9 Web Project Management

Herwig Mayr

Many Web applications are created by companies that either have been active in the software industry only briefly or are rooted in traditional thinking and development strategies. While the former (can) demonstrate little to no management competencies, the latter try to use process models from other software development areas or transpose proven practices to the domain of Web applications, which makes them fail just as often as inexperienced newcomers.

What makes Web project management different from traditional software project management, and what traps does it hide? As far as potentially successful methods and approaches for Web project management have already evolved, they will be presented briefly and embedded in the holistic perspective required in this field.

Project management is a human activity to shape the actions of other humans. This human-centered perspective requires Web project managers to have enormous conflict-solving competency, and Web teams to have an interdisciplinary understanding. Consequently, the model used to develop Web applications has to be very flexible, allowing for a strongly iterative-incremental development, and involving the contractor frequently. This means that tools and techniques used in Web project management are particularly characterized by the current transition from traditional software development methods towards agile methods. Consistent use of integrated tools is just as essential as consequent risk management during the entire project cycle.

If one looks at the success rate of software projects during the past ten years, one can see that it has constantly remained at a low level. In view of the explosive emergence of technologies and rapidly increasing task complexity, even meeting this rate for Web projects during the next few years may be considered a success.

9.1 From Software Project Management to Web Project Management

9.1.1 Objectives of Software Project Management

Software project management[1] supports an engineering approach to software development in that it extends the technical product development cycle (planning – implementing – checking)

1 The author prefers the term "project engineering" over "project management", because the term "management" is often reduced to pure planning or controlling functions. However, we will use "project management" in this chapter because this term is more commonly used.

to economic and social tasks, like managing, developing, and monitoring. This turns software development into an iterative, *controlled process*, allowing a well-understood and continuous adaptation to the objectives (see Figure 9-1). Software project management thus ties the technical product development to economic product manufacturing.

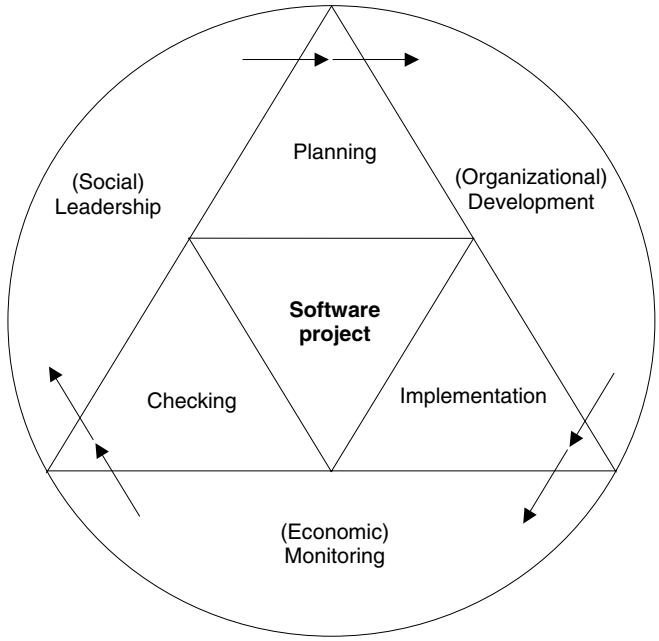


Figure 9-1 Project management objective: an engineering approach to software development.

9.1.2 The Tasks of Software Project Management

A *project* is an innovative and complex undertaking with conditions, such as costs, deadlines, resources, and quality. A company's performance process has to be coordinated by (*project*) *management* so that the general conditions/restrictions can be maintained. More specifically, "the management has to specify the objectives and strategies for the company, operationalize them in plans, monitor the achievement of the objectives, develop an adequate corporate organization for implementation of the objectives and plans, lead and motivate the staff, control the corporate processes, and take decisions. . . . This means that management can be defined as an activity that deals with shaping the actions of other people" (Gernert and Ahrend 2001). This definition results in the following tasks for (software project) management (according to Gernert and Ahrend 2001, and structured as in Figure 9-1):

- *Leadership*: Organize, control, lead staff, inform.
- *Development*: Set, plan, and define objectives.
- *Monitoring*: Check and control.

9.1.3 Conflicting Areas in Projects

From an economic point of view, a project is often seen as a system that has to be well balanced between the available *budget*, the fixed *time* horizon, and the projected product *quality* (see Figure 9-2). The important aspect about this point of view is that none of the three parameters can be changed without entailing a change to one or both of the other parameter values. A project that has to be completed within the shortest possible time becomes more expensive than originally planned, or the quality drops. In practice, both will occur in most cases.

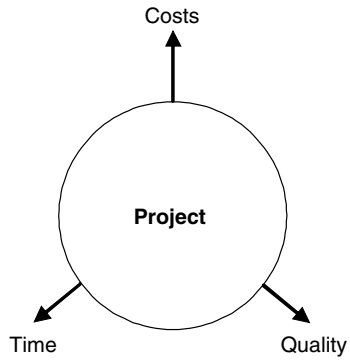


Figure 9-2 The traditional conflicting areas in projects.

It is important to make the customer aware of these "areas of conflict" in a project from the very beginning, and stress the impact of changing deadlines, cutting costs, etc. This cannot be done emphatically enough. In fact, especially for Web projects, which frequently have to be handled under tight budgets and even tighter deadlines, the "simple" relation between budget, time, and quality is often lost in the development hustle.

9.1.4 Specifics of Web Project Management

It can generally be observed that many large and monolithic applications developed in the past have been replaced by a large number of (very) small and networked Web applications (Reifer 2002). This trend entails shorter development cycles, leading to situations where software is increasingly less developed in the traditional way – based on specified requirements – from scratch. Instead, components are coupled in an agile approach (see Chapter 10), and *refactoring* is used to develop a meaningful design on the job. Table 9-1 shows the characteristics resulting for *Web project management*, compared traditional software project management (adapted from Reifer 2002).

Many young developers are not familiar with traditional models and methods that ensure and increase development maturity (such as CMMI or ISO 15504) and time to learn and apply these models is frequently not available. Process development, discipline, or estimation skills are typically shed as unnecessary ballast.

