & customers	40%-50%

Software Development's Classic Mistakes 2008

Mistake	Approximate Average Reported Frequency
Adding people to a late project	35%-45%
Lack of stakeholder buy-in	35%-45%
Trusting the map more than the terrain	35%-45%
Uncontrolled problem employees	35%-45%
Letting a team go dark	35%-45%
Subcontractor failure	30%-40%
Outsourcing to reduce cost	30%-40%
Assuming global development has little impact	30%-40%
Silver-bullet syndrome	30%-40%
Overestimating savings from tools/methods	30%-40%
Push me, pull me negotiation	30%-40%
Premature or too frequent convergence	30%-40%
Research-oriented development	25%-35%
Lack of automated source control	20%-30%
Switching tools in mid-project	15%-25%

D. Classic Mistakes by Reported Severity

Table D-1. Frequency with which Mistakes are Reported to Produce *Serious* or *Catastrophic* Consequences

Mistake	Frequency of Response
Unrealistic expectations	83%
Weak personnel	78%
Overly optimistic schedules	78%
Wishful thinking	76%
Shortchanged quality assurance	72%
Inadequate design	72%
Lack of project sponsorship	71%
Confusing estimates with targets	71%
Excessive multi-tasking	71%
Lack of user involvement	70%
Code-like-hell programming	68%
Unclear project vision	68%
Abandoning planning under pressure	67%
Shortchanged upstream activities	67%
Lack of automated source control	65%
Insufficient planning	64%
Heroics	62%
Subcontractor failure	61%
Lack of stakeholder buy-in	61%
Outsourcing to reduce cost	61%
Feature creep	61%
Politics placed over substance	61%
Insufficient risk management	60%
Friction between dev & customers	59%
Undermined motivation	59%
Uncontrolled problem employees	59%
Planning to catch up later	58%

Software Development's Classic Mistakes 2008

Mistake	Frequency of Response
Assuming global development has little impact	58%
Push me, pull me negotiation	56%
Requirements gold-plating	56%
Silver-bullet syndrome	56%
Research-oriented development	53%
Noisy, crowded offices	51%
Letting a team go dark	50%
Trusting the map more than the terrain	50%
Omitting necessary tasks from estimates	49%
Switching tools in mid-project	49%
Adding people to a late project	48%
Wasted time in the fuzzy front end	48%
Developer gold-plating	41%
Overestimating savings from tools/methods	39%
Premature or too frequent convergence	34%

Note: The percentages in this table refer to the percentage of respondents who provided certain answers, not to the frequency with which each mistake occurs.

E. Classic Mistakes by Average Severity

Table E-1. Average Severity of Mistakes When They Occur

Mistake	Average Severity When Mistake Occurs
Unrealistic expectations	Serious
Weak personnel	Serious
Wishful thinking	Serious
Overly optimistic schedules	Serious
Lack of project sponsorship	Serious
Shortchanged quality assurance	Serious
Inadequate design	Serious
Lack of user involvement	Serious
Confusing estimates with targets	Serious
Excessive multi-tasking	Serious
Abandoning planning under pressure	Serious
Code-like-hell programming	Serious
Unclear project vision	Serious
Lack of automated source control	Serious
Shortchanged upstream activities	Serious
Heroics	Serious
Politics placed over substance	Moderate-Serious
Friction between dev & customers	Moderate-Serious
Outsourcing to reduce cost	Moderate-Serious
Insufficient planning	Moderate-Serious
Feature creep	Moderate-Serious
Subcontractor failure	Moderate-Serious
Uncontrolled problem employees	Moderate-Serious
Lack of stakeholder buy-in	Moderate-Serious
Insufficient risk management	Moderate-Serious
Undermined motivation	Moderate-Serious
Planning to catch up later	Moderate-Serious

