To prevent specifications or stakeholder expectations from spinning out of control, consider this simple, business-based view of requirements.

Fergal McGovern

Managing Software Projects with Business-Based Requirements

If you're a project manager who wants to drive business benefit from existing or new systems, it's important to understand what business processes your system must address. You need to know exactly where to change a system and, as you implement change, how to measure its completeness. To successfully address these challenges, project managers must trade traditional subsystem project planning for objective-based, requirements-driven project management. This approach differs from traditional development processes.

For many organizations that are neither software product companies nor system integrators, the expense and cultural change required for full-process rollout can be prohibitive. Proponents of agile processes/methods (such as Extreme Programming) suggest that these "lightweight" approaches are extremely effective. I would agree that there are many powerful aspects within these approaches. I suggest, however, that by taking an objective-based business requirements approach to project management, software projects have a high probability of running on time, and remaining in scope and within budget. Addressing requirement challenges, independent of adopting a full process, can offer many of the benefits of full process adoption while avoiding most of the expense and human issues involved with full-process rollout. A business-based requirements approach is an easy-to-adopt, risk-free entry point that offers tangible quality improvements.

This approach suits any project scope. Whether building a complex system for enterprise resource planning or customer relationship management, or developing small, single-user software programs, defining business requirements improves any system delivery.

Inside

SUBSYSTEM PROJECT PLANNING

According to a Standish Group study, nearly half of all application development projects cost 70 percent more than originally budgeted (John Berry, "Bringing Some Clear Thinking to IT Projects," *InternetWeek*, 6 Sept. 1999). The managers who participated in the study cited a lack of user input as the main reason for project failure. So, if project managers say they need requirements, and most reasonable project managers understand these needs, why do so many projects fail?

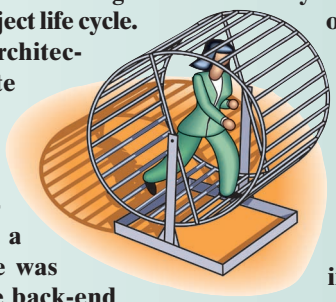
These failures boil down to a very basic premise: The way many managers structure project plans is not around business requirements or measurable goals, but around passive and unmeasurable module definitions.

Consider a typical project plan as encountered in many software environments. For example, let's say you're developing software to manage a hotel reservation system. You might begin by defining deliverables that are essentially subsystems, such

1520-9202/02/$17.00 © 2002 IEEE

Problems with Subsystem-Based Planning

- **It becomes difficult to coordinate the dependencies between each subsystem because the integration happens toward the end of the project life cycle. This model exposes logical or architectural flaws and inconsistencies late in the process. A classic example of this situation is inconsistent formatting in the messaging interchange among disparate subsystems. I've even encountered a system where the user interface was unable to communicate with the back-end subsystem because developers did not coordinate subsystem dependencies.**
- **Customers won't receive deliverables (aside from prototypes) until developers fully complete the system, running the risk that the system they're building is not what the customer wants. An example of this is interactive "cardboard" Web interfaces, which often bear no resemblance to what is technically feasible.**
- **Because customers want prototypes to demonstrate progress, the project can descend into a prototyping spiral where the real system never gets built. This can happen when a customer believes that a smoke-and-mirrors prototype represents real progress on the actual system. In reality, little effort has gone into building the system. This situation can also arise when the project manager can't say "no" and keeps altering the prototype.**
- **Customers can't review the business functionality in a meaningful way. Because developers integrate subsystems late in the cycle, customers don't have the opportunity to review the system's functionality until after integration. At this point, engineers have already "poured the concrete" of the system—if it needs to be torn up and rebuilt, costs escalate, and the project misses deadlines.**
- **Early verification of the overall architecture is difficult because testing for subsystem interactions occurs late in the planned life cycle. This delay increases the likelihood that developers will encounter oversights requiring significant redesign late in the life cycle, resulting in slipped schedules and increased cost.**
- **Complex integration issues arise toward the project's end, extending its completion time and increasing costs. Developers loose a substantial amount of work if the system doesn't perform the way a customer expects.**
- **It becomes difficult to plan for incremental development. A subsystems-based project is essentially one large increment, first building the discrete subsystems and then putting them together. Although well-crafted project plans (implying incremental development) can conceal the shortcomings of this view, it can burn stakeholders during the final delivery cycles. Looking at a system as just one large increment is identical to the waterfall approach and suffers from some of the same pitfalls.**



as staff management, customer management, business logic, and user interface screens.

