Obsessively Thorough

Software Design

A Guide to **Interface**Driven**Architecture**

Obsessively Thorough

Software Design

Hal Helms and Clark Valberg

Contents

CHAPTER 1

When Bad Things Happen to Good Projects

We might as well get the bad news out right away. In a widely-cited report, The Standish Group disclosed the results of a thorough study on custom software applications—including websites and Web applications.

Aptly named, "The CHAOS Report" revealed that, in 2004, custom software projects were delivered on time and on budget only 29% of the time. Outright project failures accounted for 18% of all attempted projects. The remaining 53% were charitably deemed "challenged"—with cost overruns averaging 43% more than the budgeted amount and with time overruns up to 180% greater than time originally allocated.

This Software Project Survival Guide was written to help you avoid the common fate of so many software projects by helping you understand the recurring causes for such failure.

An Onset of Can-Do Flu

It's not that software developers want to fail: we're as frustrated as anyone when yet another project falls into the "challenged" category or, worse, becomes an outright failure. So why does it keep happening?

Consider how software project contracts are granted. In the typical scenario, several software development firms will be asked to offer a bid on a project. Working with a little information—a Request for Proposal, perhaps, or simply from an interview with the client,—the development firm is asked to commit themselves to a firm, fixed bid for the entire project.

At exactly the point where the software firm knows almost nothing about what customer requirements will truly be, they are asked for a fixed price. At exactly the point where the risk is maximal, the development firm is asked to commit to a set price and a firm date.

It might seem that the developers would logically seek to mitigate their risk by charging as much as possible and promising as little as possible. But developers are prone to a work-related illness: the Can-Do Flu. Symptoms of the disease are a willingness to accept any challenges and make any promises in order to get the job.

We Have Another Term for it…

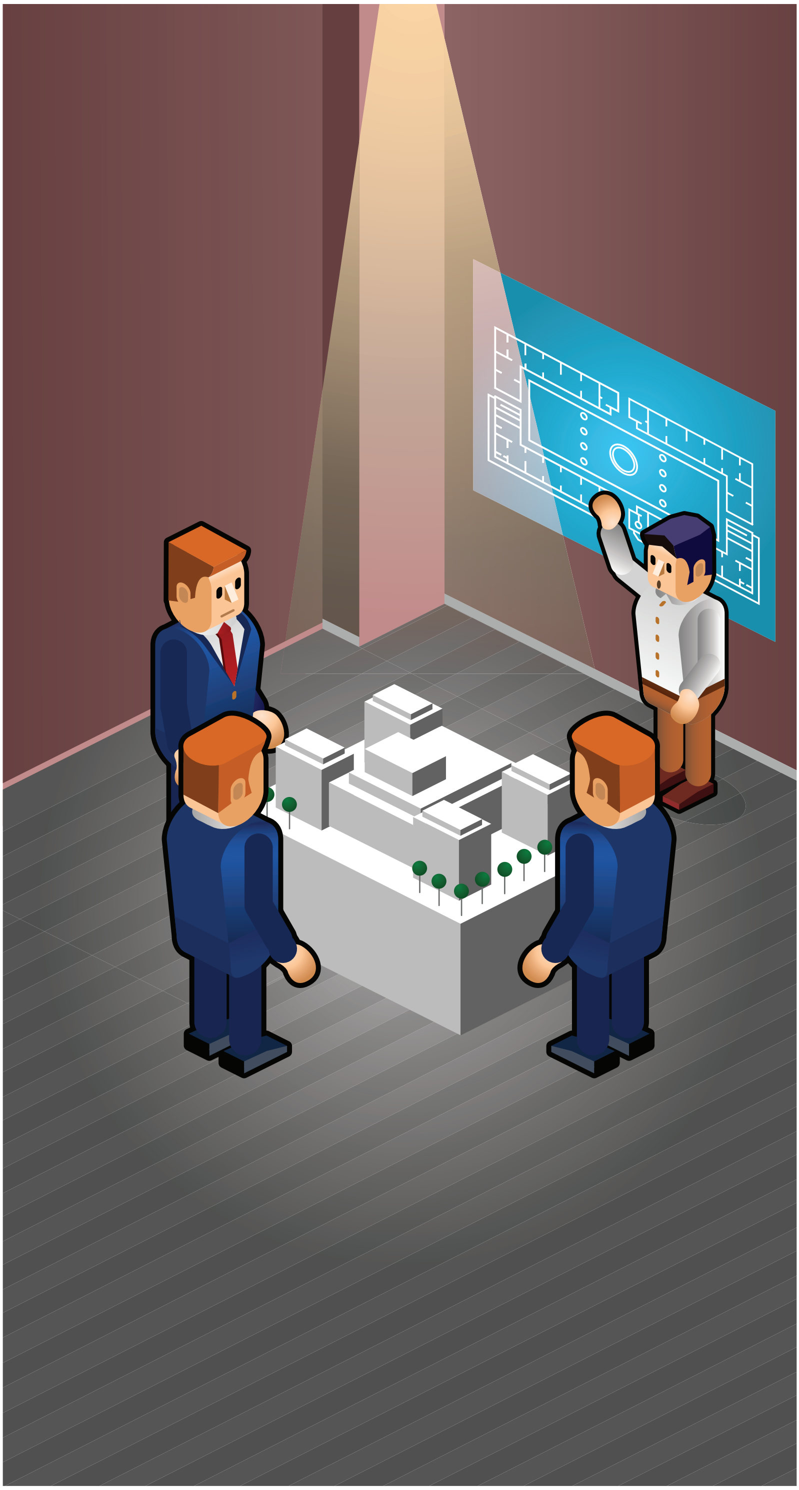
Knowing that they will be competing for your business against other developers similarly afflicted, they will happily provide you with a bid that includes a fixed price and a firm delivery date. In the industry, we refer to these as "back-of-the-envelope bids." And, unfortunately, they're usually worth the paper that envelope is constructed from.