Mistake	Average Severity When Mistake Occurs
Assuming global development has little impact	Moderate-Serious
Silver-bullet syndrome	Moderate-Serious
Research-oriented development	Moderate-Serious
Push me, pull me negotiation	Moderate-Serious
Requirements gold-plating	Moderate-Serious
Trusting the map more than the terrain	Moderate-Serious
Letting a team go dark	Moderate-Serious
Noisy, crowded offices	Moderate-Serious
Omitting necessary tasks from estimates	Moderate-Serious
Wasted time in the fuzzy front end	Moderate-Serious
Adding people to a late project	Moderate-Serious
Switching tools in mid-project	Moderate-Serious
Developer gold-plating	Moderate-Serious
Overestimating savings from tools/methods	Moderate
Premature or too frequent convergence	Moderate

F. Classic Mistakes by Mistake Exposure Index (MEI)

Table F-1. Complete List of Mistake Exposure Indices

Mistake	MEI
Unrealistic expectations	9.9
Overly optimistic schedules	9.6
Shortchanged quality assurance	9.0
Wishful thinking	8.9
Confusing estimates with targets	8.8
Excessive multi-tasking	8.7
Feature creep	8.1
Noisy, crowded offices	7.8
Abandoning planning under pressure	7.8
Insufficient risk management	7.8
Heroics	7.7
Shortchanged upstream activities	7.6
Inadequate design	7.6
Lack of user involvement	7.6
Weak personnel	7.4
Insufficient planning	7.2
Omitting necessary tasks from estimates	7.2
Planning to catch up later	6.9
Code-like-hell programming	6.9
Unclear project vision	6.9
Lack of project sponsorship	6.8
Wasted time in the fuzzy front end	6.6
Politics placed over substance	6.4
Requirements gold-plating	6.3
Undermined motivation	6.3
Friction between dev & customers	6.1
Trusting the map more than the terrain	5.8

Software Development's Classic Mistakes 2008

Mistake	MEI
Outsourcing to reduce cost	5.7
Assuming global development has little impact	5.6
Adding people to a late project	5.5
Lack of stakeholder buy-in	5.5
Uncontrolled problem employees	5.4
Push me, pull me negotiation	5.3
Subcontractor failure	5.2
Letting a team go dark	5.2
Silver-bullet syndrome	5.1
Developer gold-plating	5.1
Research-oriented development	4.8
Overestimating savings from tools/methods	4.4
Premature or too frequent convergence	4.3
Lack of automated source control	3.9
Switching tools in mid-project	2.6

G. Classic Mistake Descriptions

Table G-1. Classic Mistake Descriptions

Mistake	Description
Abandonment of planning under pressure	Projects make plans and then routinely abandon them when they run into schedule trouble. This would not be a problem if the plans were updated to account for the schedule difficulties. The problem arises when the plans are abandoned with no substitute, which tends to make the project slide into code-and-fix mode.
Adding people to a late project	When a project is behind, adding people can take more productivity away from existing team members than it adds through new ones. Adding people to a late project has been likened to pouring gasoline on a fire.
Assuming global development has a negligible impact on total effort	Multi-site development increases communication and coordination effort between sites. The greater the differences among the sites in terms of time zones, company cultures, and national cultures, the more the total project effort will increase. Some companies naively assume that changing from single-site development to multi-site development will have a negligible impact on effort, but studies have shown that international development will typically increase effort by about 40% compared to single-site development.
Code-like-hell programming	Some organizations think that fast, loose, all-as-you-go coding is a route to rapid development. If the developers are sufficiently motivated, they reason, they can overcome any obstacles. This is far from the truth. The entrepreneurial model is often a cover for the old code-and-fix paradigm combined with an ambitious schedule, and that combination almost never works.
Confusing estimates with targets	Some organizations set schedules based purely on the desirability of business targets without also creating analytically-derived cost or schedule estimates. While target setting is not bad in and of itself, some organizations actually refer to the target as the 'estimate,' which lends it an unwarranted and misleading authenticity as a foundation for creating plans, schedules, and commitments.
Developer gold-plating	Developers are fascinated by new technology and are sometimes anxious to try out new capabilities of their language or environment or to create their own implementation of a slick feature they saw in another product—whether or not it's required in their product. The effort required to design, implement, test, document, and support features that are not required adds cost and lengthens the schedule.
Excessive multi-tasking	When software developers are assigned to more than one project, they must 'task switch' as they change their focus from one project to another. They must get out of 'flow' on one project and into 'flow' on another. Task switching can be a significant factor—some studies have said that each task switch in software development can incur a 5-30 minute downtime as a developer works out of flow on one project and works into flow on the other.

