39

Timebox Development

Timebox Development is a construction-time practice that helps to infuse a development team with a sense of urgency and helps to keep the project's focus on the most important features. The Timebox Development practice produces its schedule savings through redefining the product to fit the schedule rather than redefining the schedule to fit the project. It is most applicable to in-house business software, but it can be adapted for use on specific parts of custom and shrink-wrap projects. The success of timeboxing depends on using it only on appropriate kinds of projects and on management's and end-users' willingness to cut features rather than stretch the schedule.

SUMMARY

Efficacy

Potential reduction from nominal schedule:	Excellent
Improvement in progress visibility:	None
Effect on schedule risk:	Decreased Risk
Chance of first-time success:	Good
Chance of long-term success:	Excellent

Major Risks

- Attempting to timebox unsuitable work products
- Sacrificing quality instead of features

Major interactions and Trade-Offs

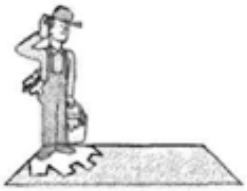
- Depends on the use of Evolutionary Prototyping
- Trades feature-set control for development-time control
- Often combined with JAD, CASE tools, and Evolutionary Prototyping on RAD projects
- Can be combined with Evolutionary Delivery when timing of deliveries matters more than contents

It's amazing how much you can get done the day before you leave for a vacation. You can pick up the dry cleaning, pay the bills, stop the mail and the newspaper, buy new travel clothes, pack, buy film, and drop off a key with the neighbors. Just as amazing are all the things that you don't do the day before you leave. You don't seem to need to spend quite as long in the shower that morning or as long reading the newspaper that night. You might have many other things that you would like to do that day, but suddenly some of those day-to-day priorities slip down a notch.

Timebox Development is a means of harnessing the same sense of urgency that accompanies preparing for a vacation except that it usually accompanies preparing to work hard instead! When you follow the Timebox Development practice, you specify a maximum amount of time that you will spend constructing a software system. You can develop as much as you want or whatever kind of system you want, but you have to constrain that development to a fixed amount of time. This sense of urgency produces several results that support rapid development.

CROSS-REFERENCE
For more on trading off resources and product attributes for schedule gains, see Section 6.6, "Development-Speed Trade-Offs."

It emphasizes the priority of the schedule. By making the schedule *absolutely fixed*, you stress that the schedule is of utmost importance. The time limit, or "timebox," is so important that it overrides all other considerations. If the project scope conflicts with the time limit, you reduce the scope to fit the time limit. The time limit itself is not allowed to change.



CLASSIC MISTAKE

It avoids the 90-90 problem. Many projects get to the point where they are 90 percent complete and then stay at that point for months or even years. Many projects spend an undue amount of time in sinkholes that don't move the project forward but that consume huge amounts of resources. You build a small version first, and you build it quickly so that you can get on to version 2. Rather than building a feature-rich first version, it's often more efficient to get a basic version working, learn from the experience, and build a second version after that.

It clarifies feature priorities. Projects can expend a disproportionate amount of time quibbling about features that make little difference in a product's utility. "Should we spend an extra 4 weeks implementing full-color print-preview, or is black-and-white good enough?. Should the 3D sculpting on our buttons be one pixel wide or two? Should our code editor reopen text files in the exact location they were last used or at the top of the file?" Rather than spending time arguing about whether to include features of "very low' priority or only "moderately low priority," tight time constraints focus attention on the top end of the priority list.

CROSS-REFERENCE
For details on the way that gold-plating can occur unintentionally, see "Unclear or Impossible Goals" in Section 14.2.

It limits developer gold-plating. Within the bounds of what was specified, you can often implement a particular feature in several ways. There is often a 2-day. 2-week, and 2-month version of the same feature. In the absence of the clarifying presence of a development timebox. developers are left to choose an implementation based on their own goals for quality, usability, or level of interest in the feature's design and implementation. Timeboxing makes it clear that if there is a 2-day version of a feature, that is what you want.

It controls feature creep. Feature creep is generally a function of time and averages about 1 percent per month on most projects (Jones 1994). Timebox Development helps to control feature creep in two ways. First, if you shorten the development cycle, you reduce the amount of time people have to lobby for new features. Second, some feature creep on long projects arises from changing market conditions or changes in the operational environment in which the computer system will be deployed. By cutting development time, you reduce the amount that the market or the operational environment can change and thus the need for corresponding changes in your software.