Now, developers aren't really quite that foolish. They know that once they have the job, they have a truly effective risk-mitigation strategy: the Change Order. Since they don't know enough to provide a fully specified bid, the language of the proposal is necessarily vague. When, during the course of building the software, they discover something they did not anticipate, the change order provides convenient cover.

The Case of the Disappearing Developer

In our scenario, one of the firms is granted the project. Now, something curious happens: after a few initial meetings, the developer disappears! No more meetings, only occasional phone calls and possibly a few emails. You are not overly concerned—you've discussed the requirements with the developer. They may have interviewed some of the people who will use the software.

Custom software projects were delivered on time and on budget only 29% of the time.



Now they have to go produce the code.

What you don't know—and what the developer likely has no idea of—is that the success or failure of your project is already determined *before a single line of code is written*. The root cause of failure is ignorance.

A piece of software that both meets your current needs and has the flexibility to adapt easily to new ones is based on a scale model of your business. It needs to reflect both the formal knowledge of your business and the techniques and policies that make your business uniquely yours—a sort of "tribal knowledge."

This kind of information—vital to the success of your project—cannot be gleaned from a few meetings and the occasional phone call. The user requirements the developer has uncovered are but the tip of the iceberg. What lies hidden beneath are the real needs of your business. Given this, is it any wonder that only 29% of custom software succeeds?

Project Timewarp

While the developers are hard at work on your project, time advances. At some point, you'll start making inquiries about the state of the project. It's a running joke within the industry that at any point when a developer is asked, "How far along are you?", s/he will answer: 90%. Indeed: the first 90% of the job only took six weeks; it's the final 10% that can run into many more months.

There's a good reason for this project timewarp: the first 90% of the job was the easy part. It's the "techie" part developers enjoy—designing databases, writing software modules, and so forth. All of this would be fine, if the developers were the ultimate judges of the project's success. But, they're not. Users are. And so history repeats itself: developers spend their time and your money making software that users will not use.