Mistake	Description
Feature creep	The average project experiences about a 25-percent change in requirements over its lifetime. Such a change produces at least a 25-percent addition to the software effort and schedule, which is often unaccounted for in the project's plans and unacknowledged in the project's status reports.
Friction between developers and customers	Friction between developers and customers can arise in several ways. Customers may feel that developers are not cooperative when they refuse to sign up for the development schedule that the customers want or when they fail to deliver on their promises. Developers may feel that customers are unreasonably insisting on unrealistic schedules or requirements changes after requirements have been baselined. There might simply be personality conflicts between the two groups. The primary effect of this friction is poor communication, and the secondary effects of poor communication include poorly understood requirements, poor user-interface design, and, in the worst case, customers' refusing to accept the completed product.
Heroics	Some project teams place a high emphasis on project heroics, thinking that the certain kinds of heroics can be beneficial. However, emphasizing heroics in any form usually does more harm than good. Sometimes there is a higher premium placed on a can-do attitudes than on steady and consistent progress and meaningful progress reporting. By elevating can-do attitudes above accurate-and-sometimes-gloomy status reporting, such project managers undercut their ability to take corrective action. They don't even know they need to take corrective action until the damage has been done. Can-do attitudes can escalate minor setbacks into true disasters. An emphasis on heroics can encourage extreme risk taking and discourage cooperation among the many stakeholders in the software development process.
Inadequate design	A special case of shortchanging upstream activities is inadequate design. Rush projects undermine design by not allocating enough time for it and by creating a pressure-cooker environment that makes thoughtful consideration of design alternatives difficult. This results in going through several time-consuming design cycles before the system can be completed.
Insufficient planning	Planning can be done well, and planning can be done poorly. But some projects suffer from simply not doing enough planning at all, i.e., not prioritizing planning as an important activity.
Insufficient risk management	Some mistakes have been made often enough to be considered classic mistakes. Other potential problems need to be identified project-by-project through risk management. The most common problem with risk management is not doing any risk management at all. The second most common problem with risk management is not doing enough risk management.

Mistake	Description
Lack of automated source-code control	Failure to use automated source-code control exposes projects to needless risks. Without it, if two developers are working on the same part of the program, they have to coordinate their work manually and risk accidentally overwriting someone else's work. People develop new code to out-of-date interfaces and then have to redesign their code when they discover that they were using the wrong version of the interface. Users report defects that you can't reproduce because you have no way to recreate the build they were using.
Lack of effective project sponsorship	High-level project sponsorship is necessary to support many aspects of effective development including realistic estimates, adequate resource allocation, and achievable schedules, as well as helping to clear roadblocks once the project is underway. Without an effective project sponsor, other high-level personnel in your organization can force you to accept unrealistic deadlines or make changes that undermine your project.
Lack of stakeholder buy-in	All of the major players in a software-development effort must buy in to the project. That includes the executive sponsor, team leader, team members, marketing, end-users, customers, and anyone else who has a stake in it. The close cooperation that occurs only when you have complete buy-in from all stakeholders allows for precise coordination of a software development effort that is impossible to attain without good buy-in.
Lack of user involvement	User involvement is necessary for defining meaningful requirements. The degree of user involvement can affect how quickly or how slowly issues get resolved.
Letting a team go dark	On some projects, management allows a team to work with little oversight and little visibility into the team's progress. This is known as "letting a team go dark." This practice restricts visibility into project status, and the project doesn't receive timely warnings of impending schedule slips. Before you can *keep* a project on track, you have to be able to tell *whether* it's on track, and letting a team go dark prevents that.
Noisy, crowded offices	About 60 percent of developers report that their work environments are neither sufficiently quiet nor sufficiently private. For many developers, this can prevent concentration and prevent achieving a state of 'flow' that is helpful in achieving high levels of productivity. Workers who occupy quiet, private offices tend to perform significantly better than workers who occupy noisy, crowded work bays or cubicles.
Omitting necessary tasks from estimates	If people don't keep careful records of previous projects, they forget about the less visible tasks, but those tasks add up. Forgotten activities can add 20 to 30 percent to a development schedule.