The one main problem with this type of subsystem project planning is the emphasis on system components as deliverables rather than viewing deliverables from an objective-based perspective. The "Problems with Subsystem-Based Planning" sidebar lists other pitfalls.

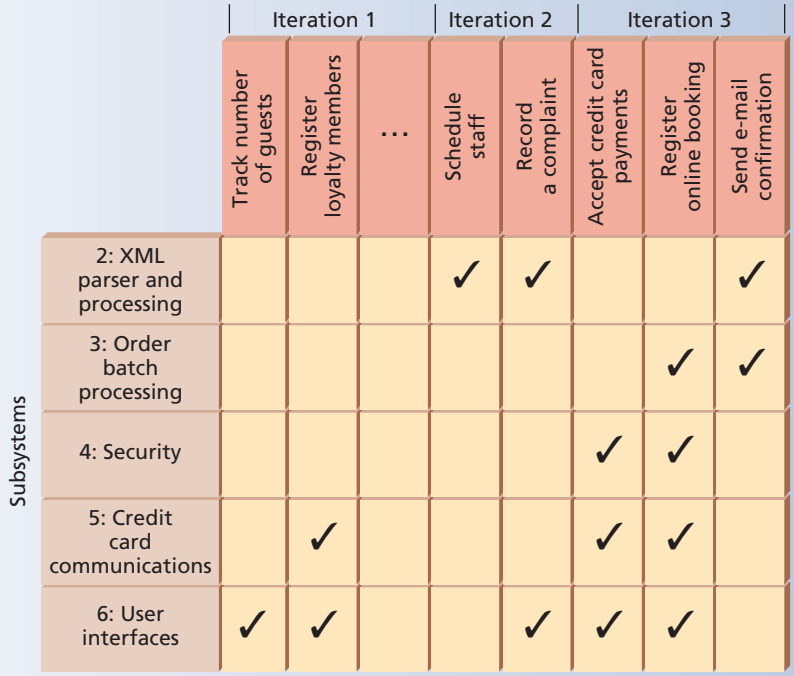
An objective-based perspective might include "track the number of guests visiting the hotel" or "register new loyalty members." In the subsystem scenario, project managers have no real way of measuring completion or progress. How can they measure the completion of business logic? Or how can they effectively judge whether a subsystem meets customer expectations?

OBJECTIVE-BASED PERSPECTIVE

Taking an objective-based perspective still requires project managers to manage the types of requirements that apply across subsystems. These requirements are called *nonfunctional* and break down into qualities and constraints. *Qualities* are characteristics that a system must have, but they are not objective based. Such qualities relate to branding, security, scalability, performance, and so on. *Constraints* are environmental considerations that pertain to the system's development or deployment. For example, a system might have to exchange formats with other systems (possibly across different enterprises). A corporate entity might insist on a certain development language or environment, or mandate a specific platform.

The question remains, where best should managers record these types of system (nonfunctional) requirements? In keeping with an objective-based view of requirements, it is appropriate to associate these qualities and constraints with either a specific business objective (or set of business objectives) or with the entire system. For example, you can combine a requirement concerning a system's

Figure 1. Objective-based business requirements interact with subsystem requirements.



user interface with a business objective such as "register new loyalty members." This means that objective-based requirements still provide a testable and measurable framework from the project management side. Project managers can analyze nonfunctional requirements that are testable, and verify those that are not while delivering increments of the system.

