A PROJECT PLANNING AND DEVELOPMENT PROCESS FOR SMALL TEAMS

Marc Rettig and Gary Simons

A good manager is a remover of obstacles and a provider of resources.

This phrase circulates through management circles in various versions. It normally conjures up visions of managers resolving crises during projects, providing people and equipment as they are needed. But the work of removing obstacles and providing resources should start before a project even begins. A good manager can remove many obstacles before they appear by providing one of the most fundamental resources any project can have: a clearly defined plan and process definition. The plan tells team members what needs to be done and when, while the process definition tells them how it should be done and how to know when it has been done with adequate quality. This leads to another aphorism: "A good manager makes sure that all team members know what they are supposed to do, and how to tell whether they have done a good job."

With a good process definition in hand, the project manager becomes a shepherd or facilitator, seeing to it that the team has what it needs and ensuring that the project is progressing as it should. This article documents a process for software development designed and successfully used by the Academic Computing Department of the Summer Institute of Linguistics (SIL) in Dallas, Texas. Standing at the start of a complex five-year development project, the department realized that seat-of-the-pants management techniques would not be successful. We decided to organize ourselves as a "structured open team," and to adopt some of the techniques promoted by the total quality movement.

Organization

The goal of this article is to describe our process in enough detail that readers can borrow from it to define or improve their own development process. The process embodies a few new ideas, a few twists on old ideas, and many time-honored and often-repeated ideas. In the interest of completeness we have described everything, even the old notions, recognizing that many project managers have never had any management training.

The bulk of this article describes the development process outlined in the sidebar, addressing both the 'hard' technical issues and the 'soft' social aspects of our work. Limits in space and the reader's tolerance for tedium prevent us from elaborating on every point. Since the headings in the article correspond to points in the outline, it should be easy to keep track of the discussion's context. The process description is preceded by background on our project, to give context, and by a description of our team organization.[1]

This article presents a snapshot of an evolving method as it is used at SIL. This is not a formal method, and it is not guaranteed to be complete and coherent. Readers should feel free to borrow and adjust wherever they see fit without offending the methodology gods.

The Academic Computing Department at SIL is creating an object-oriented software development environment called CELLAR [15], which facilitates construction of applications that deal with large collections of structured multilingual information. This is R&D work: "research" in that we do not always know how to accomplish our goals, "development" in that we are committed to delivering quality software to our user community.

Now that CELLAR is nearing completion, we are using it to prototype a "Performance Shell"—a tool for building performance support systems [11, 14]. Other departments in our organization are forming teams to build CELLAR applications. These efforts expanded our focus to include applications development in addition to systems development, motivating many revisions in our development process. This article describes the result: a process designed to work for many kinds of projects and many kinds of people—so far we are pleased with its success.

Project Teams and Guidance Teams

Our current arrangement of a project team, team leader, and guidance team follows the approach suggested in Peter Scholtes's *Team Handbook* [16]. Every project is carried out by a *project team* under the watchful eye and helpful hands of a *guidance team.*

The guidance team. Each project team is initially formed by, and then remains responsible to, a *guidance team.* The guidance team is made up of at least three managers who must sign off on the results produced by the project team, and who have the clout to remove obstacles and provide resources at the corporate level. They do not manage the project team—that is the role of the team leader. Rather,

they oversee the work, remove obstacles, and serve as an interface between the project team and the result of the organization. Thanks to the guidance team, the project team can forget about corporate politics and concentrate on their work.

The guidance team has the following responsibilities:

- Identify project goals
- Prepare a mission statement (by writing a first draft of the project brief—see the following)
- Select and assign the project team members
- Determine other resources needed by the project team, and provide them
- Monitor the progress of the project team
- "Sign off" on the results produced by the project team
- Provide accountability to the rest of the organization for the work of the project team

The project team. This team is made up of the people who will do the work. Our project teams were originally inspired by Larry Constantine's notion of "structured open teams" [4, 12]—groups of at least three people acting as peers and making decisions by consensus. As the team matured and people adopted roles befitting their personal strengths, we evolved aspects of Fred Brooks's "surgical team" organization [2]. That is, the lead designer spent less time coding and more time designing, as other people learned enough to take over the coding burden and support the designer in building his vision.

The Project-Level Process

Establish the Project

As the outline in the "PADRE Project Planning and Development Process" sidebar shows, "establishing" a project means forming teams and writing clear goals. The latter takes the form of a "project brief," a short document that sets forth the charter for the project team. Of course, it may be necessary to write separate requirements documents that lead to and support the project brief.