With each new day come new questions developers must face about how your software should function. But removed from the process are the only people who can and should answer these decisions: users. And without these users, developer must make guesses about how your business works and how the software should behave.

These are the dog days of the project, where developers are caught in the quagmire of absent user requirements. Long gone are the halcyon days where lofty promises were made. We've entered the phase of the project known as "The Death March." The goal now is to deliver *something*—and get on with the next project that's already behind schedule.

"Anyway", the developers say to themselves, "the client is bugging us to deliver the software and, besides, this project was underbid. What can they expect?" A jibe all developers know is that a project will run out of time and money long before it runs out of excuses.

A project will run out of time and money long before it runs out of excuses

Ta-Da Time!

It's time to deliver the software. After many weeks of patient waiting, the users will get a chance to see the software designed to make them more productive and even open new opportunities for the business.

A meeting to demonstrate the new software is held. There's palpable excitement as the first screen is displayed. Then the next screen. And the next. The developer races through the features of the software, ignoring the blank looks of the users.

"Is this what we're actually going to use?" a bold user may inquire. Another may venture a suggestion: "You know what would be nice..." And now, for the first time in the project, the developer gets real user requirements—when there's neither time nor money to make the software work.

Now, things aren't always this bleak, of course. Sometimes, the developers get it just right. Actually, that happens 29% of the time. The problem is that we've involved real users exactly twice in the process— at the beginning of the project, when it was too early for them to give us any meaningful feedback, and at the end—when it's too late to make use of that feedback.

We said earlier that the success or failure of a project was determined before the first line of code is written. Why? Because the die was cast when the developers proceeded before the true requirements were known. This is, far and away, the greatest single cause for software project failure. In fact, Dr. Ralph Young, of Northrup Grumman Information Technology Division, estimates that 85% of defects in developed software originate in failed requirements.

Time for Some Good News?

The good news is that the causes of software project failure are well known—and avoidable. The rest of this Survival Guide explains how you can make sure your project is in that elite 29% of successes.

Beating the Odds: Conducting Successful Software Projects

While it's true that over 70% of software projects involve some significant degree of failure, that number is not spread evenly across all software production methodologies. In the last several years, a new and highly successful approach to software project management has emerged under the banner of "agile software practices."

CHAPTER 2

So, Now What?

The Problem with Programmers

Traditional Software Engineering places great emphasis on writing code that machines can run without ambiguity. The history of successful technologies can be traced as a move from technology-centric to user-centric. This has been true of televisions, cameras, and cars. As the technology improves, more time and effort are available to improve the user experience. This should also be true of software projects.

What we might call "the programmer mindset" could be characterized in this statement: the *code* is the application. Users, on the other hand take

"A picture is worth a thousand words. An interface is worth a thousand pictures."

Ben Shneiderman, *Computer Scientist* a very different view: to them, the *interface* is the application.

The problem with the programmer mindset is that, while it produces high-quality code, it too often produces programs that users find unusable. It's as if an auto-maker produced a car that required three arms to drive. The car might be a technological tour-de-force, but the audience that would find such a car usable would be…limited.

Interface-Driven Architecture

Forty years of Software Engineering experience indicate that it's much cheaper to change a product early in the development process than it is to make changes later in the process. The most common estimate is that it's 100 times cheaper to make a change before any code is written than it is to wait until the implementation is complete. —Jakob Nielsen, Usability Expert

The technology for creating software is orders of magnitude better than it was 40 years, ago but our Software Engineering methodologies don't reflect this. Software engineers use the derisive term, "lipstick", to refer to the user experience (the user interface). It is still the last thing to be considered and is often relegated to the most junior of developers.

Traditional development methods spend a small amount of time creating project requirements. Then, code work is begun. After the bulk of time and money is devoted to code, the "lipstick" is applied and the application is delivered.

Perhaps the biggest change that agile methodologies have given us is a new understanding of the power that comes from creating the *user interface first*. Interface-Driven Architecture (IDA), as this practice is known, offers a process that lets a variety of project stakeholders (including developers) discover what the real needs of the project are—before any code is written.