In Figure 1, the orange blocks indicate the (functional) objective-based business requirements. The brown blocks represent (nonfunctional) subsystems, some of which are involved in more than one business objective. Nonfunctional requirements, such as security, apply directly to the subsystems on the left, and these are measurable. However, certain nonfunctional requirements, such as branding, cannot be tested; project managers must associate them with relevant business objective(s).

WHY REQUIREMENTS?

Software is difficult. Unlike the situation in many human systems, software project managers can't actually *see* the software they produce. Consequently, it's hard to measure and manage its production. As a project manager, you need to see what your team is producing to coordinate and orchestrate the different elements that go toward a system's production or modification.

Many project managers would agree that orchestration is their primary job. This task amounts to managing resources effectively to meet the requirements of specific tasks or objectives. Without documented and clearly articulated *business* requirements, effective orchestration is difficult—if not impossible—to achieve, even for the best project managers, as the "10 Pitfalls of Developing Software without Requirements" sidebar explains. Although many project managers would argue that time and cost constraints are the most critical obstacles to effective project management, the key problem lies in not having a clear understanding of the project's business objectives, followed by not managing project delivery by business objectives, or some combination of these problems.

For example, a project manager coordinating the development of a hotel reservation system has a better understanding of the project and a better chance of making accurate timing and resourcing estimates if she tracks these against objective-based requirements. Fulfilling requirements—such as "check-in guest" or "issue loyalty discount"—requires resources and costs money. By accessing the resources necessary against the business requirement, project managers can more realistically calculate the resources and time needed to complete a project. In this case, you might start by allocating six weeks to a reasonably sized business objective such as "issue loyalty discount." This estimate, of course, would subsequently undergo refinement based on a first iteration of development, which helps refine and validate initial estimates.

In contrast, making concrete time and resource estimates based on subsystems is much harder because subsystems are more variable; their functionality tends to be less concise (in terms of communicable definition) early in the project life cycle. This means subsystems tend to have clear design and code that evolved later in the project; they rarely have external-facing (English-based) boundaries specified early in the life cycle. Thus, considering subsystems as the primary vehicle for estimation is prone to variability and so increases project risk.

REQUIREMENTS-DRIVEN PROJECT PLANNING

Rather than focusing on subsystem deliverables, requirements-based project planning focuses on real, measurable business requirements. Several key principles drive suc-

cessful project planning based on business requirements.

Specify objectives clearly

First, you must structure requirements in terms of objectives. The requirements must be measurable and understandable, and express a goal or objective.

For example, a typical project objective is "build user interface," but this objective does not describe what the user interface will accomplish or the entities shown in the user interface at a given point in the flow. A better objective would be "register guests" which has a specific step called "display room data," which includes the following:

- total number of rooms,
- number of rooms available for VIP guests,
- number of over-allocated rooms, and
- room rates per room type (either corporate suite, double, or single).

Also strive to express specifications in a structured way that is accessible (understandable) to all stakeholders. Software specifications are best expressed in structured English defined as specific steps, or groups of steps within a process. For example, "build XML parsing infrastructure" might be understandable to your developers, but not to your customers, whereas "exchange purchase data with suppliers" makes sense to customers. This particular requirement is what the XML-parsing subsystem fulfills.

Plan for incremental delivery

Project planning should center on incremental delivery. Structure the project deliverables in terms of iterations, each iteration corresponding to a group of functional requirements. These requirements should consist of collections of use cases (a common way of representing objective-based requirements) that let you accurately identify issues early on and take appropriate action.

A sensible delivery increment per iteration is approximately four weeks, depending on the system's overall complexity. It should not extend beyond six weeks. Even in very large system developments (with many developers spanning durations greater than six months) it is always a good idea to partition the overall project into subprojects, each of which consists of several iterations. As a rule of thumb, each project should contain three iterations. It is a good idea to have micro-increments within an iteration. See *Surviving Object-Oriented Projects*, chapter 5, "Increments and Iterations" (Alistair Cockburn, Addison-Wesley, 1998) for more details on this approach.