Requirements specification. In our current projects, and hence in the process described in this article, we have chosen to evolve requirements rather than attempt to complete a formal requirements statement at the start. We believe that requirements for large, complex systems are best evolved through an iterative process. Eric Raymond explains this in *The New Hacker's Dictionary*:

> **creationism** n. The (false) belief that large, innovative designs can be completely specified in advance and then painlessly magicked out of the void by the normal efforts of a team of normally talented programmers. In fact, experience has shown repeatedly that good designs arise only from evolutionary, exploratory interaction between one (or at most a handful of) exceptionally able designer(s) and an active user population—and that the first try at a big new idea is always wrong. Unfortunately, because these truths don't fit the planning models beloved of management, they are generally ignored.

In our project, end users motivate the effort, but they have never seen anything comparable to what we are building. Until we had a working prototype—the only specification that communicates adequately—it was difficult for us to teach the language of our innovation to others and begin incorporating their ideas into subsequent iterations of the specification.[2] Rapid prototyping tools let us get user input early and often, "growing" our software in a series of stages. Each stage produces a working product, each makes use of the one before.

The decision whether or not to work from careful specifications is a strategic decision that should be made early in the project. And it should be made *consciously*. Many projects write (or fail to write) specifications simply out of the habit of following the same creationist (or evolutionary) management model for every project, regardless of its nature.

Project brief. So, projects in our organization typically start with an informal but detailed statement of known requirements, summarized in a project brief.

The brief has four parts:

Project title. The name the entire organization will use to describe the project.

Purpose. A concise statement of the specific goals of the project. This serves as a one- or two-paragraph requirements statement for the project stated in measurable terms. The purpose statement may refer to a separate user requirements document for a fuller description.

Stages. A numbered list of the development stages the project will go through on its way to completion. The stage descriptions should include a schedule of completion dates. A detailed discussion of stages appears later in this article.

Team. The people assigned to the project team, with role assignments.

Form a project team. The first several weeks of our team's existence were dedicated to planning and process designing.[3] The project leader (Gary Simons) was assisted in this work by Dave O'Bar, a successful total quality manager who had recently come to SIL from the aerospace industry. This pairing of an experienced software engineer with an experienced manager was a good combination for facing the task of figuring out how to manage a large project. They drafted a process, then put it through a series of refinements: the team made suggestions on paper, we had meetings to discuss changes, and finally, agreed we could all work together in the manner described by the process.[4] In fact, everyone seemed excited to see how it would work.

The results of this early work included much more than the kind of description included in this article.

---

[1] *Principles*
Where the article focuses on process, these comments serve to extract the principles that underlie the recommendations in the accompanying text. Some principles appear because they motivated features of the process. Others appear because they describe important lessons we have learned along the way.

[2] When a system seems too complex to specify, build a rapid prototype [3].

[3] Teams should create standard development processes for themselves, and describe them in detail. This is part of what it means to form a team.

[4] Include the entire team in creating the development process, thus building a communal sense of ownership for both the project and the process.

We worked together to produce detailed standards for coding, testing, and configuration management; we agreed on meeting conventions, created metrics gathering forms and time sheets; and we set up a group archive with an electronic index. This collection of forms, procedures, conventions, and unwritten guidelines developed over the next few months (and which are constantly evolving to accommodate new people and new situations)—is the full development process, of which this article describes only an essential shell.[5] These standards relate to the adage in our introduction: they tell people how to work, and they define in specifics the meaning of 'quality.'

Roles in the project team. The particular roles filled by project team members vary according to the project. A systems development project might involve a lead designer and two programmers. We have begun to involve people from user departments in application teams, and are having encouraging results with assigning explicit roles to members of these "mixed teams." At a minimum,

[5]If your process isn't changing, if it isn't a subject of discussion and debate, it probably isn't being used. A good process is organic, embodied in the habits and conversations of the team. Like any behavior, you can document it, you can do your best to guide its development, but attempts to enforce it by strict mandate are more likely to encourage rebellion than participation.

The PADRE Project Planning and Development Process

Project-level process

1. Establish the project
   - The organization
     - decides to do the project
     - forms a guidance team
   - The guidance team
     - writes a project brief
     - forms a project team
2. Plan the project
   - The project team
     - breaks the project into stages
     - establishes standards and procedures for doing quality work
     - gets approval for plans from guidance team (revising as necessary)
3. Do the project
   - The project team
     - follows the stage-level process for each stage in the project plan
   - The guidance team
     - removes obstacles and provides resources as needed
4. Evaluate the project
   - Both the guidance team and the project team look for ways to
     - improve the project's product
     - improve the project's plan
     - improve the project's process