Table 9-1 Traditional software project management versus Web project management

Parameter	Software Project Management	Web Project Management
Main objective	Create a quality product at lowest possible cost!	Create a usable product in shortest possible time!
Project size	Medium to large (10 to 100 people and more)	Usually small (6 +/− 3 people)
Duration	12 to 18 months on average	3 to 6 months on average
Cost	several million dollars	several thousand dollars
Development approach	Based on requirements; structured into phases; incremental; documentation-driven	Agile methods; assembly of components; prototyping
Technologies	OO methods, CASE tools	Components-based methods; visual programming; multimedia
Processes	CMM, ISO, etc. ("rigid")	ad-hoc ("agile")
Product	Code-based; poor reusability; complex applications	High reusability; standard components; many standard applications
Staff profile	Professional software developers with several years of experience	Multimedia designers; Web programmers (Java, etc.); PR/marketing people

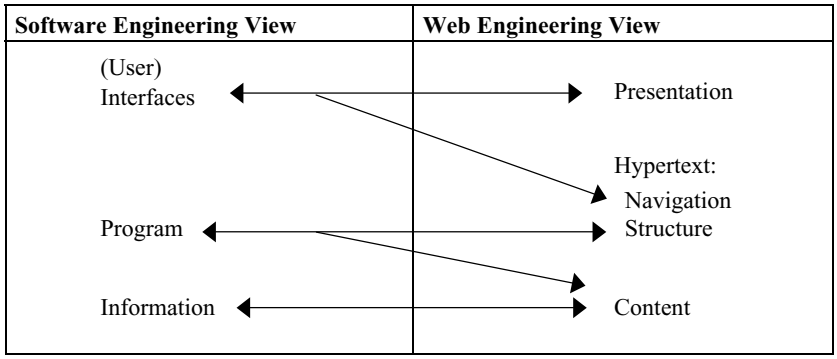
Web projects also differ from traditional software projects in their results:

- Traditional software systems are comprised of parts grouped by functions, where the key metric of these parts is functionality. In contrast, software functionality and content depend on each other in Web applications, and the joint availability of both elements is essential from the very first delivery on.[2]
- The design and the creation of the content are at least as important as the application's functionality. For web applications, the structuring into design components is done in different ways by the different development communities, using different naming conventions (see Table 9-2).

As mentioned in the literature (e.g., Chan 2002), these areas have to be coordinated and – ideally – developed jointly. While *information design* aims at the content, *interface design* deals with user interaction and navigation in the Web application. *Program design* comprises the functionality and communication with the application in the backend (databases, data warehousing systems, etc.). The main objective of Web project management is to optimally match the presentation of information, access, and functionality of a Web application, and coordinate all these areas with the content from the product perspective.

2 A more extensive (object-oriented) perspective considers functionality to be part of the content (see also Chapter 3).

Table 9-2 Design components of Web applications



9.2 Challenges in Web Project Management

9.2.1 General Challenges in Software Development

Conventional project management in traditional software projects is confronted with challenges in all three management tasks (see section 9.1.1). These challenges also apply to the development of Web applications, as described in the following subsections.

Leadership Challenges

- *Unique software systems*: Software systems are frequently developed from scratch. The experience drawn from past projects is too little to be able to make reliable cost estimates. Web project management counters these challenges by a much higher degree of reusability and reuse.
- *Extremely technical leadership perspective*: Project management has been dominated by technology freaks, particularly technology-minded engineers. In many cases, this has led to a neglect of organizational development in favor of software development. In addition, engineers tend to plan overly optimistically. This attitude is often "benevolently" supported by marketing and sales people. Web project teams are much more heterogeneous and less geekish. However, this doesn't necessarily mean that they are more experienced in project management, and it can cause other problems within the group (see also section 9.3.2).
- *Poor planning*: Many software products are characterized by unclear or incomplete planning objectives, frequent changes to the planning objectives, and defects in the project organization. Compared with traditional software development, these problems arise even more frequently in the development of Web applications, as we will discuss in the next section.

Development Challenges

- *Individuality of programmers*: Even today, many software development projects are seen as an art rather than a technique. Software programmers are individualists and their

performance differs a lot. This is the reason why it is particularly difficult to estimate the actual manpower needed. Moreover, it is difficult to put individualists into an organizational straight-jacket. This problem arises especially due to "artists" in Web teams, because their creativity is subject to a high degree of individuality.

- *High number of alternative solutions*: In software development, there is virtually an unlimited number of alternatives to solve a specific problem. In many cases, it is impossible to compare and evaluate different solutions in advance. This problem is slightly smaller in the creation of Web applications, because many components and semi-finished products can be used, but it shouldn't be underestimated.
- *Rapid technological change*: The rapid technological development of hardware and software makes it more difficult to plan and organize software projects. It often happens that, while a large software system development is under way, new and better performing components (e.g., enhanced graphics functionalities) enter the market. This means that novelties introduced to the market while a project is in the works can make the system conception appear outdated and require a change to the design, making the plan obsolete. On the other hand, it can mean that new software tools become available, and their benefits are hard to tell. This problem is typical for Web projects.

Monitoring Challenges

- *The immaterial state of software products*: The "intangibility" of software products makes them hard to control. It is very difficult to determine how much of a software product is actually completed, and the programmer has a wide range of possibilities to veil the actual development state. Since Web projects are characterized by parallel development of functionality and content, the product is more "tangible" for customers and the project manager. And since Web projects are subject to short iteration cycles, they are usually easier to check. For these reasons, this challenge is of less significance in Web projects.

In addition to these challenges that have to be dealt with in any kind of software development, the environment and restrictions in the development of Web applications lead to particular difficulties and challenges. These difficulties and challenges were briefly mentioned in section 1.3 to form a basis for further discussion in various sections in this book. The challenges posed to Web project management in dealing with these characteristics will be discussed in the following section.

9.2.2 Development-related Challenges in Web Projects

The following properties are typical for the development of Web projects, representing special challenges for Web project management.

Novelty

Many Web applications are designed to address a new or unknown user group (see also Chapter 2). Naturally, if these future users don't know what is offered or what they can or should

expect from a new Web application, they cannot formulate requirements or express their expectations (see Rahardjam 1999). For this reason, Web application developers are confronted with new or changing requirements in a Web project much more often than developers of traditional software systems.

But even for known user groups conflicts can arise quickly. For example, users of a Web application might be interested in links pointing to competitive products while reading a company's product information, which is usually not in the interest of that company. For the Web project manager, this means a lot of prudence and intuition in weighing interests in an extremely insecure requirements portfolio for a Web application. Also since most Web applications are new, it is difficult to fall back on past experience. Section 2.5 deals with solutions in this respect.

Dynamics

Many Web application development projects are characterized by high time-to-market pressure and short development cycles. For example, (McDonald and Welland 2001a) observed in a study that no Web project lasted more than six months, the average being less than three months!

The reason for this high time pressure in Web projects is seen in the rapid change on the Web and the consequent short lifecycles of Web applications, or their high update frequency. In addition, there is currently fierce competition in the market of Web application development, since this is an expansion *and* ousting market. The literature (e.g., Holck and Clemmensen 2002, Pressman 1998) has more recently been arguing that the necessity for quick development, high quality, and maintainable products/technologies is not limited to Web applications. These properties characterize a general trend in software development, which is dealt with by the use of agile methods, component systems, and an increasing concentration and integration of development tools (IBM with Rational, Borland/Inprise with Togethersoft, etc.).