What makes IDA projects so successful is that *non*-interface issues are resolved during the interface process. Features, usability, understanding the business processes—all these are discoverable through the interface process. There is no expectation that the first user interface attempt will be perfect. Instead, iteration is built into the IDA process. But since no code is involved, the cost of each iteration is very small. When interfaces are used this way, rather than as something "pasted" onto the underlying code, the chances for a successful software deployment soar.

"Design is not just what it looks like and feels like. Design is how it works."

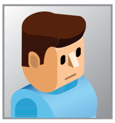
***Steve Jobs**, Apple CEO*

Different Stakeholders, Different Needs

Successful software must work on multiple levels. Each stakeholder in the project has different needs, and good software addresses them all:

Administrators

are concerned that the software have the features needed to help them do their existing jobs.



Managers

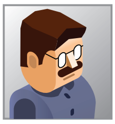
want the software to provide increased user productivity and to offer the business new opportunities—either to generate new revenues or to save on expenses.

End users

if they are not to be frustrated, want the software to score high in usability.

Funders or investors

prefer working software over Powerpoint presentations and need realistic costs in order to make a decision about their return on investment.



Business consultants

wish to be provide business advice that will influence the way the software works.

By using the Interface-Driven Architecture approach, each stakeholder is able to see what the finished application will look like. Their valid concerns, feedback, and suggestions can be incorporated before the expense of coding the application is undertaken.

Rethinking the Fixed Bid

All software production firms are accustomed to their clients' understandable desire for a fixed bid on a project. The problem is that so little is truly understood about the final project requirements at the time the bid must be placed that the fixed bid is necessarily little more than a "guesstimate". This no-win situation forces developers to adopt the goals of:

(A) getting the job and

(B) using change orders to bring the bid into alignment with the scope of actual work.

Interface-Driven Architecture practitioners find the adversarial nature of this relationship with clients unacceptable. Instead, they work with the different stakeholders to create the user interface *first*. The process of creating the interface is highly iterative, involving multiple versions of the interface. A new version of the interface is produced when any of the following triggers is encountered:

1. client feedback
2. discovery of new project requirements
3. negative usability test results conducted with real target users

The best time to give a fixed bid on the software is at the end of the interface process.

A firm using IDA can provide a good estimate of the cost.

How many versions are produced? As many as it takes to get things right.

The problem with all this is pricing. The best time to give a fixed bid on the software is at the *end* of the interface process, since each new version further defines exactly what the software development firm must deliver. Because so much more is known about the real goals and requirements of the software after the interface is complete, there is very little risk of changing requirements, allowing the developer to bid on a highly-specified project, eliminating the change orders that are the bane of software projects.

Two Are Better Than One

IDA solves the pricing problem by breaking the project pricing into two distinct phases, each with their own separate steps:

Phase I: Architecture

1. Analysis
2. Interface Design
3. Usability Testing

Phase II: Fabrication

4. Software Engineering
5. Coding
6. BETA Testing
7. Deployment

*. Monitoring

The greatest reason for failed software projects is that the software delivered does not do what users need.

A fixed bid is prepared for Phase I: **Architecture.** Because none of these steps involve the time and expense of code, clients can work through the complete discovery and specification of the project (through the interface) at relatively low cost.

While going through interface iterations, a firm using IDA can provide a good estimate of the cost (in both time and money) of fabricating the software for each iteration. This allows organizations to keep to a budget without needing to share that budget with the developer. This eliminates the "all or nothing" aspect of fixed bids for the entire project.

After all project stakeholders agree that the user interface is complete, another fixed bid is issued for the Phase II: **Fabrication** of the architected application in code.

This two-phased approach provides several options to the client. They may, of course, continue with the original development firm, contracting with them for fabrication of the architecture..

They may also put the fabrication out for bid. Since any firm asked to provide a bid has highly detailed information, they can apply their "sharpest pencil" to the bid. In practice, we find that the combined cost of both phases to be considerably less (15-20% on average) than a single bid given at the point of minimal information—the traditional method.