Mistake	Description
Outsourcing to reduce cost	Valid reasons to outsource include accessing capabilities that you don't have in house, diversifying your labor force, freeing up your in-house staff to focus on mission-critical or core-competency projects, adding "surge capacity" to your development staff, and supporting around-the-clock development. Many organizations that have outsourced for these reasons have accomplished their objectives. However, some organizations outsource primarily to reduce development costs, and historically those initiatives have not succeeded. Usually outsourcing motivated by cost savings results in higher costs and longer schedules.
Overestimated savings from new tools or methods	Organizations often assume that first-time usage of a new tool or method will reduce costs and shorten schedules. In reality, first-time use of a new tool or method tends to be subject to a learning curve, and the safest planning assumption is to assume a short-term *increase* in cost and schedule before the benefits of the new tool or method kick in.
Overly optimistic schedules	The challenges faced by someone building a three-month application are quite different than the challenges faced by someone building a one-year application. Setting an overly optimistic schedule sets a project up for failure by under scoping the project, undermining effective planning, and abbreviating critical upstream development activities such as requirements analysis and design. It also puts excessive pressure on developers, which hurts developer morale and productivity.
Planning to catch up later	If you're working on a project and it takes you four weeks to meet your first two-week milestone, what's your status? You're know that you're behind schedule, but will you stay behind schedule, or will you catch up later? Project planners commonly plan to catch up later, but they rarely do. Most projects that get behind schedule stay behind schedule.
Politics placed over substance	Larry Constantine reported on four teams that had four different kinds of political orientations. "Politicians" specialized in "managing up"—concentrating on relationships with their managers. "Researchers" concentrated on scouting out and gathering information. "Isolationists" kept to themselves, creating project boundaries that they kept closed to non-team members. "Generalists" did a little bit of everything: they tended their relationships with their managers, performed research and scouting activities, and coordinated with other teams through the course of their normal workflow. Constantine reported that initially the political and generalist teams were both well regarded by top management. But after a year and a half, the political team was ranked dead last. Putting politics over results is fatal to software development effectiveness.

Mistake	Description
Premature or too frequent convergence	Shortly before a public software release there is a push to prepare the software for release—improve the product's performance, create final documentation, stub out functionality that's not going to be ready for the release, perform end-to-end testing including tests that can't be automated, test the setup program, and so on. On rush projects, there is a tendency to force convergence too early. If the software isn't close enough to a releasable state, the attempted convergence will fail, and the team will need to attempt to converge again later in the project. The extra convergence attempts waste time and prolong the schedule.
Push me, pull me negotiation	One bizarre negotiating ploy occurs when a manager approves a schedule slip on a project that's progressing slower than expected and then adds completely new tasks after the schedule change. The underlying reason for this is hard to fathom because the manager who approves the schedule slip is implicitly acknowledging that the schedule was in error. But once the schedule has been corrected, the same person takes explicit action to make it wrong again.
Requirements gold-plating	Requirements gold plating is the addition of requirements or the expansion of requirements without a clear business justification. Requirements gold plating can be done by end users who want the "system to end all systems" or it can be done by developers who are sometimes more interested in complex capabilities than real users are.
Research-oriented development	Some projects have goals that push the state of the art—algorithms, speed, memory usage, and so on. That's fine, but when those projects *also* have ambitious cost or schedule goals, the combination of advancing the state of the art with a tight budget on a short schedule isn't achievable.
Shortchanged quality assurance	Projects that are in a hurry often cut corners by eliminating design and code reviews, eliminating test planning, and performing only perfunctory testing. It is common for design reviews and code reviews to be given short shrift in order to achieve a perceived schedule advantage. This often results in the project reaching its feature-complete milestone but then still being too buggy to release.
Shortchanged upstream activities	Projects sometimes cut out non-coding activities such as requirements, architecture, and design. Also known as "jumping into coding," the results of this mistake are predictable. Projects that skimp on upstream activities typically have to do the same work downstream at anywhere from 10 to 100 times the cost of doing it earlier.
Silver-bullet syndrome	On some projects, there is an over reliance on the advertised benefits of previously unused technologies, tools, or 3rd party applications and too little information about how well they would do in the current development environment. When project teams latch onto a single new methodology or new technology and expect it to solve their cost, schedule, or quality problems, they are inevitably disappointed.