The following solution strategies are suitable for Web project management. The contents of many Web applications can be divided into a number of smaller subprojects, which can be easily handled by subproject groups (see also section 9.3.2). Due to the short development time, (Pressman 2000b) suggests that both the development plan and the design should have a fine granularity. This allows the developers to break the project tasks down into daily work packages and (mini) milestones.

Parallelism

Due to the short development cycle and the component-based structure of the application domain (e.g., authentication, similarity search, news ticker, chat room) often found in Web applications, many Web applications are developed by subgroups in parallel. However, these subgroups are structured differently than in traditional software projects. While the latter are generally structured in a development-oriented way (e.g., division into GUI development, database connection, mathematical/algorithmic modeling), which means that the composition of these groups varies with regard to the expertise of their members, subteams in Web projects have to exchange more than just information about these sorts of component interfaces. Many also develop components with similar functionality and design structure, but with different customer significance (e.g., product list and customer reference list) (McDonald and Welland 2001a).

Web project management is responsible for ensuring that experts with similar experiences or competencies (e.g., interaction designers, graphics artists) working in different groups

communicate across group boundaries. This ensures consistency of the Web application as a whole and prevents double-tracked development.[3] This type of communication is poorly mapped by traditional software development tools and CASE tools (Holck and Clemmensen 2002).

Continuity

The objectives of Web applications and the tools to create and use them as well as the Web itself are subject to continuous evolution. Consequently, the transition from development to maintenance is hard to schedule for Web applications, and it generally doesn't make sense. An integral look at the application and its content increases this characteristic even more. To manage this continuous evolution, (Pressman 2000b) suggests defining an incremental process model for Web Engineering, which ". . . allows the development team to freeze the focus of an increment so that a workable release version of a Web application can be generated. The next increment will perhaps address focus changes, which had been suggested in a review in the previous increment. But as soon as the second increment has been defined, the focus will again be frozen." This technique is nothing else but the usual release policy in traditional software projects, except that the time between two releases is often just a few weeks or even days, rather than several months.

Maintenance is difficult particularly if a Web application has to offer 24×7 availability. In this respect, documenting the requirements and design decisions (including their reasons!) in writing despite short deadlines is extremely important to ensure that development paths and the change history can be reproduced and understood. The use of a configuration management tool is a must in this respect (see section 9.4.1).

Juvenility

Compared with the average software developer Web application developers are on average much younger and less experienced. Based on surveys conducted by (Reifer 2002), the average developer of Web applications has less than three years of experience and typically is self-taught while the average software engineer has a formal education and at least six years professional experience.

Due to their youthful enthusiasm, many Web application developers are very interested in new tools and technologies, while being ignorant of existing knowledge and best practices. They like to try new tools and software systems, and updating to the most recent versions is a must for them. This attitude often leads to a situation where especially Web startups use a large number of tools and technologies in the same organizational unit without being able to give clear reasons, aside from developer preferences (see McDonald and Welland 2001a).

Web project management is responsible for utilizing the enthusiasm of its staff to select suitable technologies and tools. These people are extremely keen on getting trained and further educated, and it is not important for them whether these training programs are offered during working hours or in their spare time. Once the development environment and the approach models have been specified, it is necessary to define an orderly change and update policy. Care should be taken that this policy is observed. The specified tools have to actually be used, and no other tools should be used prior to being tested and approved.

3 (Friedlein 2002) compares the task of a Web project manager accurately with the work of a composer.

Immaturity

Many Web application environments are constantly updated or changed not only because of the developers' enthusiasm for innovation; they are often immature so that error removal, interface expansion, etc. represent important productivity increases. Many immature tools are used only due to the deterrent pricing policy of some tool vendors, or because there are no other alternatives.

As a consequence of the extremely fast pace of technological development, experience with tools is poor, which can lead to a considerable hindrance of a Web application's evolution and frustrating experiences among the staff. Many release changes of a Web application are tied to changes in the development environment, often due to newly introduced techniques and standards for the Web or changing browser technologies. The consequence is that hardly any Web application can build on the technology of its predecessor, which means that experience and knowledge are lost or not built in the first place. One strategy to deal with this problem in Web project management can be to rely on the technologies and tools of large vendors, because they are likely to follow an orderly update and release policy and continuous assistance. Open-source projects from reliable sources (e.g., the GNU environment or the Apache foundation) have become interesting alternatives thanks to their Web community support.

9.2.3 Product-related Challenges in Web Projects

The following properties are typical for Web applications or their use, representing special challenges for Web project management.

Apparent Simplicity

The early period of Web application development, which dates back only a few years, created the impression that designing Web applications is very easy. In fact, this was true for the first, static hypertext pages, at least if we ignore the linking logic. It was a common belief that everybody could use some text, HTML commands, and pictures to quickly and easily create Web sites.

However, modern Web applications are fully fledged software systems (see Chapter 1). In addition to the user interface, they include a sophisticated process logic (e.g., created in Java) and connect to large databases or data warehouses for delivery of information; the main pieces of information are generated dynamically just in time. Naturally, these applications are called from within the same browsers (though in a more recent and more extensive version), so that many users (and even operators) don't see a difference to the early static Web pages. It is often hard to convey the development costs and the required software systems and computing power to these customers.

Aesthetics

The Web (or rather, its applications) is referred to as the "most fashion-aware software". The look of Web pages is subject to more updating and fashion trends than any other software; to stay "in" with one's Web design is seen as the critical success factor (Pressman 2005). This necessity to change products from the artistic/aesthetic view accelerates the pressure driven by the technical evolution to change things even more.

Looking back at completed Web projects, however, it can be observed that most aesthetic changes of Web applications are merely required reactions to (aesthetic) changes of the content. Slogans, logos, etc., are frequently adapted. Certain content becomes fashionable and disappears just as fast as it has emerged. Many of these aesthetic aspects concern only the "static" area of Web applications. If the Web project manager plans sufficient flexibility and dynamics for this partial area (similar to the multi-language area, for example), then aesthetic changes to applications can be implemented quickly and often even purely within the content, without intervening in the code.

Spontaneity

A *Web user* cannot be expected to be *loyal* to a Web vendor. In this respect, (Holck and Clemmensen 2002) suggest that people use Web applications only when they see a direct benefit in them. In contrast to many traditional information systems, there is no pressure to use a specific Web application. If users don't like a Web application, they will find other ways to obtain the information they are interested in.