CROSS-REFERENCE
For more on motivation, see Chapter 11, "Motivation."

It helps to motivate developers and end-users. People like to feel that the work they're doing is important, and a sense of urgency can contribute to that feeling of importance. A sense of urgency can be a strong motivator.

39.1 Using Timebox Development

Timebox Development is a construction-phase practice. Developers implement the most-essential features first and the less-essential features as rime permits. The system grows like an onion with the essential features at the core and the other features in the outer layers. Figure 39-1 on the next page illustrates the timebox process.

CROSS-REFERENCE
For details on this kind ot prototyping, see Chapter 21, "Evolutionary Prototyping."

Construction in Timebox Development consists of developing a prototype and evolving it into the final working system. Timeboxing usually includes significant end-user involvement and ongoing reviews of the developing system.

Timeboxes usually last from 60 to 120 days. Shorter periods are usually not sufficient to develop significant systems. Longer periods do not create the sense of urgency that creates the tirnebox's clear focus. Projects that are too big for development in a 120-day timebox can sometimes be divided into multiple timebox projects, each of which can be developed vviihin w to 120 davs.

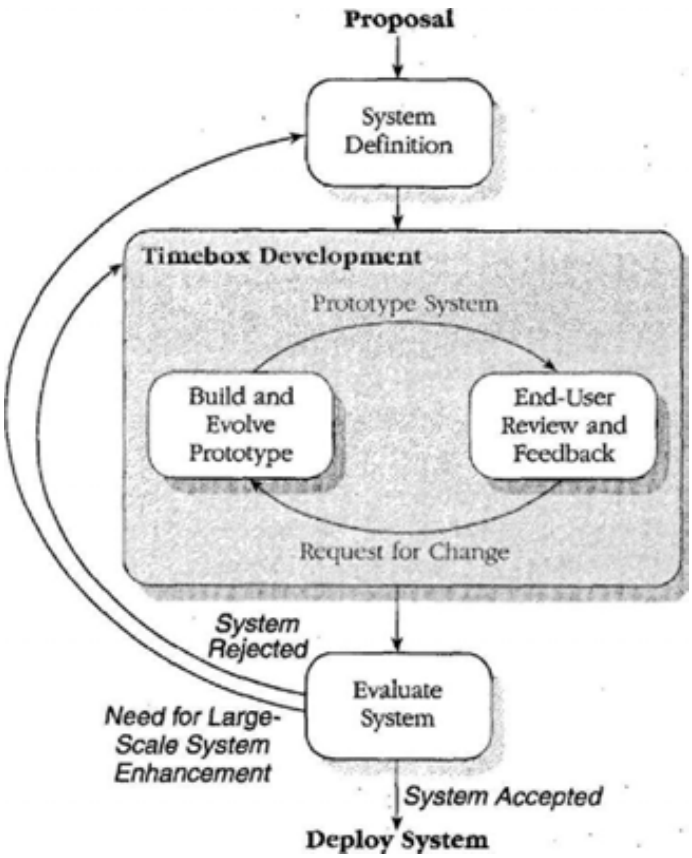


Figure 39-1. *Timebox Development cycle. Timebox Development consists of constructing and evolving an Evolutionary Prototype with frequent end-user interaction,*

After the construction phase, the system is evaluated and you choose from three options:

- Accept the system and put it into production.
- Reject the system because of a construction failure. It might have insufficient quality, or the development team might not have been able to implement the minimum amount of functionality needed for the core system. If that happens, the organization can launch a new Timebox Development effort.
- Reject the system because it does not meet the needs of the organization that built it. A perfectly legitimate outcome of a timebox development is for the team to develop the core system that was identified before Timebox Development began, but for end-users to conclude that the system is not what they wanted. In that case, work begins anew at the system-definition stage, as-shown in Figure 39-1.

The people who evaluate the system include the executive sponsor, one or more key users, and a QA or maintenance representative. Technical support and auditing personnel can also be involved in the evaluation.

