Chapter

2

ITERATIVE & EVOLUTIONARY

Experience is that marvelous thing that enables you to recognize a mistake when you make it again.
—F. P. Jones

OVERVIEW

- ❑ Basic practices of iterative and evolutionary methods, including timeboxing and adaptive planning.
- ❑ A common mistake adopting iterative methods.
- ❑ Specific iterative and evolutionary methods, including Evo and UP.

Iterative and evolutionary development is a foundation not only of modern software methods, but—as the history section of the "Evidence" chapter shows—of methods used as far back as the 1960s. Agile methods are a subset of iterative and evolutionary methods. This chapter summarizes key practices:

history p. 79

iterative development	*evolutionary development*
risk-driven and client-driven	*evolutionary requirements*
timeboxing	*adaptive planning*

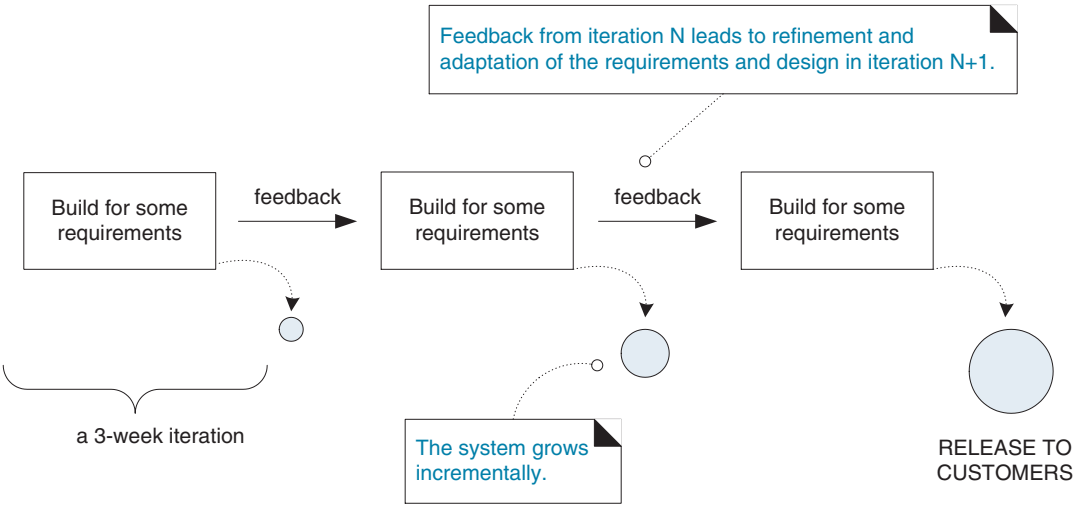
ITERATIVE DEVELOPMENT

Iterative development is an approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence. Each **iteration** is a self-contained mini-project

iterative planning tips start on p. 248

composed of activities such as requirements analysis, design, programming, and test. The goal for the end of an iteration is an **iteration release**, a stable, integrated and tested *partially* complete system. To be clear: *All* the software across all the teams is integrated into a release each iteration. Most iteration releases are *internal*, a baseline primarily for the benefit of the development team—they are not released externally. The final iteration release is the complete product, released to the market or clients. See Figure 2.1.

Figure 2.1 iterative and incremental development



Although an iteration can in theory be only for clean-up or performance tuning, usually the partial system grows incrementally with new features, iteration by iteration; in other words, **incremental development**. The concept of growing a system via iterations has been called **iterative and incremental development** (**IID**), although simply "iterative development" is common. Some older process literature [Wong84] used the term "incremental development" to mean a combination of frozen up-front specifica-