Spontaneity leads to another consequence: the *use* of a Web application has to be (widely) possible *without instructions*! Web application users are even less willing than users of conventional software systems to read extensive (online) instructions, let alone paper manuals. This means that Web applications have to be self-explanatory and feature a highly repetitive control flow (navigation!). The usage logic has to be uniform across the entire Web application, so that users can acquire operating routine quickly and feel "comfortable" with the Web application.

Good *usability* (see Chapter 11) is a critical success factor for Web applications, which cannot be stressed enough in the planning phase. As an aid for web project management, the American National Institute of Standards and Technology supplies free tools to evaluate the usability of web applications (National Institute of Standards and Technology 2005). These tools test Web sites according to a set of usability guidelines, do category analyses and classifications, automatically insert code to log user behavior, and visualize possible navigation paths. For details, refer to (National Institute of Standards and Technology 2005).

Ubiquity

In addition to being available worldwide, mobile devices have made the Web available at virtually any location. The potential user group for a (publicly accessible) Web application is accordingly large. This means that the number and characteristics of the actual user group are unknown, and the spectrum of potential user types is extremely wide.

This situation leads to the problem that it is impossible to determine a representative user group as a basis for identifying requirements during the development of a Web application (see Chapter 2 for a more detailed discussion of this issue). Even though a test application deployed on the Web can reach Internet users, one can often observe neither approval nor denial, but merely indifference (Holck and Clemmensen 2002) due to the spontaneous interaction behavior. If Internet users like an application, they will use it, but they will hardly spend time and cost to contribute to its improvement by providing feedback. The collection and validation of requirements is clearly more difficult for Web applications, compared with conventional software development, confronting Web project management with much insecurity.

Compatibility

People use browsers to visualize and use a Web application. Though only a handful of browsers dominate the market (Microsoft Internet Explorer, Mozilla Firefox, Opera and other Mozilla-based browsers), they have a very different *compatibility behavior* and often unclear support of certain standards (HTML, CSS, Java, etc.). In addition, the platform adaptations and software versions of these browsers differ, and neither downward nor upward compatibility is necessarily a given. A uniform design of the set of functionalities and user guidance is, accordingly, hard to implement and maintain due to frequently surprising and serious changes effected by browser vendors. On the other hand, limiting Web applications to users with specific browsers would aggravate large potential customer groups, or force them to run several browsers in parallel on their devices.

In addition, browsers can be extensively *configured* by their users. Users can change both the look and access capabilities and functions. A solution that would allow Web project management a certain security in development doesn't appear to be forthcoming in this respect. Though there are test benches that can be used to test the compatibility of a Web application with various browsers and browser versions (for example, at `http://www.cast.org/bobby`, `http://www.etest-associates.com`), they normally lag behind the development of browser and Web technologies.

Stability and Security

Users expect Web applications to be *available round the clock* (24×7 operation). These expectations mean high requirements on application quality with regard to reliability, but also on the underlying hardware and network connection. In addition, an application has to ensure that unauthorized users cannot access the private, confidential area of a Web application, neither inadvertently nor intentionally, by exploiting security holes.

This requirement represents an enormous challenge for the maintenance of Web applications, because they have to be maintained either on the production system or in parallel to it. In the latter case, synchronizing the development system with the production system and testing (while data change dynamically) are especially difficult. An appropriate configuration management system, combined with logging mechanisms (e.g. "WinRunner" – `http://www-svca.mercuryinteractive.com/products/winrunner/`) is, therefore, essential for the sound development of web applications.

Scalability

The ubiquity of Web applications, combined with the spontaneity of users, imply that Web applications have to be scalable to an extent that cannot be projected prior to and during development.

A poorly scalable Web application can cause a dramatic loss of performance perceived by all of its users as soon as their number reaches a certain maximum. This situation can aggravate the entire user group, or cause data loss in the worst case. A poorly scalable e-banking system, for example, can lead to enormous material loss (incomplete or incorrect financial transactions) and immaterial loss (corporate reputation).

The most important aspect for Web project management aside from the scalability of a software product is to consider simple expansion of the hardware structure (e.g., server farms), ideally

without interrupting normal operation. Some scalability aspects (e.g., bandwidth of the network connection to a user), however, are beyond the influence of the developer or site owner.

9.3 Managing Web Teams

9.3.1 Software Development: A Human-centered Task

Due to the rapid evolution that a Web application project is particularly subject to, and due to the fact that modern software developments are all managed by groups of people and no longer by individuals (Mayr 2005), the communication among the team members and their motivation and coordination by the project manager are among the most important success factors for a project (see also Chapter 2). For this reason, software development is often called a *human-centered activity* (Humphrey 1997).

Technical managers are particularly inclined to underestimate the psychological and social aspects in development teams. As soon as they have been identified, conflicts have to be addressed and resolved. In the field of technical development, there is often no room for compromise; there are "winners" and "losers", which leads to more conflicts. In the long term, however, it is not possible for the staff and the project to try to avoid conflicts at any cost.

As mentioned in section 9.1.4, Web application development teams are usually rather small. For software projects with a team size of less than ten people, the "surgical team" as known from medicine was recommended as a model ("chief programmer team") for software development back in the 1970s (Baker 1972). The following changes in software development culture led to the evolution of this chief programmer team:

- A software engineer can no longer divide his or her work area into design, implementation, and testing. He or she has to know all these techniques and must be able to use them concurrently.
- A substitute has to be identified who can replace each team member in an emergency situation, at least for a short period of time. This substitute should continuously be kept informed about the current project state (cf. "pair programming" in section 10.5.1). Since modern team members must be experts in a broader field of competency and should be able to substitute for other members, the term "team performance" (focusing on specialization) is increasingly being replaced by "group performance" (focusing on the holistic project approach for each group member).
- A chief programmer with both perfect management qualities and up-to-date implementation skills is rarely available in practice. Therefore, a modern project manager should have content-specific understanding of the implementation, but not actively participate in it.
- The unrewarding – and often less interesting – tasks of a project secretary as "chief documenter" and controller makes it difficult to fill this position permanently. Document generation should be distributed across the team, according to the tasks.

Due to the intentional toleration of group-dynamic processes for self-regulation, it is important to plan enough project time at the beginning of the project to form the group and clarify the relationships between group members. Since software development is mainly a planning activity, and thus of intellectual nature, staff leadership is very important, and the project manager should have appropriate social skills. Due to the heterogeneous composition of Web teams, this challenge is even greater.

In software development, the work ethic is extremely significant. Monitoring and control are not sufficiently emphasized in many projects, or they cannot be performed at an appropriate level and amount. This means that the obligation of the team members to do as good a job as they can by themselves is an extremely important factor, both technically and economically (see initial experiences with the introduction of a "professional software engineer" title Meyer 2001).