Identify steel threads

Out of these iterations, you must next identify the steel threads. A steel thread is a set of logically grouped objectives that does not consider any alternate or exceptional scenarios. The steel thread includes that portion of the system that satisfies these scenarios. Identification of the business objectives that comprise a steel thread considers two factors: the technical risk associated with these scenarios

10 Pitfalls of Developing Software without Requirements

If you don't document the requirements for your software system, you risk several problems.

- **Communicating what the system will do for the customer becomes very difficult.**
- **The team has a tendency to exclusively rely on prototypes to communicate requirements to the customer. This situation can lead to prototyping spirals in which you never really build the actual system.**
- **Project estimation becomes difficult because there is no concrete notion of what the customer required and no effective way—beyond a gut-feel approach—to measure the effort.**
- **Incremental delivery of the solution becomes impossible.**
- **Your company has an over-reliance on having the right technical people available for every project, a situation that is neither repeatable nor scalable.**
- **Communication and coordination between midsize teams become difficult, as does the production of reliable and accurate project plans and design.**
- **Planning is ad-hoc at best, because the team doesn't document metrics such as how many requirements are outstanding, in progress, or already built.**
- **Code reuse is impossible because the team lacks a clear way to identify reusable artifacts, whether these are designs, project plans, or sets of generic system requirements.**
- **Without documented requirements, software production is not predictable or repeatable. This results in resourcing issues that prevent you from reliably predicting the project's duration or complexity.**
- **Managing customer expectation is difficult without requirements. If business analysts and project managers fail to manage expectations, they lose credibility with the customer.**

Figure 2. Requirements-driven project template.

ID	Task Name	Duration	Start
1	Project set up	27 days	Mon 04/09/00
25	Engagement/Business Analysis	14 days	Mon 04/09/00
26	Engagement	2 days	Mon 04/09/00
29	Business Analysis	14 days	Mon 04/09/00
41	Systems Analysis	20 days	Mon 04/09/00
42	Systems Analysis	2 days	Mon 25/09/00
45	UI Design	2 days	Mon 04/09/00
50	Architectural Design	4 days	Tue 26/09/00
54	Project Management	16 days	Mon 04/09/00
60	Design/Construction	32 days	Mon 28/08/00
61	Production Env. Planning	19 days	Mon 04/09/00
67	Functional Group 1 (<use case name(s)>)	12 days	Mon 28/08/00
68	Steel Thread (1st Pass)	12 days	Mon 28/08/00
83	Full Build (2nd Pass)	9 days	Tue 29/08/00
84	Update design	5 days	Tue 29/08/00
91	implementation	2.5 days	Tue 05/09/00
94	Test	1.5 days	Thu 07/09/00
96	Shared Package 1	1.5 days	Mon 04/09/00
104	Release Functional Group 1	9 days	Mon 04/09/00
105	Release	1 day	Mon 04/09/00
116	Customer Release	9 days	Mon 04/09/00
122	Functional Group 2 (<use case name(s)>)	10 days	Wed 30/08/00
160	Release Functional Group 2	1 day	Mon 04/09/00
165	System Test	6 days	Tue 03/10/00
169	Go Live	9 days	Mon 04/09/00
170	Release	1 day	Mon 04/09/00
182	Customer Release	9 days	Mon 04/09/00
189	UAT	5 days	Mon 04/09/00
192	Post Go Live	2 days	Mon 04/09/00

and the areas of importance to the customer.

Identifying these threads reduces risk. Developers build and verify steel threads of functionality early in the project's construction life cycle, permitting early modification of the architecture or design, if necessary. This steel-threading approach builds what is a main success scenario for a significant objective-based requirement or set of requirements.

Once developers complete a first steel thread, they can wrap additional threads around the first, each thread representing one or more alternate scenarios in the requirement. By choosing initial steel threads that prove the technical architecture and design, you can discover any technical issues early in the project life cycle and build in additional contingencies into the project plan. In a sense, steel threads are sub-increments within a given iteration and typically last about two weeks.