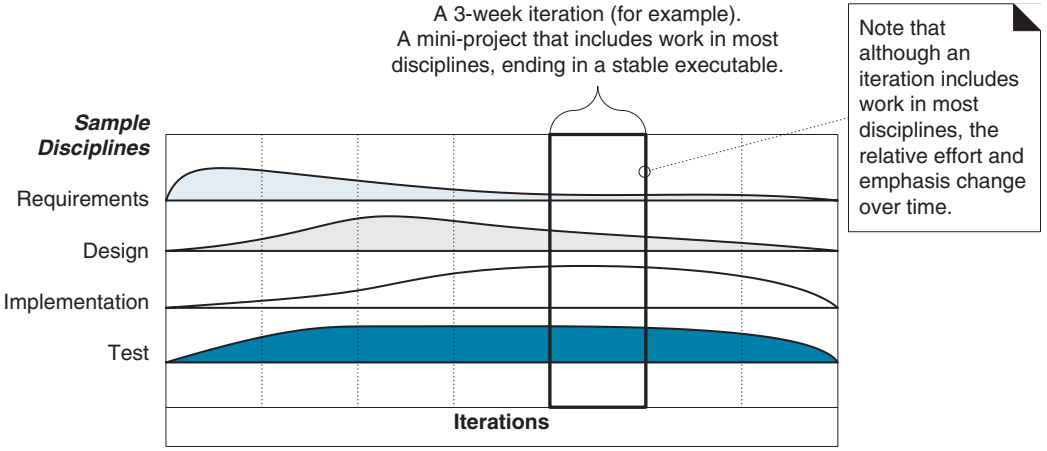
tions followed by iterative development of the features, but there is no widespread agreement on usage. In this era, most development methods are IID methods. And, IID is at the core of all the agile methods, including Scrum and XP.

Most projects have at least three iterations before a final public release; I've seen a two-year Valtech project composed of close to 20 iterations averaging around four weeks each, and I know of at least one long project with 45 iterations.

> In modern iterative methods, the recommended length of one iteration is between one and six weeks.

Each iteration includes production-quality programming, not just requirements analysis, for example. And the software resulting from each iteration is not a prototype or proof of concept, but a subset of the final system.

Figure 2.2 disciplines across iterations



More broadly, viewing an iteration as a self-contained mini-project, activities in many disciplines (requirements analysis, testing, and so on) occur within an iteration (see Figure 2.2).

Risk-Driven and Client-Driven Iterative Planning

risk-driven p. 263

ranking risks p. 273

first iteration p. 268

use cases and iteration planning p. 269

What to do in the next three-week iteration? IID methods promote a combination of risk-driven and client-driven[1] priorities. **Risk-driven iterative development** chooses the riskiest, most difficult elements for the early iterations. For example, maybe the client says "I want the Web pages to be green and the system to handle 5,000 simultaneous transactions." Green can wait. In this way, the highest risks are surfaced and mitigated early rather than late. Risk is a broad concept—maybe you are making a new 3D modeling tool and market research shows that what will capture market interest is a novel, much easier user interface metaphor. The high risk is not getting the UI right.

adaptive and client-driven planning p. 253

Client-driven iterative development implies that the choice of features for the next iteration comes from the client—whatever they perceive as the highest business value to them. In this way, the client steers the project, iteration by iteration, requesting the features that they *currently* think are most valuable. Note that the customer **adaptively plans** the choice for the next iteration, shortly before it starts, based on their latest insight, rather than speculatively at the start of the project. The customer has ongoing control and choice, as fresh information arises.

mixing and ranking iteration goals p. 265

Apply both schemes. Clients do not always appreciate what is technically hard or risky. Developers do not always appreciate what has high business value.

1. Throughout this book, *client* or *customer* could mean a proxy, such as a marketing or product manager for a consumer software product, true end-users for an internal application, etc.

TIMEBOXED ITERATIVE DEVELOPMENT

Iteration **timeboxing** is the practice of fixing the iteration end date and not allowing it to change. An overall project may be timeboxed as well. If it eventually appears that the chosen requests (the scope) for the iteration can't be met within the timebox, then rather than slip the iteration end date, the scope is reduced (placing lower priority requests back on the wish-list), so that the partial, growing system always ends in a stable and tested state on the original planned iteration end date. See Figure 2.3.

multi-site timeboxed iterations p. 248

overlapping activities across timeboxes p. 251

> It is important that timeboxing is not used to pressure developers to work longer hours to meet the soon-coming deadline. If the normal pace of work is insufficient, do less.