9.3.2 The Web Project Team

Teams formed to develop Web applications are characterized by three important properties:

1. *Multidisciplinarity*: Since a Web application is composed of content, hypertext structure, and presentation for an – ideally – very broad audience, Web developers have to have different special domain knowledge.
2. *Parallelism*: While the tasks in traditional software projects are divided by development-specific aspects, Web projects are typically divided by problems. The result is that subgroups of a Web project team are similarly composed with regard to their expertise, which means that many parallel developments have to be coordinated (see section 9.2.2). The communication effort is, therefore, higher in Web project teams than it is in the development of traditional software systems.
3. *Small size*: Due to short development cycles and often a rather limited budget, Web project teams are composed of a small number of team members (around six on average, and rarely more than ten (Reifer 2002, McDonald and Welland 2001a)). Larger tasks are subdivided and worked on by subteams in parallel.

The important aspect is that each member fully understands their roles and responsibilities. If roles and responsibilities overlap, then the team members, together with the team manager or project manager, are responsible for solving conflicts that may arise in the project. Since the development cycles of Web projects are short, it is also important to solve conflicts quickly, even though this may lead to suboptimal solutions from an overall perspective.

Figure 9-3 shows what a typical composition of a Web project team might look like. Each of the roles may be further divided into special fields. For example, "software engineer" can include software architects, (Web) programmers, database administrators, Web masters, etc.

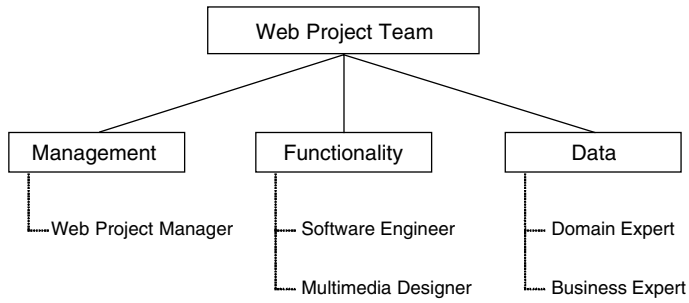


Figure 9-3 Typical composition of a Web project team.

Application experts supply the content for a Web application. Business experts also supply content; in addition, they contribute particularly to the overall structure of a Web presence and its orientation to the company's business objectives (added value for the PR portfolio, additional product marketing vehicle, customer assistance, etc.). McDonald and Welland (2001a) suggests that application and business experts have additional communication needs since they often work geographically separated from the rest of the development team.

A special problem relates to *assistance during operation and maintenance* of a Web application, which is particularly important in Web engineering. All mentioned roles of a Web project team are required for this assistance. Since the individual team members generally focus on other projects as soon as the development of a Web application is completed, it is important to introduce a tool-supported, joint project management for parallel projects, a so-called *multi-project management*, in companies that develop Web applications.

9.3.3 The Web Project Manager

The key task of a Web project manager – that distinguishes him or her from traditional software project managers – is that he or she has to lead a team composed of people with different abilities and competencies. The team members are specialists with different degrees of knowledge, habits, and values. Every type of developer appears to have problems in recognizing and appreciating the contributions made by developers coming from other educational backgrounds (McDonald and Welland 2001a). And vice versa, experts have difficulties in even rudimentarily estimating tasks that don't fall within their scope. This behavior is not purely specific to Web engineering, as the author has noted when assisting a group of game developers, where experts like game designers, asset managers, and implementers dealt with each other in a very similarly conflicting way. In such cases, the project manager has to assume the role of a translator and mediator, who has to translate both the contents and the motivations for decisions from one field of application and one language to the other(s).

To meet this multidisciplinary communication task, a Web project manager has to be able to encounter organizational, technical, and social challenges. The mostly *organizational challenges* for a Web project manager were discussed in section 9.2.2 in connection with the properties of development-related Web project management, while the mostly *technical challenges* were discussed in section 9.2.3 in connection with the properties of product-related Web project management. A Web project manager has to primarily deal with the following *social challenges*:

- Inspire all project members with the project objectives.
- Be capable of leading a multidisciplinary team.
- Create the willingness and readiness for (democratic) cooperation.
- Constantly motivate the team and solve conflicts.

Since most project team members are young and inexperienced, the Web project manager not only has to lead them, but must also train them in the fields of communication and social competence, and motivate them to participate in further education.

Of course, the most important task of a Web project manager from an economical point of view is that of all project managers: he or she ". . . has to constantly keep an eye on the project

progress against all hindering influences on time, cost, and quality with regard to customer, work place, market forces, other external factors, and the development team" (Friedlein 2002).

Another important task of the Web project manager is the assistance and integration of the customer during the development of a Web application. There are two peculiarities to this task, compared with conventional software projects:

1. The transition of a Web project from development to regular use is fluid. Also, it is frequently difficult for the Web project manager to determine when a Web application has been fully taken into operation, and thus when the actual development project has been completed and regular use (including maintenance) of the developed Web application has begun.
2. In addition, it is often unclear whether or not a Web project manager should still be involved with the project once the application has moved to the operation and maintenance phase. This becomes more critical because of the fact that, due to the special knowledge of single members of a Web project team, the main contact with the customer in Web projects is not maintained through the Web project manager, but directly through experts (see Burdmann 1999). This fact could be another sign of the immaturity of Web project management as currently practiced.

Table 9-3 shows the most important rules a Web project manager should observe to ensure successful management of Web projects.

Table 9-3 Ten golden rules for Web project managers

	Ten Golden Rules for Web Project Managers
1	Promote the professional self-conception of each team member. Take care of ethics in the team.
2	Stress the importance of different application knowledge for the project.
3	Solve conflicts quickly. Make sure no team member is a winner or a loser all the time.
4	Explain to each team member his or her roles and responsibilities continuously.
5	Identify parallel developments and utilize potential synergies.
6	Distribute documentation tasks to team members fairly according to their scope.
7	Promote and coordinate the continuous use of tools from the very beginning of the project.
8	Translate costs and values into different project areas.
9	Promote the continuous integration of the customer in the project.
10	Always keep an eye on the project progress *and* the project objective.

9.4 Managing the Development Process of a Web Application

9.4.1 Deploying the Tools

Web applications are necessarily developed using a very flexible approach, characterized by a high degree of reusability, agile process methods, close customer contact, and a large number of intermediate products. Changes as may be required to the development processes used for

Web engineering will be discussed in Chapter 10. Chapters 11 through 13 will discuss important quality assurance aspects.

However, with regard to the tools used and the (intermediate) results produced, one can also observe a transition from the document-driven approach in traditional (rigid) software development towards a highly tool-supported approach in agile development projects (Orr 2002). This transition to tool support in agile approaches is shown in Table 9-4, which uses the documentation model from (Mayr 2005).