Mistake	Description
Subcontractor failure	Companies sometimes subcontract pieces of a project when they are too rushed to do the work in-house. ("Subcontractor" can refer either to an individual or to an outsourcing firm.) But subcontractors frequently deliver work that's late, that's of unacceptably low quality, or that fails to meet specifications. Risks such as unstable requirements or ill-defined interfaces can be magnified when you bring a subcontractor into the picture. If the subcontractor relationship isn't managed carefully, the use of subcontractors can undermine a project's goals. (Note: This question deals specifically with subcontracting *part* of a project—other items focus on outsourcing full projects.)
Switching development tools in the middle of a project	This is an old standby that hardly ever works. Sometimes it can make sense to upgrade incrementally within the same product line, from version 3 to version 3.1 or sometimes even to version 4. But the learning curve, rework, and inevitable mistakes made with a totally new tool usually cancel out any benefit when you're in the middle of a project.
Trusting the map more than the terrain	Project teams sometimes invest more confidence in the plans they create than in the experience their project is giving them. Sometimes they will trust the delivery date written on a plan more than the delivery date implied by the project's track record. If the project reality and the project plans disagree, the project's reality is correct, and the plans *must be* wrong. The longer a project team trusts the plans rather than the project reality—i.e., trusts the map more than the terrain—the more difficulty they will have adapting their course successfully.
Unclear project vision	The lack of clearly defined and communicated vision undermines the organization's ability to make and execute project-level plans that are consistent with organization-level goals. Without a clear understanding of the vision, people draw their own conclusions about the purpose of the project and how it relates to their day-to-day work, and they make decisions that run counter to the project's business objectives. The unclear vision contributes to changes in project direction, including detailed requirements; detailed plans that are misaligned with project priorities; and, ultimately, inability to meet schedule commitments.
Uncontrolled problem employees	Failure to deal with problem personnel (e.g., a prima donna programmer) can threaten development effectiveness. This is a common problem and has been well-understood at least since Gerald Weinberg published *Psychology of Computer Programming* in 1971. Failure to take action to deal with a problem employee is the most common complaint that team members have about their leaders.
Undermined motivation	Study after study has shown that motivation probably has a larger effect on productivity and quality than any other factor. On some projects, management can undermine morale throughout the project. Examples include giving a hokey pep talk, going on a long vacation while the team works through the holidays, and providing bonuses that work out to less than a dollar per overtime hour at the end.