In most IID methods, not all timebox lengths need be equal. The first iteration may be four weeks, the second iteration three weeks, and so forth. On the other hand, the Scrum method recommends that each timebox be exactly 30 calendar days. As mentioned, most IID methods recommend an iteration timebox between one and six weeks.

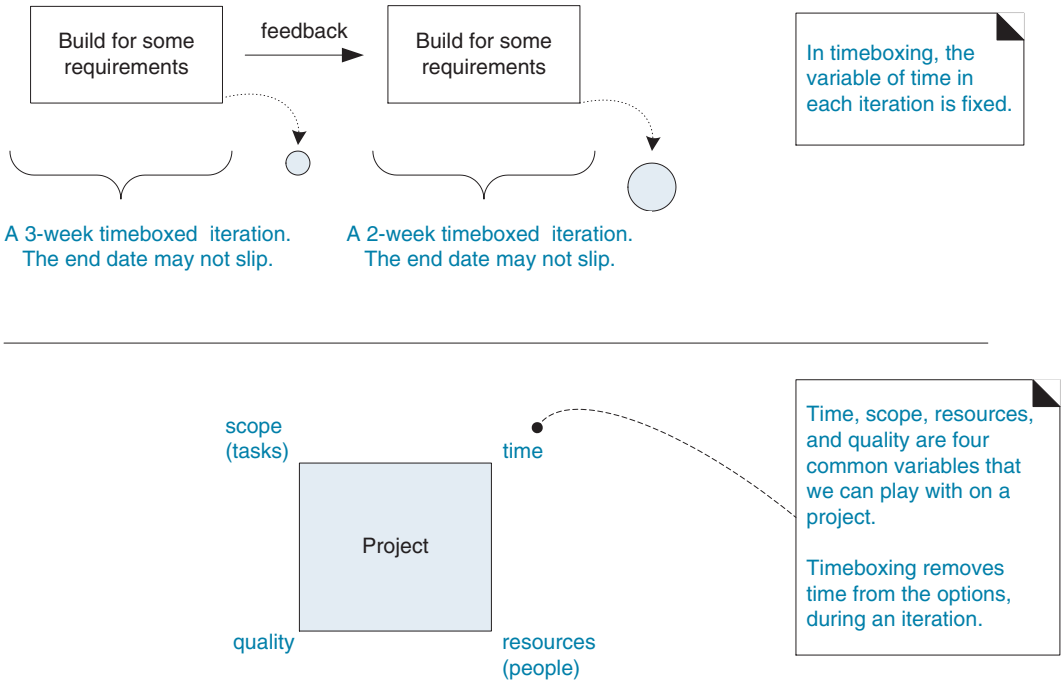
iteration length p. 267

what day to end a timebox? p. 258

A three-month or six-month timeboxed "iteration" is extraordinarily long and usually misses the point and value; research shows that shorter steps have lower complexity and risk, better feedback, and higher productivity and success rates. That said, there are extreme cases of projects with hundreds of developers where a three-month iteration is useful because of the overhead.

All the modern IID methods (including Scrum, XP, and so forth) either require or strongly advise timeboxing the iterations.

Figure 2.3 timeboxing



During the Iteration, No Changes from External Stakeholders

Iterative and agile methods embrace change, but not chaos. In a sea of constant change, a point of stability is necessary. In IID methods this is achieved with the rule:

> *Once the requests for an iteration have been chosen and it is underway, no external stakeholders may change the work.*

scope reduction: primary and secondary iteration goals p. 270

One week into a three-week iteration, the product manager should not come along, and ask, "Can you do this too?" They wait for the next iteration. However, the team itself can reduce the scope of an iteration if the timebox deadline cannot otherwise be met.