Finally, the client may elect to take coding in-house. This often makes sense for large organizations that have internal programming resources. We've seen that, while programmers are very good at solving technical problems, they usually have neither the training nor the interest in solving the requirements and usability issues that are key to a successful software project. Providing them with a fully-scoped and detailed specification for the project often provides an ideal solution.

Questions?

Clients new to Interface-Driven Architecture often have many questions: Will I really save money? Will this process protect me from the high failure rate of traditional software projects? How will this process work with our existing processes?

Throughout the rest of this *Software Project Survival Guide*, we will explore the objectives and deliverables of each step in an IDA project. If you'd like to discuss further whether IDA makes sense for your software project, please contact us:

Quick Summary

- ✓ Interface-Driven Architecture (IDA) is a two-phase process that uses the user interface to discover and fully specify a software project.
- ✓ IDA produces far better results than traditional software development approaches
- ✓ Steps in the IDA process:

CHAPTER 3

Interface-Driven Architecture Project Plan

Phase I: Architecture

1. Analysis

2. Interface Design

3. Usability Testing

Phase II: Fabrication

4. Software Engineering

5. Coding

6. BETA Testing

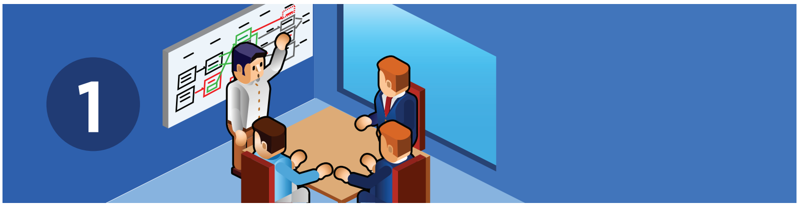
7. Deployment

***.** Monitoring

The client receives two fixed bids

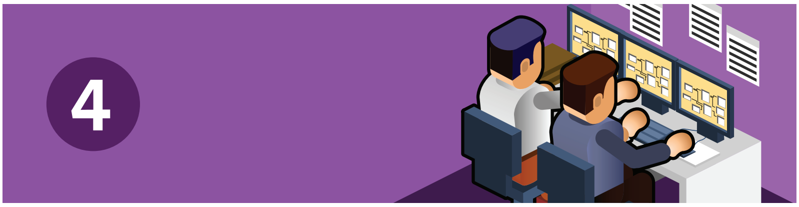
1. an **Architecture** bid, presented at the beginning of the project
2. a **Fabrication** bid, presented after the architecture is complete

- The combined cost of both bids is usually 15-20% less than a single bid based on traditional practice. More importantly, the finished product more accurately reflects the needs of the organization.


1

2

3


4

5

6

7

Phase I: Architecture

1. Analysis

The quality of your finished application depends in large part on the depth of understanding the software development shop has of your business needs. During Analysis, we will often ask to interview those involved in the day-to-day work that the project is concerned with in order to understand your business processes. This provides us with a ground-level view of the software requirements so that the way your company works is reflected in the software we create. During this phase, we will identify various user profiles (often with different needs and goals for the software) and create a shared vocabulary that identifies the various users, deliverables, and processes the software must account for.

Objectives

- ✓ Review existing business process and infrastructure
- ✓ Identify organization/brand position
- ✓ Align goals of the client and the software development firm
- ✓ Identify user profiles
- ✓ Create use cases
- ✓ Develop relational semantic (shared vocabulary)
- ✓ Research supporting technologies

Deliverables

- Project Analysis Report

ARCHITECTURE
✓ Analysis
Interface Design
Usability Testing
FABRICATION
Software Engineering
Coding
Beta Testing
Launch
14%

2. Interface Design

Once Analysis is complete, we begin a process of designing how users will interact with the software. This is far more than look-and-feel this process provides users with a functioning model of the software. Interface Design is the heart of Interface-Driven Architecture (IDA). Interface Design is a highly iterative process that asks users for feedback, provides a new iteration of the interface, and asks users again for feedback. This process continues until (a) users tell us that what we've shown them is accurate and complete, and (b) we are assured that we know everything we need to proceed.