Stage-level process

1. Plan the stage
   - The project team
     - breaks the stage into modules
     - assigns each module to a team member
     - gets approval for plans from the guidance team (revising as necessary)
2. Do the stage
   - The project team
     - follows the module-level process for each module in the stage plan
     - revises product of stage following review from customers
   - The team leader
     - removes obstacles and provides resources as needed
     - keeps track of progress on the stage progress chart
3. Evaluate the stage
   - The project team looks for ways to
     - improve the stage's product
     - improve the stage's plan
     - improve the project's process

Module-level process

1. Plan the module
   - The programmer (or the lead designer)
     - develops a detailed implementation design
     - develops a detailed testing design
     - gets approval for design from the project team (revising as necessary)
2. Do the module
   - The programmer
     - follows the detailed design to code the module
     - keeps track of progress on the module progress chart
     - revises implementation following review by project team
   - The team leader
     - removes obstacles and provides resources as needed
3. Evaluate the module
   - The team leader and the programmer look for ways to
     - improve the module's product
     - improve the stage's plan
     - improve the project's process

PADRE and PDCA

At each level, the process involves steps called 'Plan,' 'Do,' and 'Evaluate.' Each level also includes external *approval* for the results of the planning step, and iterative *review* and *revision* of the results of the 'Do' step. Putting it all together gives a five-letter acronym, PADRE.

PADRE is similar to the total quality movement's PDCA (Plan-Do-Check-Act) cycle, originally applied to manufacturing situations [1]. In our experience PADRE better expresses the development cycle in a team-based approach, where what is planned by a few should be approved by the rest of the team, and what is done by a few should be reviewed by the team.

Note

Where this outline uses 'programmer' and 'code,' it could just as well substitute other words suitable to other kinds of modules, such as 'documentor' and 'write' or 'interface specialist' and 'prototype.'

Tackling Complexity With an Iterative Strategy

Complex systems are difficult to specify. You never get them right the first time, and you rarely get them right the second time. Requirements for such systems notoriously change throughout development.

One way to deal with these troublesome facts of life is to adopt an iterative strategy for software development. We have adopted just such a strategy, bringing five weapons to bear on the problem of software complexity.

- ***Divide and conquer.*** We plan in minute detail, breaking the project into two-week chunks of work, which we run through a mini product life cycle.
- ***A spiral life cycle*** (see Figure A). A fundamental strategy that we seek to follow in our project is to have every implementation module result in a working system with added functionality. Rather than performing a large testing and integration step at the end of each stage, we include testing and integration at the end of each two-week module. The implementer includes the new code into the base code and makes the new version available to the rest of the team. At each level of planning we set aside time for user or peer review.
- ***Object technology,*** which we believe lends itself well to iterative development. In our experience, object-oriented analysis, design, and programming facilitate the strategies described in the following two bullet points. In particular, we are using Smalltalk.
- ***Rapid prototyping tools.*** Software such as HyperCard and Authorware lets us quickly mock up several alternative designs. Prototyping and user testing helps designers express their ideas to users, and gives users something concrete to respond to.
- ***Bite-sized incremental testing*** [13]. Each module is fully tested before the results are released to the team: programmers write automated test scripts which exercise their new code, then execute *all* the scripts for the entire system. Since this usually involves no more than a day or two spent writing test procedures, and since test plans and test code are both subjected to peer review, we believe people are more likely to do a thorough job than they would in an extended test phase. Since every module includes a validation of the entire system, we catch many side effects of new code before they are released to the rest of the team.

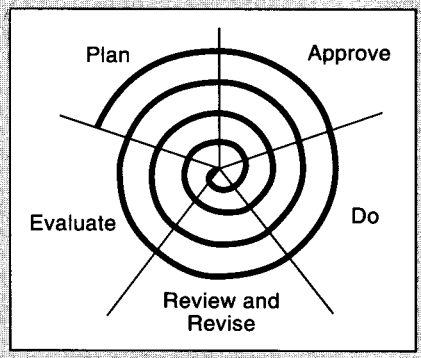

Figure A. A spiral life cycle

an application project team consists of three people uniquely filling the following three roles:

Implementer. A programmer who writes the program code that implements the project

Domain specialist. A specialist in the application domain, who need not know much at all about programming, who serves to specify requirements and review results

Technical reviewer. A programmer other than the implementer who serves to review technical aspects of the implementation whenever needed—a source of design ideas, consultation, and peer review

A large project team of five to seven members could have two or three people designated for each role. Even though a single person might have the skills to fill more than one of the preceding roles, this person should be identified as filling a single role on the team.[6]

We assume the three core team members are resident in the same location and are available to meet together once a week. Many teams, however, will be well served by including members who are at different locations and not available to participate in regular team meetings. These members are designated as filling the role of:

Consultant. A specialist in the application domain who participates in the team from a distance, or intermittently at the request of the core team.