Mistake	Description
Unrealistic expectations	One of the most common causes of friction between developers and their customers or managers is unrealistic expectations. Often customers simply start with unrealistic expectations (which is probably just human nature). Sometimes project managers or developers ask for trouble by getting project approval based on optimistic estimates. A Standish Group survey listed realistic expectations as one of the top five factors needed to ensure the success of an in-house business-software project.
Wasted time during the fuzzy front end	The “fuzzy front end” is the time before the project starts, the time normally spent in the approval and budgeting process. It’s not uncommon for a project to spend months or years in the fuzzy front end and then to come out of the gates with an aggressive schedule.
Weak personnel	After motivation, either the individual capabilities of the team members or their relationship as a team probably has the greatest influence on productivity. Hiring from the bottom of the barrel can threaten a development effort. On some projects, personnel selections were made with an eye toward who could be hired fastest instead of who would get the most work done over the life of the project. That practice gets the project off to a quick start but doesn’t set it up for successful completion.
Wishful thinking	Wishful thinking isn’t just optimism. It’s closing your eyes and hoping something works when you have no reasonable basis for thinking it will. Wishful thinking at the beginning of a project leads to big blowups at the end of a project. It undermines meaningful planning and can be at the root other problems.”

H. Classic Mistakes Summary, Alphabetical

Table H-1. Classic Mistakes Summary, Alphabetical

Mistake	MEI	*Almost Always* and *Often* Responses	*Almost Always* Responses	Modal Frequency	Modeled Average Frequency	*Catastrophic* and *Serious* Responses	*Catastrophic* Responses	Modal Severity	Average Severity
Abandoning planning under pressure	7.8	59%	25%	Often	50%-60%	67%	21%	Serious	Serious
Adding people to a late project	5.5	36%	12%	Sometimes	35%-45%	48%	10%	Moderate	Moderate-Serious
Assuming global development has little impact	5.6	30%	10%	Rarely	30%-40%	58%	16%	Serious	Moderate-Serious
Code-like-hell programming	6.9	44%	15%	Sometimes	40%-50%	68%	21%	Serious	Serious
Confusing estimates with targets	8.8	65%	36%	Almost always	55%-65%	71%	18%	Serious	Serious
Developer gold-plating	5.1	35%	8%	Sometimes	40%-50%	41%	5%	Moderate	Moderate-Serious
Excessive multi-tasking	8.7	71%	34%	Often	55%-65%	71%	17%	Serious	Serious
Feature creep	8.1	69%	32%	Often	55%-65%	61%	13%	Serious	Moderate-Serious
Friction between dev & customers	6.1	36%	13%	Sometimes	40%-50%	59%	24%	Serious	Moderate-Serious
Heroics	7.7	58%	26%	Often	50%-60%	62%	19%	Serious	Serious

Software Development's Classic Mistakes 2008

Mistake	MEI	*Almost Always* and *Often* Responses	*Almost Always* Responses	Modal Frequency	Modeled Average Frequency	*Catastrophic* and *Serious* Responses	*Catastrophic* Responses	Modal Severity	Average Severity
Inadequate design	7.6	54%	20%	Sometimes	50%-60%	72%	21%	Serious	Serious
Insufficient planning	7.2	54%	21%	Often	50%-60%	64%	14%	Serious	Moderate-Serious
Insufficient risk management	7.8	68%	34%	Often	55%-65%	60%	10%	Serious	Moderate-Serious
Lack of project sponsorship	6.8	42%	12%	Sometimes	40%-50%	71%	26%	Serious	Serious
Lack of automated source control	3.9	14%	6%	Rarely	20%-30%	65%	32%	Serious	Serious
Lack of stakeholder buy-in	5.5	33%	6%	Sometimes	35%-45%	61%	13%	Serious	Moderate-Serious
Lack of user involvement	7.6	57%	21%	Often	50%-60%	70%	20%	Serious	Serious
Letting a team go dark	5.2	28%	10%	Sometimes	35%-45%	50%	14%	Serious	Moderate-Serious
Noisy, crowded offices	7.8	69%	45%	Almost always	60%-70%	51%	8%	Serious	Moderate-Serious
Omitting necessary tasks from estimates	7.2	61%	26%	Often	50%-60%	49%	5%	Moderate	Moderate-Serious
Outsourcing to reduce cost	5.7	29%	10%	Rarely	30%-40%	61%	25%	Serious	Moderate-Serious