Typical development methodologies keep the user interface hidden from the user until delivery—when it's too late to make necessary changes. The greatest reason (by far) for failed software projects is that the software delivered does not do what users need. By creating the user interface first in conjunction with actual users, we drive out the risk of project failure.

Objectives

- ✓ Define site/application structure
- ✓ Create site/application design
- ✓ Design user interaction

Deliverables

- Completed User Interface software

ARCHITECTURE
✓ Analysis
✓ Interface Design
Usability Testing
FABRICATION
Software Engineering
Coding
Beta Testing
Launch
28%

Phase I: Architecture

3. Usability Testing

Usability Testing begins in the late part of the Interface Design step. Working with focus groups comprised of actual target users, we use special software designed for testing the users' ability to easily and successfully use the system. One source of project failure is user rejection—the software is just too hard or too frustrating to use. With methodical usability testing, we discover in advance any sources of difficulty or frustration and make any adjustments necessary. While usability testing is routine on commercial off-the-shelf software, it is an unfortunate rarity in the custom software development process. A software vendor not doing usability testing should be prepared to explain what they know that companies such as Apple, Google, and Microsoft (all of whom do extensive usability testing) do not.

Objectives

- ✓ Assemble virtual focus groups
- ✓ Test use cases

Deliverables

- Software Usability Report

ARCHITECTURE
✓ Analysis
✓ Interface Design
✓ Usability Testing
FABRICATION
☐ Software Engineering
☐ Coding
☐ Beta Testing
☐ Launch
42%

Phase II: Fabrication

4. Software Engineering

Now, it's time to turn the architectural plan into a fabrication plan. Only a foolish person would begin building a house without a blueprint; in the same way, only a foolish software shop will begin coding a project without modeling the components that will provide working code. The component model provides the foundations on which code will be written. It encompasses software best practices (known as design patterns) expressed in UML diagrams, and a solid relational database schema. It is, in large part, the component model of your project that will determine how robust, scalable, and maintainable the code will be.

Objectives

- ✓ Create domain model
- ✓ Create data schema
- ✓ Create API (Application Programming Interface)
- ✓ Create user documentation/help system

Deliverables

- UML class model diagram
- Entity/Relationship diagram

ARCHITECTURE
✓ Analysis
✓ Interface Design
✓ Usability Testing
FABRICATION
✓ Software Engineering
Coding
Beta Testing
Launch
66%

Phase II: Fabrication

5. Coding

Interface-Driven Architecture makes back-end coding far easier, faster, and less expensive, since (unlike traditional development methodologies) all the business rules and system requirements have been firmly established. While some shops may specialize in a particular programming language, the language and database should be chosen to best fit the type of project you envision. Further, all code should be fully tested to ensure that deployed software will be free of dangerous "bugs".

Objectives

- ✓ Develop system component
- ✓ Perform unit testing
- ✓ Perform coverage testing
- ✓ Perform load testing

Deliverables

- Beta version of application deployed to staging server

ARCHITECTURE
☑ Analysis
☑ Interface Design
☑ Usability Testing
FABRICATION
☑ Software Engineering
☑ Coding
☐ Beta Testing
☐ Launch
80%

Phase II: Fabrication

6. Beta Testing

During the coding phase, developers do their best to build robust code. During the Beta Testing phase, dedicated testers do their best to *break* that code. In too many development shops, beta testing is left as an exercise for the user. But, just as editors help writers produce the best possible product, testers work with coders to ensure that annoying and possibly costly bugs are discovered and "squashed" before the software is launched. At the end of the Beta Testing phase, acceptance testing should be performed. An acceptance test is a formal review in which you agree that the system has been fully and accurately implemented.