Unlike the potentially numerous reviewers who will evaluate the work of the team at the end of each stage (described later), a consultant is a full member of the team who is kept informed of the activity during each stage, and has the opportunity to review the team's work as it is developing. It is up to each team to work out how they will incorporate consultants into their work.

Social roles. The roles just described are all technical roles. Constantine [4, 8] and many others describe another set of roles necessary

[6] Explicitly designate the role of each member of the team, and make sure that each team has a person designated for each key role.

for effective communication in a project team, noting that the social interaction within a team is a commonly overlooked cause of trouble for a project. Early in our team's formation, we explicitly assigned roles such as facilitator, critic, and scribe at the beginning of each meeting, rotating the assignments among the team members [12].[7] With time, these roles either become habits or do not, and role assignments now tend to be less formal: "I see Larry is taking notes—will you send us all a copy?" Still, we believe attention to interpersonal communication was an important ingredient in the early success of our team.

The team leader. Each project team has one member designated as the *team leader*. This could be one of the persons already slotted for one of the three positions previously mentioned, it could also be someone appointed solely to fill the position of leader. The role of team leader is not synonymous with guiding light or chief decision maker; a consensus approach to decision making should be followed within the teams [5–7, 16]. Rather, being team leader means taking on the following responsibilities:

Facilitator. Schedule and lead team meetings, or nominate another team member to do so.
Archivist. Take minutes of team meetings and maintain an archive of project documents, or nominate another team member to do so [8, 12].
Manager. Maintain a chart of progress toward project milestones to give the project team feedback on progress toward goals, to assist the project team in planning its activities, and to provide accountability to the guidance team and others outside the project team.
Contact. Serve as the contact point between the project team and the guidance team.

The day-to-day details of a project leader's life (and that of all the team members) will vary depending on individual strengths and personalities. The roles we have described are essential. Their assignment to a particular person on the team may vary (with the exception of the "contact" role).

Plan the Project. Planning right—taking time to do a thorough job before jumping into the work—makes the rest of project management far more effective and pleasant. Therefore, the rest of this article is almost entirely about planning, saying very little about the daily life and responsibilities of a project manager.

Divide the project into stages. A project is usually too big to plan in detail all at once, so planning progresses through a series of top-down steps. Complex projects may involve several semiautonomous components, which may be planned independently. If the work on any component is expected to take a long time, it should be divided into *stages*. Each stage lasts four to six months (adjust to suit the situation), and each produces a working subset of the project's results.

A stage is defined in terms of the resulting behavior and appearance of its product, and the information structures that underlie it. The project team plans its work by identifying a succession of stages from most basic to most advanced (in other words, the team performs requirements analysis and general system design at this point in the process). If a project involves a number of tools that work with the same data, each could be relegated to a separate stage. If implementing all the features in a tool seems too big a task for one stage, divide the functionality into subsets and assign each set to a separate stage.

Stages go through phases. Each component goes through its own software life cycle. We use the following four generic *phases*:

Mockup. Build a quick-and-dirty mockup of the application to get quick feedback from potential end users. Our team is using Macromedia's *Authorware* for this purpose. Other people recommend anything from overhead transparencies to HyperCard applications.
Prototype. The mockup (with changes called for by reviewers) becomes a functional specification for a team of programmers who develop a fully functional prototype.
Refinement. The prototype is successively refined as end users evaluate it and suggest changes.
Maintenance. After the refined program has been officially released as version 1.0, it is maintained in response to user requests for service.

Other systems of life cycle phases could just as well be used. For some components—documentation, for example—'prototype' will mean something very different than it would for program code. We typically skip the mockup phase for stages that produce internal system enhancements. Your life cycle should suit the nature of your efforts.

In our case, since we are iteratively developing a large system with test releases every six months, the maintenance phase is just rolled into the next stage. That is, bug reports from users become part of the requirements for the next stage, and there is no specific planning or staffing for maintenance. This is likely to change when version 1 of the software ships to the field, a fact that illustrates the notion of process refinement. Since the process is a living, working document, we are free to change it to accommodate new circumstances.

At this point, the team has decomposed the project by function and by time. They can create the axes of the planning chart shown in Figure 1, a framework for the next level of detail. The chart accompanies the project brief, which describes the functional boundaries of each component, and assigns beginning and ending dates to the stages. Complex projects may require more detailed descriptions of the functional decomposition. (In practice, things don't *always* divide tidily into components. We sometimes define stages to implement a set of enhancements or user requirements, possibly scattered across several components.)