Table 9-4 Increasing use of tools in agile approaches

Important in rigid approaches	Equally important	Important in agile approaches
	Organizational chart, roles	
	Project library, diary	
	Protocols	
Progress reports		Interim products, prototypes (protocols)
	Configuration management (tool!)	
	Quality characteristics	
	Objective description	
Requirements specification		Requirements suite (tool!)
	Project plan	
Master plan		Strategic idea, operative plan (tool!)
		Risk management plan
User documentation		Interim products, prototypes
System specification		Design model (tool!), continuous modeling language
	Source code	
System documentation		(Application) data
	Executable program	
		Test plan, test suite
Error report, error log		Error management (tool!)
	Product version	
	Installation protocol, commissioning protocol, acceptance protocol	
Final report		Maintenance plan
	Project archive	

Agile approaches increasingly require tools, e.g. for requirements management, planning, design/implementation, test planning and implementation, and error handling as well as continuous configuration management. This applies in particular to young and inexperienced developers, since they may easily lose their grip on the project, especially if it is highly iterative. In one of the few studies that explicitly deal with management problems in Web projects, (Cutter Consortium 2000) suggests that "... iterations can easily become ad-hoc developments without tools for

progress measurement, change management, and results review . . . [and] . . . careful monitoring of the requirements both by the developer and the customer". The same source mentions that Web projects in particular have to be handled based on a documented plan, and that extensive and careful testing is unavoidable, even if the project already suffers from delays.

These tools must interact well (or ideally be integrated) to ensure that a development project's efficiency improves (see the Eclipse project at `http://www.eclipse.org` for an example). However, care should be taken to ensure that the development process is independent from these tools and technologies (McDonald and Welland 2001a). Since the technologies available to develop Web applications change rapidly and in an unpredictable way, the development process should be clearly separated from implementation tools and languages.

In many cases, when moving to an agile approach, people forget that the availability of these tools is only one side of the coin; handling them has to be learned and practiced, too. The time required for such learning processes frequently is not planned for in short Web projects, or their cost cannot be justified. Since the developers of Web projects are typically inexperienced, individual "autodidactic" learning of agile methods and the required tools could, in fact, lead to a situation where everything subjectively unimportant is omitted, while hacking makes its entrance (Beck and McBreen 2002).

Tools for Web Project Management

Since Web developers are especially familiar with the Web, Web-based tools are ideally suited for Web project management. Web-based project management tools, such as *PHProjekt* (`http://www.PHProjekt.com`), allow to handle traditional project management tasks, such as time recording, maintaining a diary, archiving and versioning result documents, logging, blackboards, chat rooms, e-mail distribution, etc., to support the Web team's communication. Many of these tools are even free for personal and educational use. In addition, such tools facilitate communication and collaboration beyond local boundaries, which frequently occur in Web projects. Project Management Institute (1999) gives an overview of project management tools, including Web-based tools.

Tools for Configuration Management

A *configuration management system* (Dart 2000) is an important tool to support an orderly project process. In Web engineering, configuration management is used mainly for the following tasks due to the short iteration cycles:

- *managing versions* of the source code and the application content, and regulating access to the content,
- *creating configurations*, both for the source code and the application content to establish an orderly release policy,
- *managing change requests* and handling errors and defects,
- *controlling document state* to monitor project progress.

Creating *variants* is less common in Web engineering. If variants are created (by branching in the development tree), then the reason is often just to complete a customer version due to the

short iteration cycles, while other team members are working on product progress along a second branch (see section 10.3.4 for notes on parallel development of different releases).

Since many Web projects start small and then successively grow, it is important to use configuration management systems from the very start of the project even if the project would then seem "too small" for tool-supported configuration management. Changing to configuration management later on would be costly and time-consuming. Moreover, the evolution history cannot be tracked unless one uses a configuration management tool from the start of the project. When a Web project is broken down into many small subprojects and their results have to be consolidated for each (intermediate) product, a configuration management tool will quickly become an indispensable part of the Web application development project (Baskerville and Levine 2001).

Configuration management allows to specify who may change what and when, and it lets one control these changes. Especially in Web engineering, components of an application are often created by people less experienced in software engineering. They often create a large number of versions of one single component or document, quickly losing track of the purposes of each of these versions, or whether a version is still up-to-date. This situation causes the amount of documents in Web projects to grow dramatically, particularly because documents with multimedia content (so-called *assets*, comprising, e.g., music clips or videos) require a lot of memory. Many tools available to manage configurations of software being developed (e.g., Visual Source Safe, `http://msdn.microsoft.com/ssafe`; Subversion, `http://www.subversion.com`; ClearCase, `http://www.rational.com/products/clearcase`) have difficulties in processing assets (large memory demand) and identifying version differences (binary format). Some configuration tools may not even be able to process such assets due to the data demand. This is probably the reason why many configuration management tools for multimedia are still being developed individually (Wallner 2001).

9.4.2 Measuring Progress

When developing Web applications, often just two documents are created (McDonald and Welland 2001a). The first document is the *system specification*, containing the results from the requirements analysis and design decisions. The second "document" is the finished Web application. This Web application is normally created in a quick sequence of intermediate results by means of highly iterative and evolutionary prototyping. Sharma (2001) suggests that the iteration steps should be as small as possible and have clearly defined functionality. Each of these steps should be followed by a review, ideally involving the customer.

This approach is identical with Rapid Application Development (RAD), a technique that has been well known and frequently practiced in traditional software development for almost twenty years. The development of Web applications is characterized by the fact that their requirements generally cannot be estimated beforehand, which means that project size and cost cannot be anticipated either (Reifer 2002). In addition, the usual high pressure with regard to a defined and very short delivery deadline means that "the quality of a fixed-time product should be estimated rather than the costs for a well-specified system" (Pressman 1998).

The difficulties in anticipating these factors, and the subsequent progress control of Web projects have been summarized by D. Reifer (see Table 9-5 adapted from (Reifer 2002)) and

Table 9-5 Estimating traditional software development projects versus Web development projects

Criterion	Traditional Software Engineering	Web Engineering
Estimation process	Analogies, experience	Structured by developers
Measurement	From requirements	Projection of frameworks
Measure	Source lines of code (SLOC); function points (FP); object points (OP)	–(not uniform)
Development effort	Cubic root relation (Boehmet al. 2000)	Cube root too long; (square root?)
Calibration	Experience, comparative values	–(not uniform)
Risk estimate	Quantitative	Qualitative (no models available)
Return on investment (ROI)	Estimation models	–(not uniform)

used to develop a Web-specific project metric.[4] This metric considers the use of Web-specific components and the large amount of reuse in the following manner. Analogous to "software objects" of the COCOMO II model (Boehm et al. 2000), Reifer defines so-called "Web objects", enabling the use of COCOMO-like object points for Web components (Reifer 2002), where the functional links have to be adapted considerably (see Reifer 2002 for details and Olsina et al. 2005 for an overview of other Web metrics).