Evolutionary and Adaptive Development

Evolutionary iterative development implies that the requirements, plan, estimates, and solution evolve or are refined over the course of the iterations, rather than fully defined and "frozen" in a major up-front specification effort before the development iterations begin. Evolutionary methods are consistent with the pattern of unpredictable discovery and change in new product development.

evolutionary requirements p. 15

adaptive planning p. 17

Adaptive development is a related term. It implies that elements adapt in response to feedback from prior work—feedback from users, tests, developers, and so on. The intent is the same as evolutionary development, but the name suggests more strongly the feedback-response mechanism in evolution.

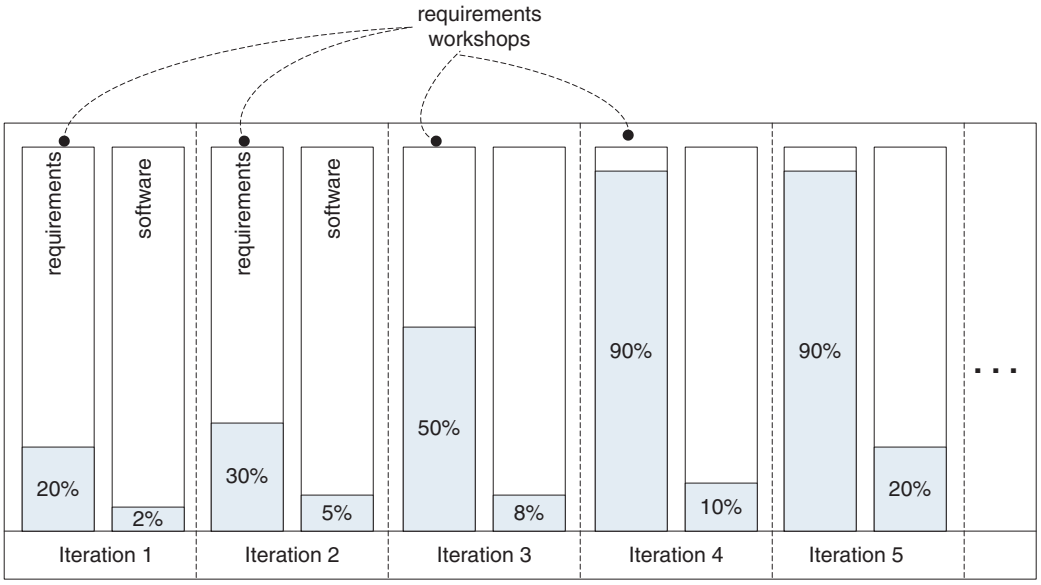
Some methods or methodologists emphasize the term "iterative" while others use "evolutionary" or "adaptive." The ideas and intent are similar, although strictly speaking, evolutionary and adaptive development does not require the use of timeboxed iterations.

Evolutionary Requirements Analysis

In evolutionary and adaptive development, it is *not* the case that the requirements are forever unbounded or always changing at a high rate. Rather, most requirements discovery and refinement usually occurs during early iterations, and the earliest attention is given to understanding the most architecturally significant or high-business-value requirements. For example, on an ultimately 20-iteration project, it is likely that most requirements will be discovered and refined within the first three or four iterations (that include, in parallel, early software development).

evolutionary requirements tips start on p. 281

Figure 2.4 evolutionary and iterative requirements



Imagine this will ultimately be a 20-iteration project.

In evolutionary iterative development, the requirements evolve over a set of the early iterations, through a series of requirements workshops (for example). Perhaps after four iterations and workshops, 90% of the requirements are defined and refined. Nevertheless, only 10% of the software is built.

workshops p. 284

In each iteration, there is a one- or two-day requirements workshop in which the specifications expand and refine, in response to further analysis and feedback from the system under development. See Figure 2.4. For example, the first workshop focuses on detailed analysis of 20% of the most architecturally significant and risky requirements; this gives the software architect enough meaningful input to start development and test in short cycles.

Note as a design comment, that it is not true that 100% of the *functional* requirements need be known to start building an excellent core architecture. The architect needs to know most nonfunctional or quality requirements (e.g., load, internationalization)

and a much smaller representative subset of functional requirements.