The Stage-Level Process

Plan the Stage

Our entire process is based on the observation that small projects are easier to manage than big projects. Planning proceeds top-down until the work has been divided into many

[7] Build effective communication into your process—train team members in team dynamics and effective meeting techniques.

small pieces called 'modules.' Each team member uses the module-level process definition (described later) to manage the progress of his or her current module, while the manager guides the progress of the whole series of modules toward the final goal.[8]

We have experienced many benefits to this approach, among them:

- Plans are more accurate than they were before we started working this way. It is easier to estimate how much one person can accomplish in two weeks than it is to estimate how much five people can do in six months.
- It is easier to track progress. A programmer who is 5 days through a 10-day module can easily tell if the schedule is slipping and either apply more effort or notify the manager so dependent modules can be rescheduled. On any day you can tell at a glance where the project stands. This is in strong contrast to the uncomfortable mysteries of tracking a monolithic 12-month project.
- We can often resolve problems before they affect the whole project. Unexpected problems are sure to arise, and questions like, "What was the cause, exactly?" and, "How long will the delay be?" are notoriously difficult to answer for long projects. But a dependency chart of small modules can quickly reveal the long-term effect of a delay in any single module and help managers plan how to adjust.
- Developers get more "bones," "biscuits," "herring," "six-packs," . . . . That is, it's nice for everyone involved to see the progress they are making. Every couple of weeks or so, each person gets the satisfaction of completing some useful bit of work and releasing it to the rest of the team. This builds momentum and morale, two things that can suffer over the long haul.

At the end of each stage (every four to six months in our department), the project team summarizes the results of the just completed stage, plans the details for the next stage, and publishes both in a written status report for the guidance team.[9] The detailed plan is the most difficult part of this report, since although they know the major deadlines for the next stage, the team still has to divide the work into "bite-sized chunks" they can confidently assign and monitor.

Producing the detailed plan involves breaking the stage into *modules*. These are not "modules" in the sense of subprograms, but simply "modules" in the sense of "units of work." A module should be no more than about two weeks' work for one person. In our object-oriented setting, typical modules might require implementing a single object class, implementing the application window for a tool, or writing a section of documentation.

In our case, the job of planning a stage falls to the designer with the most experience in whatever component the stage involves. He or she makes an inventory of all the work that must be done to accomplish the goal for the stage. Additional modules may come from user testing, bug-report forms, or work deferred from previous stages. That is, planning takes into account scheduled work, surprises left over from earlier stages, and predictable maintenance tasks whose substance is unknowable in advance.

The team members most qualified to do so grade each module's difficulty as "hard," "medium," or "easy," as a guideline for estimating the time required to complete them. As an additional and more concrete measure, the designers estimate the key complexity factors that affect productivity. In our Smalltalk project, they estimate the number of classes and methods that will need to be created or modified in the module.[10] If it looks as if it will take longer than two weeks, we try to break it down into smaller modules, with the schedule taking into account dependencies between tasks.

The result of this work is given to the project leader or lead designer, who uses project management software to assign and schedule each module. This gives a detailed plan for the stage, consisting of a PERT chart showing the dependencies between the modules, and a Gantt chart showing the schedule for each team member's assignments during the stage. Attached to this, a description of each module specifies the work and often includes preliminary design notes.

This plan is circulated among the project team for review, modification, and final acceptance, and a copy is given to the guidance team for the same purpose. Only then does "real work" begin.[11]

The Module-Level Process

Many of the principles mentioned in this article are incorporated into our process at the micromanagement level—the process each programmer follows to develop a module. Each module goes through a minilife cycle with five *milestones*. The actual interpretation of the milestones may differ with the phase and type of module, but they essentially describe a plan-do-refine approach punctuated by peer reviews of results. In general terms, the five milestones are as follows:

Plan. This milestone is passed when some member of the project team has prepared a plan for implementing and testing the module, and submitted the plan for review by other members of the project team (the

[8]Divide and conquer—break projects into lots of bite-sized chunks. "To make a long journey, take lots of small steps."

[9]Produce a written status report at the end of each stage, however pressed for time you might be. This forces you to think about what just happened, and to plan the next stage carefully before you start working. Publishing the report for your peers and superiors is great for corporate communication.

[10]A few metrics are better than none, and almost as good as a lot of metrics. The trick is to measure the right things and measure them consistently. Early in the project we kept a lot of statistics so we could learn to estimate the number of classes and methods needed for a module, and to convert those numbers to effort estimates. Later in the project, when modules have been way off schedule we have run the same metrics in the *post mortem*, usually finding that programmer productivity rates have remained constant; we just grossly underestimated complexity.