Regardless of the outcome, it is critical to the long-term success of timeboxing that the timebox not be extended. The end-date for the limebox is not a *due date*. It's a *deadline*. It needs to be clear to the timebox team that whatever they have completed at the end of the timebox is what will be either put into operation or rejected. If the organization has a history of extending its timebox deadlines, developers won't take the deadline seriously and the practice will lose much of its value.

Entrance Criteria for Timebox Development

Timeboxing is not suited for all kinds of development. Here are some guidelines you can use to be sure it is suitable for your project.

Prioritized list of features. Before timebox construction can begin, the functions and design framework of the system need to be defined. The end-users or customers need to have prioritized the system s features so that developers know which are essential and which are optional. They should have defined a minimal core feature set that you are sure you can implement within the timebox time frame. If this prioritization cannot be done, the system is not well-suited to timebox development.

CROSS-REFERENCE
For more on motivation and setting realistic goals, see "Goal Setting" in Section 11.2 and: Section 43.1, "Using Voluntary Overtime."

Realistic schedule estimate created by the timebox team. An estimate for the timebox construction should be created by the development team. The construction team needs to estimate both how much time they need (usually 60 to 120 days) and how much functionality they think they can implement within that period. From a motivation point of view, it is essential that the team create its own estimate. Timeboxing is an ambitious practice, and it won't succeed if developers are simply presented with an impossible combination of schedule and functionality goals.

CROSS-REFERENCE
For more on languages that support rapid code generation, see Chapter 31, "Rapid-Development Languages (RDLs)."

Right kind of project. Timeboxing is best suited for in-house business software (IS software). Timeboxing is an evolutionary prototyping practice and should be built with rapid-development languages, CASE tools, or other tools that support extremely rapid code generation. Highly custom **applications** that require hand-coding are usually not appropriate projects for timebox development. Before beginning a timebox project, verify that you can build the project with the available tool set and staff.

Sufficient end-user involvement. As with other prototyping-based activities, the success of Timebox Development depends on good feedback from end-users. If you can't get adequate end-user involvement, don r use a limebox,

The Timebox Team

CROSS-REFERENCE
For more on effective teamwork, see Chapter 12, 'Teamwork.'

A timebox-development team can consist of from one to five people. The full "timebox team" also includes end-users who have been designated to assist the construction team. These end-users are often dedicated full-time to their role of supporting the construction team.

The timebox team needs to be skilled in developing systems with the rapid-development software that will be used. There is no time to learn new software on a timebox project.

The timebox team needs to be highly motivated. The urgency created by the timebox development practice itself will provide some of the motivation. The ability to achieve a level of productivity rare within the organization should provide the rest.

Although my understanding of developer motivation makes me wary of his advice, James Martin recommends that motivation on a timebox project also include the following (Martin 1991):

- Tell developers that they will be judged by whether they create a system that is in fact accepted. Point out that most timebox efforts succeed and that they shouldn't distinguish themselves by being one of the rare failures.
- Tell developers that success will be rewarded and that their efforts are visible to upper management. Follow through on the rewards.
- Tell developers that if they succeed, you'll' hold a major victory celebration. Follow through on the celebration.

Don't use Martin's advice verbatim without first thinking through how the issues discussed in Chapter 11, "Motivation," apply within your environment.

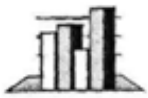
Variations on Timebox Development

Timeboxing is usually applied to the design and construction phase of entire business systems. It is generally not well-suited to the development of shrink-wrap software products because of the long development times needed. But timeboxing can be quite useful as a strategy for developing parts of software systems—live user-interface prototypes or throwaway prototypes on a shrink-wrap software project. Timeboxes for prototypes are much shorter than the time recommended for information systems, perhaps on the order of 6 to 12 days rather than 60 to 120. The development team will have to define a timebox that makes sense for the specific prototype they're building.

You can use timeboxing on a variety of implementation activities—software construction, help-screen generation, user-documentation, throwaway prototypes, training-course development, and so on.

39.2 Managing the Risks of Timebox Development

Here are some of the problems with timeboxing.



HARD DATA

Attempting to timebox unsuitable work products. I don't recommend using timeboxes for upstream activities (or beginning-of-the-food-chain activities) such as project planning, requirements analysis, or design—because work on those activities has large downstream implications. A $100 mistake in requirements analysis can cost as much as $20,000 to correct later (Boehm and Papaccio 1988). The software-project graveyard is filled with the bones of project managers who tried to shorten upstream activities and wound up delivering software late because small upstream defects produced large downstream costs. Time "saved" early in the project is usually a false economy.