Software Development's Classic Mistakes 2008

Mistake	MEI	*Almost Always* and *Often* Responses	*Almost Always* Responses	Modal Frequency	Modeled Average Frequency	*Catastrophic* and *Serious* Responses	*Catastrophic* Responses	Modal Severity	Average Severity
Overestimating savings from tools/methods	4.4	24%	4%	Rarely	30%-40%	39%	4%	Moderate	Moderate
Overly optimistic schedules	9.6	77%	40%	Almost always	60%-70%	78%	24%	Serious	Serious
Planning to catch up later	6.9	51%	20%	Often	45%-55%	58%	10%	Serious	Moderate-Serious
Politics placed over substance	6.4	37%	15%	Sometimes	40%-50%	61%	24%	Serious	Moderate-Serious
Premature or too frequent convergence	4.3	24%	5%	Rarely	30%-40%	34%	3%	Moderate	Moderate
Push me, pull me negotiation	5.3	26%	6%	Rarely	30%-40%	56%	12%	Serious	Moderate-Serious
Requirements gold-plating	6.3	44%	14%	Sometimes	40%-50%	56%	9%	Serious	Moderate-Serious
Research-oriented development	4.8	19%	4%	Rarely	25%-35%	53%	18%	Serious	Moderate-Serious
Shortchanged quality assurance	9.0	70%	40%	Almost always	60%-70%	72%	21%	Serious	Serious
Shortchanged upstream activities	7.6	58%	23%	Often	50%-60%	67%	19%	Serious	Serious
Silver-bullet syndrome	5.1	26%	4%	Sometimes	30%-40%	56%	14%	Serious	Moderate-Serious

Software Development's Classic Mistakes 2008

Mistake	MEI	*Almost Always* and *Often* Responses	*Almost Always* Responses	Modal Frequency	Modeled Average Frequency	*Catastrophic* and *Serious* Responses	*Catastrophic* Responses	Modal Severity	Average Severity
Subcontractor failure	5.2	27%	8%	Rarely	30%-40%	61%	18%	Serious	Moderate-Serious
Switching tools in mid-project	2.6	3%	0%	Rarely	15%-25%	49%	16%	Serious	Moderate-Serious
Trusting the map more than the terrain	5.8	32%	9%	Rarely	35%-45%	50%	15%	Moderate	Moderate-Serious
Unclear project vision	6.9	44%	14%	Sometimes	45%-55%	68%	18%	Serious	Serious
Uncontrolled problem employees	5.4	29%	9%	Sometimes	35%-45%	59%	14%	Serious	Moderate-Serious
Undermined motivation	6.3	45%	14%	Sometimes	45%-55%	59%	9%	Serious	Moderate-Serious
Unrealistic expectations	9.9	73%	40%	Almost always	60%-70%	83%	32%	Serious	Serious
Wasted time in the fuzzy front end	6.6	52%	22%	Sometimes	45%-55%	48%	10%	Moderate	Moderate-Serious
Weak personnel	7.4	49%	17%	Sometimes	45%-55%	78%	27%	Serious	Serious
Wishful thinking	8.9	68%	30%	Often	55%-65%	76%	26%	Serious	Serious

Further Information

This white paper was created by Construx Software Builders, Inc. For more information about Construx's support for software development best practices, please see our website at www.construx.com, contact us at consulting@construx.com, or call us at +1(866) 296-6300.

Steve McConnell
CEO/Chief Software Engineer

steve.mcconnell@construx.com
+1(425) 636-0100

Jenny Stuart
VP Consulting

jenny.stuart@construx.com
+1(425) 636-0108

Matt Peloquin
CTO

matt.peloquin@construx.com
+1(425) 636-0104

Steve Tockey
Principal Consultant

steve.tockey@construx.com
+1(425) 636-0106

Mark Nygren
COO/VP Sales

mark.nygren@construx.com
+1(425) 636-0110

Construx®
SOFTWARE DEVELOPMENT BEST PRACTICES