[11]*Everyone* agrees on a plan before *anyone* begins. Our team has decided to make decisions by consensus, which means we work together to find mutually acceptable solutions before implementing any plans. This isn't always easy, but group consensus-building techniques help [5–7, 16].

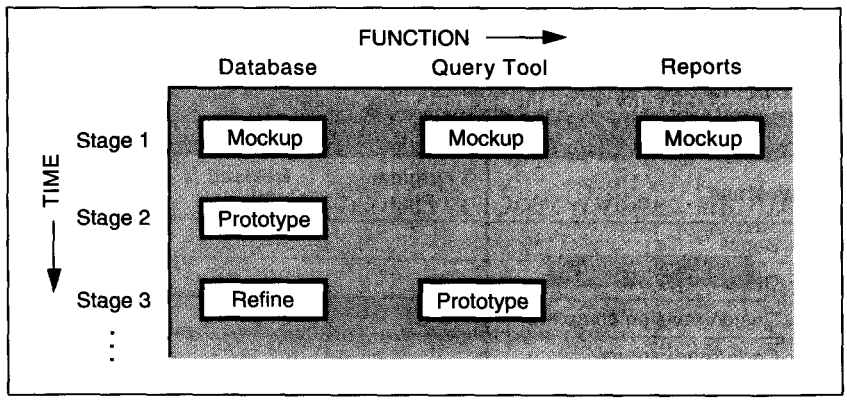

Figure 1. The top part of a sample project overview. The horizontal axis shows how the work has been divided into functional components. The vertical axis shows how the work has been divided into temporal stages. The boxes show how each component's life cycle phases map onto the temporal stages.

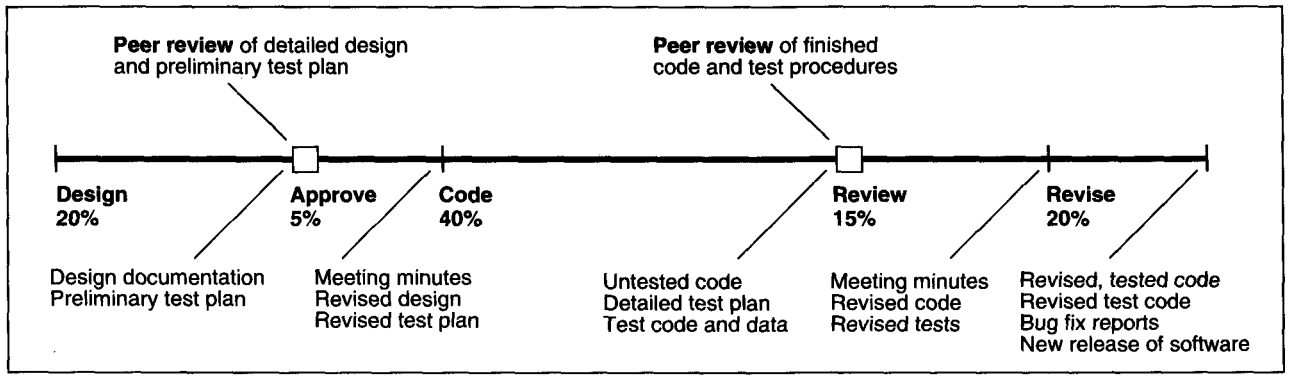

Figure 2. Once around the spiral. This represents the work typically spent in completing a single coding module, including percentages of total effort for each milestone and the documents created or updated.

bigger the team gets, the less practical it becomes to have the whole team participate in reviews).

Approve. This milestone is passed when the whole project team has worked through the initial plan and has reached agreement on a revised plan that is ready to be given to the implementer.

Do. This milestone is passed when the implementer has completed an initial implementation—for example, working code and test procedures for a programming module, or a completed first draft for a writing module.

Review and Revise. This milestone is passed when the implementer has revised the code to the satisfaction of the project team, and the approved test procedures have run successfully. That is, other members of the team have reviewed the work of the implementer, communicated any suggestions or concerns, and worked out remaining issues in a project meeting. The implementer has then revised the work to the team's satisfaction. In a particularly troublesome module, further iterations of the review-and-revise cycle might be needed. Often the work is approved as submitted, with no revision necessary.

Evaluate. When the module is completed, the team leader and the implementer evaluate the work, the plan, and the process to look for opportunities for improvement in the future. For example, they may decide on new features or design elements, change subsequent module descriptions or assignments, or modify the way meetings are conducted.