Of course, clearly defined counting guidelines for Web objects are important to obtain meaningful estimates, similar to any other metric. However, this doesn't solve the main problem that development-related design components are counted, i.e., that there is more input on the developer side and less on the customer side. Counting is a nice method to measure progress. But it doesn't answer the question of whether or not the degree of objective coverage can be improved.

Table 9-6 summarizes recommendations for an appropriate use of tools in Web projects.

Table 9-6 Recommendations for an appropriate use of tools

	Recommendations for an Appropriate Use of Tools
1	Separate the development process clearly from tools, models, and languages.
2	Pay attention that tools are usable throughout the development and are easy to integrate.
3	Start using tools early. A later introduction/change of models is cumbersome and unsatisfactory.
4	Use tools and processes that efficiently support the iterative and evolutionary approach and customer feedback.
5	Plan extra time for training and familiarization with each tool.
6	Check the necessity and consequences prior to each tool or release change.
7	Do not only measure project progress, but also the degree of objective coverage.

4 IEEE (IEEE Standard 610.12-1990 – Software Engineering Terminology) defines a *metric* as a quantitative measure for the degree in which a system component or a system process has a certain property.

9.4.3 Project Risks

Risks in Software Development

A *risk* is the possibility of an activity to cause loss or damage. However, we speak of risk only if the consequences are uncertain. This means that a risk represents a potential problem (Thayer and Fairley 1997).

There is no such thing as a project without risks (and problems resulting from them). "If a project is successful, then it is not successful because there were no risks and problems, but because risks and problems have been handled successfully" (Rook 1986). Each risk also represents an economic chance. A shorter development cycle can translate into a desirable competitive edge for a product's time-to-market. The venture of using a new technology can open up additional market segments. For example, the use of the .NET platform allows the deployment to Microsoft-based mobile phones as possible target devices.

The most important risks in software development have been identified and updated regularly by B. Boehm, one of the pioneers in the field of risk management in software engineering, since the 1980s. Table 9-7 (adapted from Boehm 1998) compares the ten most important risks for software projects in the 1980s versus those in the 1990s. One can see that the term "process" doesn't even appear in the older list, while it represents the highest new entry in the 1990s list.[5]

Table 9-7 The most important risks in software projects according to (Boehm 1998)

No.	1980s	1990s
1	Personnel deficits	Personnel deficits
2	Unrealistic time and cost specifications	Unrealistic time and cost specifications; insufficient process attention
3	Development of wrong product properties	Deficits in third-party components (COTS)
4	Badly designed user interface	Misunderstood product properties
5	"Gold-plating" (implementing unnecessary properties)	Badly designed user interface
6	Creeping functionality changes	Poor architecture, performance, quality in general
7	Deficits in third-party components	Development of wrong product properties
8	Deficits in outsourced tasks	Building on legacy systems or embedding them
9	(Real-)time performance	Deficits in outsourced tasks
10	Over-exploiting the technologies	Over-exploiting the technologies

The second new risk element of the 1990s represents the use of legacy software, i.e., the use of historically grown software systems. In the 1980s, it was common to think that this problem could be solved by re-implementation, which turned out to be wrong. Especially in the development of Web applications, many legacy systems have to be embedded, such as data management/storage

5 B. Boehm has not yet published an updated list for the current decade.

systems. For example, the author supervised a project for a large metal spare parts supplier who wanted to offer their services on the Internet. Within this project, a COBOL environment with a hierarchical database as the primary information system had to be embedded into the Web application to be built (by use of wrapper technologies). Neither did the company want to replace their existing database system, nor was it feasible, because the entire corporate operation depended on this database, as the company committed to extremely short delivery times as its major trade mark.

Specific Risks in Web Engineering

In Web engineering, the main reasons for delays or total failure of Web projects are identical to the main risks listed by B. Boehm (see Table 9-7). Projects fail primarily due to bad communication within the Web project team and with the customer, i.e. personal deficits. The second important reason is poor understanding of the development processes for a Web application (McDonald and Welland 2001a).

J. Nielsen, the "guru of Web page usability", mentions the following top-ten list of risks in a Web project, especially addressing Web project management (Nielsen 1997b):

1. *Unclear definition of objectives*: Many Web applications are created without clearly explaining the purpose of the application to the developers. Section 9.3.2 mentioned that a company can address totally different customer objectives with a Web application; a service application with help desk meets totally different tasks than an online catalog with ordering function. It is important to clearly specify the objectives and how customers can profit from a Web application, and to explain these objectives to everybody on the team. McDonald and Welland (2001a) mentions poor objectives research and requirements analysis as the main reasons for the immaturity of current development processes for Web applications.
2. *Wrong target audience*: Many Web applications are heavily oriented to the wishes of corporate management (mission statements, manager profiles, etc.), while failing to address the application's target group.
3. *Development-oriented page structure*: The easiest way for developers to create the content is by following the contractor's organizational chart. Once this structure has been applied to an application, it is frequently no longer possible to follow a customer-oriented operation. The application will be focused on the company rather than on the users.
4. *Lacking consistency due to outsourcing*: If the development of different Web applications of the same company (or parts of a larger Web application) is outsourced to different external companies, there is a high risk that the individuality of these external companies will strongly reflect in the applications, especially in the contents and navigation aids. Each of these companies will introduce its profile and try to distinguish itself from its competitors, leading to a botched-up job with regard to the look and feel and the operation of the application. Pressman (2000b) suggests defining the target audience, its interest in the product, and the general design either internally or by one single company functioning as a coordinator. In addition, regular reviews of the interim products are necessary to ensure that inconsistencies between application parts can be identified early and removed.