FURTHER HEADING
For a different point of view on timeboxing upstream activities, *see* "Timeboxing" (ZahniseM995).

Timeboxing is effective on activities at the end of the development food-chain because the penalty for poor-quality work is limited to throwing away the timebox work and doing it over. Other work isn't affected.

CROSS-REFERENCE
For more on the effects of conflicting goals, see "Goal Setting" in Section 11.2. For more on the effects of low quality, see Section 4.3, "Quality-Assurance Fundamentals."

Sacrificing quality instead of features. If your customer isn't committed to the timebox practice of cutting features instead of quality, don't use a timebox. Developers have a hard time meeting conflicting goals, and if the customer insists on a tight schedule, high quality, and lots of features, developers won't be able to meet all three objectives at once. Quality will suffer.

Once quality begins to suffer, the schedule will suffer too. The team will produce feature-complete software by the timebox deadline, but the quality will be so poor that it will take several more weeks to bring the product up to an acceptable level of quality.

With true timeboxing, the software is either accepted or thrown away at the timebox deadline. That makes it clear that the quality level must be acceptable at all times. The success of timeboxing depends on being able to meet tight schedules by limiting the product's scope, not its quality.

39.3 Side Effects of Timebox Development

Timebox Development's influence is limited to .shortening development schedules. It does not typically have any influence—positive or negative—on product quality, usability, functionality, or other product attributes.

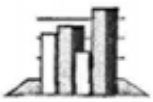
39.4 Timebox Development's Interactions with Other Practices

Timebox Development is a specific kind of design-to-schedule practice (Section 7.7). It is an essential part of RAD, which means that it is often combined with JAD (Chapter 24), CASE tools, and Evolutionary Prototyping (Chapter 21). Because Timebox Development calls for an unusual degree of commitment on the part of the development team, it is also important that each team member be Signed Up (Chapter 34) for the project.

FURTHER READING
The milestone process that Microsoft uses could be considered to be a modified timebox approach. For details, see *Microsoft Secrets* (Cusumano and Seiby 1995).

Timeboxes can also be combined with Evolutionary Delivery (Chapter 20) if you need to define each delivery cycle more by the time you complete it than by the exact functionality you deliver. Similarly, shrink-wrap and other kinds of projects can use timeboxes as part of a staged, internal-delivery approach. Delivering the software at well-defined intervals helps to track the progress and quality of the evolving product. Most projects that use timeboxes in this way won't be willing to throw away work that isn't completed by the deadline, so they -won't be using pure timeboxes. But they can still realize some of timebox development's motivational, prioritization, and feature-creep benefits.

39.5 The Bottom Line on Timebox Development



HARD DATA

Timebox Development has been found to produce extraordinary productivity at DuPont, where it was initially developed. DuPont averages about 80 function points per person month with timeboxing, compared to *15* to 25 with other methodologies (Martin 1991). Moreover, timebox development entails little risk. System evaluation and possible rejection is an explicit part of the practice, but after its first few years of use, DuPont had not rejected a single system developed with timeboxing. Scott Shultz, who created the methodology at DuPont, says that "[a]ll applications were completed in less time than it would have taken just to write the specifications for [an application in] Cobol or Fortran" (Shultz in Martin 1991).

39.6 Keys to Success in Using Timebox Development

Here are the keys to success in using timeboxing:

- Use timeboxing only with projects that you can complete within the timebox time frame (usually from 60 to 120 days).
- Be sure that end-users and management have agreed to a core feature set that the timebox construction team believes it can implement within the timebox time frame. Be sure that features have been prioritized, and that you can drop some of them from the product if needed to meet the schedule.
- Be sure that the timebox team is signed up for the ambitious timebox project. Provide any motivational support needed.
- Keep the quality of the software high throughout the timebox.
- If you need to, cut features to make the timebox deadline. Don't extend a timebox deadline.

Further Reading

Martin, James. *Rapid Application Development.* New York: Macmillan Publishing Company, 1991. Chapter 11 discusses timebox development specifically. The rest of the book explains the *RAD* context within which Martin suggests using timeboxes.