Early "Top Ten" High-Level Requirements and Skillful Analysis

It is a misunderstanding to equate evolutionary requirements analysis with "no early requirements" or sloppy requirements practices. Modern IID methods encourage the early creation and baselining of vision statements, "top ten" *high-level* requirements lists, and early analysis of architecturally influential factors, such as load, usability, and internationalization. Further, these methods encourage many skillful analysis techniques during early iterations, such as a series of requirements workshops involving both target users and developers, writing use cases, and much more.

vision boxes p. 282

product sheets p. 284

use cases p. 287

various elicitation methods start on p. 289

Evolutionary and Adaptive Planning

As with evolutionary requirements, with evolutionary and adaptive planning it is not the case that estimates and schedules are forever unbounded or unknown. Yet, due to early requirements change and other factors, there is an initial phase of high uncertainty, which drops as time passes and information accumulates. This has been called the **cone of uncertainty** (Figure 2.5) [McConnell98].

adaptive planning and related tips start on p. 253

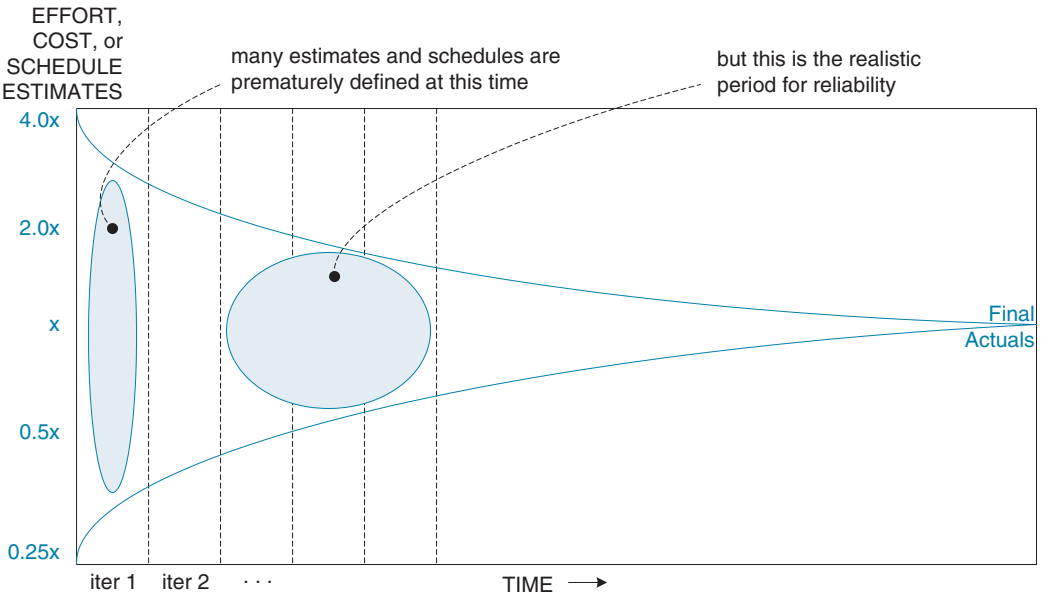
> The iterative response to this uncertainty is to defer an expectation of semi-reliable estimates for cost, effort or schedule until a few iterations have passed. Perhaps 10% to 20% into a project.

This is consistent with management practice in other new product development domains, where an initial exploratory phase is common. Further, the practice of **adaptive planning** is encouraged

adaptive and predictive planning p. 253

rather than **predictive planning**. That is, a detailed schedule is not created beyond a relatively short time horizon, so that the level of detail and commitment is commensurate with the quality of information.