5. *Lacking budget for maintenance*: While traditional software development projects estimate 10% to 15% for annual maintenance (Mayr 2005), the maintenance cost for Web applications is much higher due to their quickly aging contents and rapidly changing technologies. Nielsen (1997b) recommends estimating at least 50% of the development cost (100% is more appropriate) annually for maintenance.
6. *Content recycling*: Many try to extract contents for Web applications from traditional document sources. It is often overlooked that these media are generally designed to be read and looked at linearly (from front to back). In contrast, the Web lives on the non-linearity of the hypermedia structure and its linking. The possibilities offered by a non-linear media structure are not utilized, because many application and business experts have grown up on purely linear content (printed books, newspaper articles), and they have been educated to create linear content for traditional media. The rethinking process has been slow in this field.
7. *Poor linking*: The recycling of linear content leads to the problem that links are often added "artificially", which can be detrimental rather than beneficial (e.g., when converting printed manuals into online help systems). In addition, many links point to higher-order pages rather than pointing to the specific content. It is then up to the user to find the desired content from the general starting point and to have the patience for doing this.
8. *Mixing Internet and intranet*: While an Internet presence represents a company to the outside, thus transporting corporate culture, PR ideas, etc., in addition to information, to court potential users, an intranet serves to efficiently support the work of an exactly defined user group – the company's staff. For this reason, efficiency of Web applications in an intranet should clearly be given preference over PR and (exaggerated) aesthetics. Within the intranet, productivity is the goal, and the users are assumed to be loyal. This means that Internet and intranet applications have to be clearly separated. When developing a Web application, the developers need to know in advance whether it is created for the Internet or an intranet.
9. *Confusing marketing research and usability research*: While marketing research assesses the wishes of users, usability research is aimed at finding out how users handle a Web application and what problems they have. Marketing research can never find out that a Web application's particular functionality is desirable, but not attractive because it is hard to use. Confusing the two fields of research can lead to wrong assumptions, similar to what happened to some car suppliers who, at the end of the 1990s, cut down on complaint hotlines for cost reasons to then happily noticed a clear drop in complaint cases per product.
10. *Underestimating the strategic significance of the Web*: Web presence has become taken for granted for every company. In the long term, ignoring this communication channel will lead to significant competitive disadvantages. However, many companies have overestimated the significance of the Web (cf. the "dotcom crash") and have then made the mistake of developing a Web aversion, missing the bus in the long term.

In summary, it is important for each Web project to bear the most critical risks in mind and to ensure that they will be overcome. Limiting and mastering these risks are risk management tasks, which will be discussed in the next section.

9.4.4 Risk Management

Risk management means to proactively take into consideration that there is a certain probability that problems may occur in a project. This is done by estimating the probability, analyzing the impact of potential problems, and preparing suitable problem solutions in advance (Bennatan 2000). In this way software development "without surprises" is made possible.

Figure 9-4 shows the tasks involved in risk management. Both the risk management tasks and the processes are the same in Web engineering and in other fields of software development. The only difference lies in the kind of risks to be handled. We will thus not discuss each step involved in risk management in detail and instead refer our readers to (Mayr 2005).

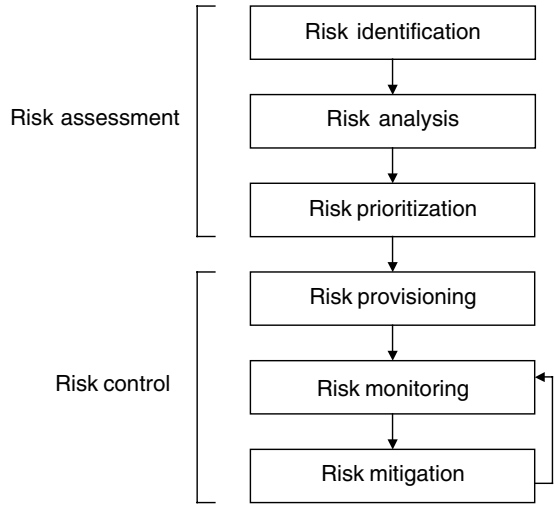


Figure 9-4 Tasks involved in risk management.

Risk management can be thought of as a form of insurance that covers a problem by immediately providing a solution as soon as the problem arises. In many cases, a problem that has been analyzed and caught in time is much easier and cheaper to solve than a problem that turns up unexpectedly. Risk management doesn't provide general recipes for all kinds of cases. However, the literature points to the fact that, while working on risk management activities, groups perform much better than individuals (Gemmer 1997). A risk-driven method, e.g., a spiral model, allows the group to optimally integrate risk management into the project development process.

Though risk management doesn't come for free, it should be cost-efficient, i.e., its benefits should outweigh its costs. However, since risk management is supposed to avoid problems, good risk management can face the situation of justifying the costs spent for analyzing problems that never occurred. For this reason, a sound cost-benefit analysis is indispensable to justify risk management activities (Pressman 2005).

9.5 Outlook

The "agile-versus-rigid discussion" in the field of software development methods has started a religious war (Orr 2002), which has recently been transported to project management. For example, the proposed *Extreme Project Management* (Thomsett 2001) turns out to be rather conventional on a closer look. On the other hand, *Scrum* (Highsmith 2002, Schwaber and Beedle 2001) shows that transporting XP (Extreme Programming) ideas to project management can indeed be successful, if the concepts are not implemented too radically.

The pendulum that has been swinging from very rigid methods (e.g., CMM) to very agile methods (XP) will stabilize somewhere in the middle in the foreseeable future, similar to earlier disputes about methods (the "data war" of the 1980s, the "OO war" in the 1990s; see Orr 2002).

Another trend is that managers have tried to learn inductively from their process experiences, passing these proven practices on to others. The notion of a "pattern" was transported to project management (Ambler 1999a, Ambler 1999b), and it has meanwhile been enhanced by collections of negative examples ("anti-patterns"; Brown et al. 2000).

When one looks at the success statistics of software projects, the picture is sobering at first sight. Figures available from different survey groups on the success of projects in the US software industry are summarized in Table 9-8. The data are based on general software projects (each survey has a base of several hundred thousand projects). For Web projects, no surveys of comparative size are known to the author, except, partially, (Cutter Consortium 2000), which is, however, also widely based on data from general software projects.

Table 9-8 Development of success statistics of software projects in the US

Surveys of the success of software projects in the US	Project successful	Project over budget or over time	Project discontinued
Standish Group (1994) (http://www.standishgroup.com)	16%	53%	31%
Center for Project Management (1995) (http://www.center4pm.com)	25%	50%	25%
Standish Group (2000) (http://www.standishgroup.com)	28%	49%	23%
Cutter Consortium (2000) (http://www.cutter.com)	16%	63%	21%
Gartner Group (2000) (http://www.gartner.com)	24%	51%	25%
Standish Group (2004) (http://www.standishgroup.com)	29%	53%	18%

The results from these surveys show that, both in the 1990s and at the beginning of the new millennium, only about one quarter of all projects were successful. Three quarters were either

considerably over budget or over time (approx. 70% on average), or discontinued for lack of success.

At first sight, these results shed no good light on software project management. However, the reader should bear in mind that the size of software projects continuously increases, and that project constraints (deadlines, resources, etc.) have become increasingly tougher. This has been the case particularly since the turn of the millennium (Paulson 2001). Without the progress project management has achieved, the success rate would probably have dropped considerably!

The available data suggest that the success statistics of software projects will not change essentially during the next few years. Taking the current intensification of project constraints into account, combined with the explosive technological development, the current situation of Web project management should not be considered a stagnation, but a success.