We know this must be a good process because the names of the five milestones make an acronym that spells a real word, and a religious one at that: PADRE. In management discussions, we sometimes talk about "PADREs all the way down," meaning that whether we are managing the project as a whole, managing the day-to-day routine of an individual programmer, or any of the levels in between, we try to go through a cycle of plan, approve, do, review and revise, and evaluate.

What we are describing is known in software engineering as a "spiral life cycle," as opposed to the traditional waterfall model of software development. The project's development could be viewed as a large number of turns through a spiral, each bringing the team closer to its goal (see the sidebar entitled, "Tackling Complexity With an Iterative Strategy").

Figure 2 shows the development of a single module in detail, providing a map of a team member's work during the three days to two weeks it takes to complete a typical module. Notice that we have DACRR here instead of PADRE, since for a coding module it is more natural to speak of planning as designing and doing as coding. Review and revision are split out as separate milestones to show how that step is initiated by a peer review session, after which the programmer makes any changes requested during the review, executes the test procedures, and revises the code until it passes all the tests.[12]

Finally, evaluation is left off the time line completely, since it is usu-

Programmer:
Module name: Module number:
Estimated days to complete this module:
Revised estimate:

Milestone	Original Plan	Revised Plan	Actual Date
1. Begin			
2. Design ready for review			
3. Design revision complete			
4. Code ready for review			
5. Code revision complete			
6. Integration complete			

Figure 3. A "module planning sheet" used by programmers to plan and track the progress of a module of work. Steps 2 through 5 correspond to "PADR" in the PADRE process.

Figure 4. An example of a milestone progress chart

Progress Chart for
Project: *Structure editor* *2 15 93* - Actual start date
Stage: *2. Graphical templates* *3 4 93* - Today's date
Phase: *Prototype* *4 26 93* - Estimated finish date

Milestone % Effort Modules	Designed 20%	Approved 5%	Coded 40%	Reviewed 15%	Revised 20%	Est. Effort	% Done
Select prototype tool	100%	100%	100%	100%	100%	3 days	100
Tree editor	100%	100%	100%	100%	50%	5 days	90
Nested box editor	100%	100%	50%			7 days	45
Graph layout editor	100%					10 days	20
Graph overview	100%					3 days	20
Zoom and pan						3 days	
Paint mult. views						7 days	
Data maps						10 days	
Total estimated effort						**48 days**	
Total % done							**24**

ally done by the team leader rather than the programmer. Evaluation is taking place throughout the stage as modules are completed, with adjustments to plan or process whenever scheduling goals are not met.

The percentages shown in Figure 2 are typical breakdowns that we found in our project. There are frequent surprises that cause these figures to change. But programmers record the time they spend on each step in the process, and these figures make interesting reading during the *postmortem* of a module that took significantly longer than we planned.

Life in the team. Of course, all the team members are working on modules at the same time, circulating documents for review and calling for meetings. That means each person not only has to complete assignments, but also read other people's design documents, participate in review meetings, and share in the other responsibilities of keeping the team organized and moving ahead.

When we started this process, some were worried that there would be so many meetings and demands on their time that there would be little time left to focus on their assigned tasks.[13] And in fact, we have had to take steps to cut down on such overhead:

- Limit the number of team meetings to no more than one per day.
- As much as possible, limit meetings to one hour. If a meeting needs more time, use the last five minutes to schedule a continuation meeting.
- Use a facilitator (or at least keep the mind-set of a facilitator) to make sure action items are recorded and assigned, to draw everyone into discussions, and to make meetings productive and effective.
- Ask each person to designate a "principal reviewer" for a review meeting. This person is honor-bound to read the design or code, and to return comments to the author within 24 hours of receiving it. Other people are asked to partici-

[12]Plan to test. Plan to spend about one-third of your time coding, one-third of your time testing and revising, and one-third doing everything else. Or as one project leader put it, "Half of us are burning toast, the other half are scraping." For some reason people consistently underestimate their ability to design and build things the wrong way. Because it is *much* cheaper to kill bugs early, teams need to integrate testing into all levels of their development process. [13]

[13]So far as possible, protect people from red tape, interruptions, and unnecessary overhead. This isn't easy advice to follow, since it involves everything from organizational change to facilities planning and ergonomics. Many books give good advice in this area, among them DeMarco and Lister's excellent *Peopleware* [10].

Our entire process is based on the observation that **small projects are easier to manage than big projects. Planning proceeds top-down** *until the work has been divided into many small modules.*

pate, but if they are busy they can bow out without suffering pangs of guilt. The role of principal reviewer rotates through the team.