Objectives

- ✓ Test all use cases
- ✓ Conduct acceptance testing

Deliverables

- Deployment-Ready Software

ARCHITECTURE
✓ Analysis
✓ Interface Design
✓ Usability Testing
FABRICATION
✓ Software Engineering
✓ Coding
✓ Beta Testing
☐ Launch
94%

Phase II: Fabrication

7. Deployment

When the proceeding phases have been conducted properly, the launch of your project should be a straight-forward process. The Deployment phase simply deploys the fully tested and accepted software into a live environment.

Objectives

- ✓ Prepare server environment
- ✓ Deploy code to production server

Deliverables

- Archival DVD with production code and supporting documentation

ARCHITECTURE
Analysis
Interface Design
Usability Testing
FABRICATION
Software Engineering
Coding
Beta Testing
Launch
100%

*. Monitoring

Once your project is deployed, Monitoring regularly checks system performance to provide metrics on usage and load. A monthly report is provided with any steps needed to ensure the ongoing health of your project.

Objectives

- ✓ Ensure stability and continuing performance of system
- ✓ Report on usage and system load

Deliverables

- Monthly System Health Report

CHAPTER 4

Five Red Flags—Warning Signs of a Project At Risk

Due Diligence Checklist

Here are a few things you'll want to keep in mind when meeting with your next software vendor.

1. Am I meeting with technology experts or just salespeople?

A salesperson might be just fine for purchasing home appliances or vinyl siding, but the complexity of a software project calls for an expert's perspective.

Any company serious about your project's success will have a technology specialist accompany any "sales" people you might meet with. The best companies will send along a technology architect to ensure that things start off on the right foot. Studies have shown that the most common reason software projects fail is that the software built simply does not fit the needs of the client and their users. Experienced development firms understand the risks of the client/programmer disconnect and promote solid lines of communication early.

✓ Make sure you're meeting with a seasoned technology expert capable of providing project guidance.

2. Am I being asked the right questions?

If the right questions aren't answered, a project can be derailed before it gets started.

Serious problems can occur when a software company glosses over important project details during the pre-proposal interview. It's important that developers ask as many questions as possible to form a clear picture of how and why the project at hand will add value to your organization. Experienced development firms will ask questions centered on your business model, company background, current technology infrastructure, work-flow, business processes, previous technology experiences, etc. One sign that you're not dealing with a technology architect is that the sales person will ask few probing questions. Asking questions early will ensure that the mission is clear and the goals well-defined.

✓ Make sure the consultant is asking enough questions to understand your business needs.

3. Am I being presented with a comprehensive project plan?

Without a proven methodology driving your project, there's just no telling where you'll end up.

An experienced development firm presents a comprehensive project plan or methodology before offering a proposal. These plans usually include a set of steps or phases intended to guide your project from start to finish and ensure a positive result. Many projects fail due to lack of planning, which can result in costly timeline and budgetary overages.

✓ Make sure the consultant is fully educating you on their methodology and plan for success.

4. Will I receive a fixed bid before important questions are answered?

"Back of the envelope bids", as they are known, only lead to broken timelines and change orders.

This single issue is responsible for more project failure than any other. The practice of providing a fixed cost bid before those responsible for developing the project have fully realized its requirements can seriously threaten a project's success. Until every last detail of the software's form and function has been established and reviewed by the developers, project stakeholders, and most importantly, the eventual users of the product, any calculations as to the price or timeline of a project is little more than guess-work.

✓ Make sure the consultant isn't pricing your project with a "back of the envelope bid".

5. Am I being advised on specific project risks?

When it comes to managing business risk, leaps of faith are usually made off a cliff.

The truth: 71% of all software projects fail! Any company that doesn't address the problems of software failure head-on is asking for trouble. Each project represents a unique set of challenges that must be addressed and planned for.

✓ Experienced firms understand the risk involved with software development and use best practices, methodology, and clear client communication to beat the odds.

CHAPTER 5

Consultant Surveys

Consultant Survey

Use this short survey to rate your experience with our firm.

You may also choose to re-use this form to gauge your experience with the other consultants you'll be meeting with.