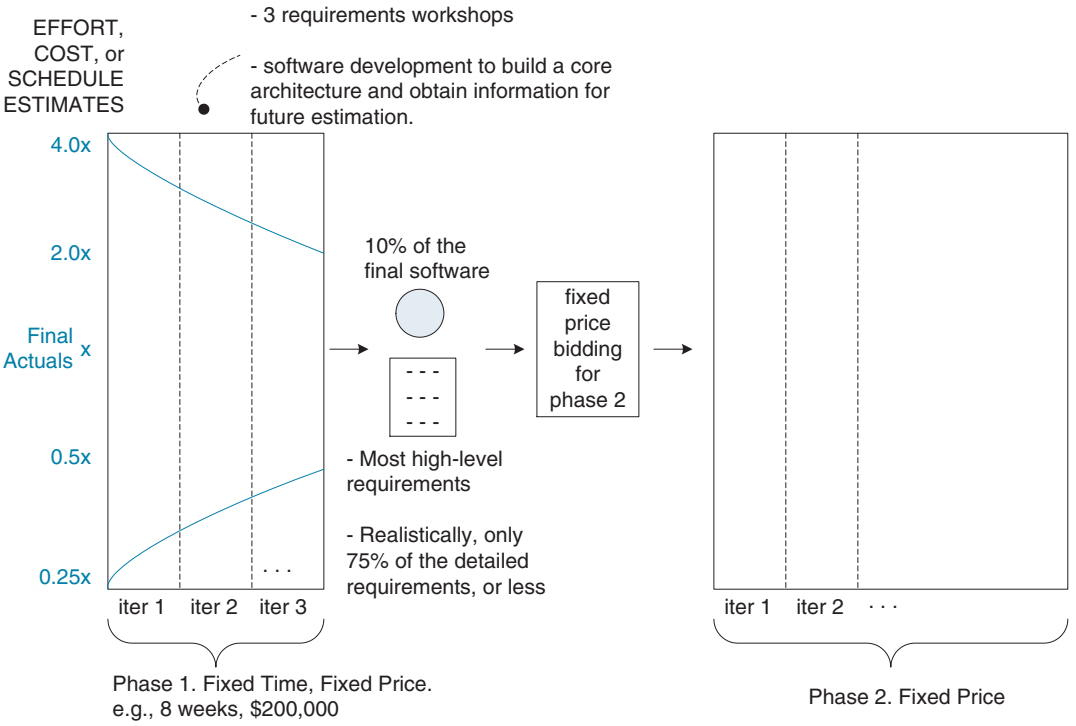
Figure 2.5 cone of uncertainty



Fixed-Price Contracts

With respect to fixed-price bidding and evolutionary estimates, some IID methods (such as the UP) recommend running projects in two contract phases, each of multiple timeboxed iterations.

Figure 2.6 two contract phases



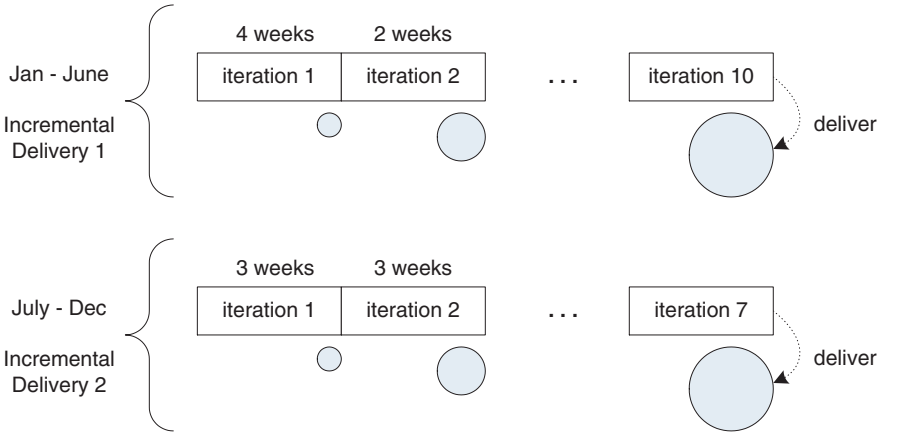
The first phase, a relatively short fixed-time and fixed-price contract, has the goal of completing a few iterations, doing early but partial software development and evolutionary requirements analysis. Note the key point that partial software is produced, not merely documents.

The outputs of phase one—including the software base—are then shared with bidders for a phase two fixed-price contract. The evolutionary refinement of specifications and code in phase one provides higher quality data for phase two estimators, and advances the software for the project (Figure 2.6).