Trace design work back to requirements

It's also important to trace the design and build from the requirements. Always ensure that you can trace any piece of design work to one or more requirements. What's important is to have business analysts on the team lead the definition of business-based requirements. They should then work with the technical staff to map those requirements into appropriate components or system interfaces, identifying suitable business objectives within a given iteration as well as suitable steel threads, such as those shown in Figure 1.

If developers are building something that isn't traceable, it indicates one of two problems:

- you might have potential issues with the accuracy of the requirements, or
- the team is building something outside of the system's scope.

Develop test specifications based on requirements

A classic symptom of a project in trouble is lack of coherent test specifications. Testing is not a glamorous activity for many developers; they often consider it outside the scope of their work. As such, developers often overlook testing. However, lack of testing is one major factor in failed or troubled projects. Always ensure that every path through the system has an associated test case. Because the objective-based requirements form the basis for traceability into the design, these requirements also form natural test cases. Each scenario in an objective-based requirement equals one test case.

Manage Web or GUI design from requirements

Most systems have a visual interface that lets users interact with the system. Ensuring that developers design the user interface to serve requirements is critical to project success. Too often, especially in recent Web-based projects, a separation of teams causes a lack of visibility into the requirements. This situation results in integration problems late in the project life cycle.

This very problem occurred on a project when a graphic designer used a frame-based approach to build a user interface that explicitly disregarded security requirements. This

discovery was made very late in the delivery cycles, resulting in an extensive Web redesign.

Manage customer deliveries by requirement set

Ensure that the customer expects delivery of portions of the system at a point that coincides with a finished iteration. If the customer is anxious to review progress, let her have real, working pieces of steel-threaded functionality. One of the key dangers with customer review cycles is offering an initial prototype that the customer perceives to be an actual system. The customer might want to use the prototype now, although it is, in fact, still very much a throwaway.

Avoid this situation by delivering steel threads instead of prototypes. Doing so ensures that the focus remains on ultimate system delivery rather than on endless prototyping, in which you never build the real system.

ADVANTAGES OF REQUIREMENTS-BASED PROJECT PLANNING

Requirements-based project planning lets you control a project's scope and have visibility into your software system. Figure 2 shows a project template you can use to enforce requirements-driven project management.

In the template, the project's design-construction phase is oriented around iteration deliveries. In this example, I refer to each iteration as a *functional group*. The functional group consists of sets of use cases (business objectives), each of which in turn has a steel thread and a *full build*, which wraps the alternate and exceptional scenarios around the core steel thread. The key fact is that this template expresses each iteration as a collection of business-based objectives. The portion of the subsystem required to fulfill each use case represents a distinct task in the project plan and a subtask of the associated use case.

In the hotel management software scenario, the project manager reduces project risk by setting clear expectations as to what elements the system must produce. For instance, let's say the system must register new members and track the number of member visits. If the software does not follow through on those requirements, developers must make changes during that iteration of the build.

Because this method expresses technical requirements in business language ("issue an invoice," for example), it becomes easier to share knowledge with nontechnical stakeholders, reducing the possibility that mismatches will occur between the requirements and the system. The key benefit is in knowing that the system you build will be what your customer expects. The quality of the hotel reservation system improves because the project manager knows the system must process applications and credit card transactions, invoice suppliers, and issue loyalty membership cards. The system's success does not depend on subsystems like staff management that are difficult to test and to deliver satisfactorily.

With no requirements, projects invariably descend into chaos. The symptoms of this are stress, long hours, jarred morale, and general dissatisfaction among project participants. Next stop: dissatisfied customers, disgruntled employees updating their resumes, and business stakeholders who doubt your company's credibility. Given these risks, business-driven requirements seem like a simple yet effective way to avoid a lot of heartburn. ■

Fergal McGovern *is founder and chief technical officer of SteelTrace, a software development firm based in Dublin, Ireland. SteelTrace develops business process and requirement tools that increase the accuracy between customer requirements and system delivery. Contact him at fergal.mcgovern@steeltrace.com.*