1 The consultant asked detailed questions about my business model and processes.

1	2	3	4	5	6	7	8	9	10

2 The consultant asked detailed questions about my project's mission.

1	2	3	4	5	6	7	8	9	10

3 The consultant educated me on his/her development process and methodology.

1	2	3	4	5	6	7	8	9	10

4 The consultant offered valuable ideas and feedback on my project.

1	2	3	4	5	6	7	8	9	10

5 The consultant identified the specific risks and challenges involved with my project and discussed how they would be addressed.

1	2	3	4	5	6	7	8	9	10

6 The consultant related past experience working with projects similar to my own.

1	2	3	4	5	6	7	8	9	10

7 The consultant listened to my needs and demonstrated understanding of my business and goals.

1	2	3	4	5	6	7	8	9	10

Key:

A: 64-70 **B**: 57-63 **C**: 50-56 **F**: 0 - 49

Your Notes

Consultant Survey

Use this short survey to rate your experience with our firm.

You may also choose to re-use this form to gauge your experience with the other consultants you'll be meeting with.

1 The consultant asked detailed questions about my business model and processes.

1	2	3	4	5	6	7	8	9	10

2 The consultant asked detailed questions about my project's mission.

1	2	3	4	5	6	7	8	9	10

3 The consultant educated me on his/her development process and methodology.

1	2	3	4	5	6	7	8	9	10

4 The consultant offered valuable ideas and feedback on my project.

1	2	3	4	5	6	7	8	9	10

5 The consultant identified the specific risks and challenges involved with my project and discussed how they would be addressed.

1	2	3	4	5	6	7	8	9	10

6 The consultant related past experience working with projects similar to my own.

1	2	3	4	5	6	7	8	9	10

7 The consultant listened to my needs and demonstrated understanding of my business and goals.

1	2	3	4	5	6	7	8	9	10

Key:

A: 64-70 **B**: 57-63 **C**: 50-56 **F**: 0 - 49

Your Notes

Consultant Survey

Use this short survey to rate your experience with our firm.

You may also choose to re-use this form to gauge your experience with the other consultants you'll be meeting with.

1 The consultant asked detailed questions about my business model and processes.

1	2	3	4	5	6	7	8	9	10

2 The consultant asked detailed questions about my project's mission.

1	2	3	4	5	6	7	8	9	10

3 The consultant educated me on his/her development process and methodology.

1	2	3	4	5	6	7	8	9	10

4 The consultant offered valuable ideas and feedback on my project.

1	2	3	4	5	6	7	8	9	10

5 The consultant identified the specific risks and challenges involved with my project and discussed how they would be addressed.

1	2	3	4	5	6	7	8	9	10

6 The consultant related past experience working with projects similar to my own.

1	2	3	4	5	6	7	8	9	10

7 The consultant listened to my needs and demonstrated understanding of my business and goals.

1	2	3	4	5	6	7	8	9	10

Key:

A: 64-70 **B**: 57-63 **C**: 50-56 **F**: 0 - 49

Your Notes

Consultant Survey

Use this short survey to rate your experience with our firm.

You may also choose to re-use this form to gauge your experience with the other consultants you'll be meeting with.

1 The consultant asked detailed questions about my business model and processes.

1	2	3	4	5	6	7	8	9	10

2 The consultant asked detailed questions about my project's mission.

1	2	3	4	5	6	7	8	9	10

3 The consultant educated me on his/her development process and methodology.

1	2	3	4	5	6	7	8	9	10

4 The consultant offered valuable ideas and feedback on my project.

1	2	3	4	5	6	7	8	9	10

5 The consultant identified the specific risks and challenges involved with my project and discussed how they would be addressed.

1	2	3	4	5	6	7	8	9	10

6 The consultant related past experience working with projects similar to my own.

1	2	3	4	5	6	7	8	9	10

7 The consultant listened to my needs and demonstrated understanding of my business and goals.

1	2	3	4	5	6	7	8	9	10

Key:

A: 64-70 **B**: 57-63 **C**: 50-56 **F**: 0 - 49

Your Notes