Incremental Delivery

Incremental delivery is the practice of repeatedly delivering a system into production (or the marketplace) in a series of expanding capabilities (Figure 2.7). The practice is promoted by IID and agile methods. Incremental deliveries are often between three and twelve months.

Figure 2.7 incremental delivery with iterations



Incremental delivery is often confused with iterative development. A six-month delivery cycle could be composed of 10 short iterations. The results of each iteration are not delivered to the marketplace, but the results of an incremental delivery are.

Evolutionary Delivery

The Evo method and evolutionary delivery p. 212

Evolutionary delivery is a refinement of the practice of incremental delivery in which there is a vigorous attempt to capture feedback regarding the installed product, and use this to guide the next delivery. Naturally, the evolutionary goal is to best meet

some difficult-to-predict need, such as the most frequently requested new features. Uniquely, the Evo method promotes—when possible—very short evolutionary delivery cycles of one or two weeks, so that each iteration delivers something useful to stakeholders.

To contrast "pure" incremental delivery with evolutionary delivery, in the former a plan is defined of several future deliveries—feedback is not driving the delivery plan. In evolutionary delivery, there is no plan (or at least no fixed plan) of future deliveries; each is dynamically created based on emerging information. In practice, a marriage of some future prediction and feedback is obvious and common, and the two terms are used interchangeably.

The Most Common Mistake?

Iterative and agile process coaches often see scenarios like this:

Jill: Sure, we don't apply the waterfall—everyone knows it doesn't work. We've adopted <iterative method X> and are into our first project. We've been at it for two months and have the use case analysis nearly finished, and the plan and schedule of what we'll be doing in each iteration. After review and approval of the final requirements set and iteration schedule, we'll start programming.

This profound misunderstanding, still superimposing waterfall-inspired, big up-front analysis and planning (predictable manufacturing) values onto iterative methods, is one of the most common mistakes that new iterative and agile method adopters make.

Specific Iterative & Evolutionary Methods

Specific *agile* methods are summarized in the next chapter. This section mentions some *iterative* methods (Evo and UP) that predate most agile methods; they may or may not be considered agile.

Of all the methods mentioned in this book (Scrum, XP, Evo, UP, OPEN, DSDM, ...) the UP or its variation the Rational Unified Process (RUP) is perhaps the most widely used. It is found in thousands or tens of thousands of development organizations worldwide. This does not mean it is well applied or well understood.

Evo

Evo details p. 211

Evo was perhaps the first iterative and evolutionary method, starting in the 1960s. Evo recommends short 1–2 week iterations, and uniquely, evolutionary *delivery* each iteration. Evo adaptively plans iterations by highest value-to-cost ratio, and strongly promotes the unambiguous definition of quality requirements (such as load) with quantified and measurable statements.

Unified Process

UP details p. 173

The UP or RUP, first developed in the mid-1990s, brings together the knowledge of many experienced large-system architects and process-leaders at Rational Corp., and their customers, into a well-defined IID method. One key UP theme is risk-driven development in the early iterations, focusing on creation of the core architecture and driving down the high risks. The UP also includes the definition of common project workproducts, such as the *Vision*, *Software Architecture Document*, and *Risk List*.

Other Methods

In addition to UP and Evo, other IID methods include:

- The **Microsoft Solutions Framework** process, available from Microsoft Education. It is a description of best practices used by Microsoft.
- The **OPEN** process from Henderson-Sellers, Firesmith, and Graham [FH01].
- **WinWin Spiral Model** and **MBASE Spiral Model** from Barry Boehm (creator in the 1980s of the well-known iterative Spiral Model) and colleagues [BEKPSM98], [BP01].

WHAT'S NEXT?

The next chapter summarizes agile method practices and values. After that, a story chapter illustrates these practices with a concrete scenario.

RECOMMENDED READINGS

- *Rapid Development*, by Steve McConnell. Examines variations of iterative development, citing plenty of research data.
- *The Mythical Man-Month*, by Frederick Brooks. The silver-anniversary edition of this classic discusses the advantages of IID, in addition to many timeless lessons.