- Make peer review meetings optional. Now that the team is established, we hold review meetings at the discretion of the author and the principal reviewer, who may call a meeting if a module involves difficult questions or has far-reaching effects. New team members' work is reviewed in meetings as a way of acquainting them with the team, the project, and the process.
- Provide good administrative support to the team: someone to answer phones, manage the group archive, make photocopies, schedule meetings, fend off unnecessary demands, and in general grease the wheels of progress.
- Keep the teams small. The peer review process works if no more than four people are developing modules at once. Larger teams will need to divide into smaller peer review groups.

With these measures in place, we have been satisfied with our productivity. When we plan, we assume that people will be able to spend four hours per day focusing on their assigned task. The other hours are spent in meetings, reading, and existential overhead. This sounds high until you realize that the four hours spent away from the assigned task are largely taken up with making other people on the team productive—far more than half the day is spent furthering the interests of the *team*. Our untested suspicion is that the variety of tasks imposed by this process, together with the momentum built by three or four people pushing to get modules completed, makes the team far more productive than if all spent eight hours a day on their own work.

Another advantage comes from the cross-training everyone receives by reviewing one another's work. People tend to settle into specialties, often receiving assignments that relate to the same subsystem. But because they constantly read about one another's work, no one is completely in the dark when asked to work on an assignment outside one's specialty.

Do the module: micromanaging. When a team member starts a module, it is clear what should be done, when it is due, and how it fits into the larger picture. When a team member sits down to work each day, he or she usually knows exactly what to work on, and how much must be accomplished to stay on schedule. Team members are, in effect, managers over a small two-week project, charged with the responsibility of herding their assignments through the PADRE process.

We have designed a form to help these micromanagers, called the "module planning sheet" (Figure 3). This sheet is a key tool for time management and project tracking, at once a byproduct of detailed planning and an enabling technology for tracking development.[14] The first day at work on a module, a team member fills in the "estimated days to complete" field and the "begin" date in the "original" column in the module planning sheet, taking figures from the original schedule for the stage. The other dates in the first column are calculated by multiplying the estimated days to complete the work by standard percentages for each milestone (see Figure 2).

The "revised plan" column is filled in after the team member has progressed far enough into the module design effort to evaluate the original estimate for days to complete the module. It may now be clear that the work was originally over- or underestimated, and the plan needs to be adjusted accordingly. If a programmer thinks the module will take longer than planned, he or she must negotiate the revised estimate with the team leader. The blank for "revised estimate" is then filled in, and the "revised plan" column is completed by beginning with the actual start date and using the standard milestone percentages to extrapolate the other dates. The new set of target dates for each of the milestones then becomes a measuring stick, for just this module in isolation, for answering the question, "Am I doing a good job?" By agreeing on the new estimate, the team leader is saying, "I think you have done a good job if you can finish the module in this many days."

The last column gets filled in as the work on the module proceeds, documenting the programmer's actual progress toward completion. This sheet, together with a time sheet that lists hours spent on each step of the module, is given to the project leader when the module is finished. The project leader uses the information to update the project status, update the productivity history for use in the next round of estimating, and evaluate what is going wrong with the process if the module has taken an excessive amount of time.

Tracking progress. Having defined the process of project management in such detail, we have a powerful tool for tracking the progress of a project. For each stage, the project

[14]Make the process concrete in your forms—design them to reflect the expectations and deliverables specified in the process. Forms such as the module planning sheet help guide people through the process of completing a module, serving as a reminder and a tool for gathering important information. A good form is an effective job aid.

leader creates a "milestone progress chart," like the one shown in Figure 4, using spreadsheet software. This chart lists the modules involved in completing the stage and shows the progress of each module through the spiral. As team members fill out module planning sheets, they report their progress to the team leader, who updates the appropriate progress chart. The chart is kept on the wall in the team's conference room, so everyone can see how things are progressing.[15]

It was stated earlier that breaking big tasks into bite-sized modules makes it possible to monitor the task. Figure 4 shows a concrete example of that concept, providing an accurate, easy to read, detailed picture of the project's status. It also provides a tool for planning—the spreadsheet can use the dates at the top of the form to extrapolate to the probable finishing date given the number of days expended so far and the percentage of scheduled days that have actually been completed.

The chart shown in Figure 4 is our simple version. We also have a more complex version that keeps track by individual team member, so we know not only what the schedule variance for the whole team is, but also each team member's contribution to that variance.

Evaluate and Refine

At the end of each stage, the team sets aside time to reflect on the previous few months. Did any modules take far more or less effort than planned? Why? What can be done to avoid such miscalculations in the future?

This makes an excellent framework for talking about problems and identifying improvements. If a module takes too long to complete, people are less likely to place blame on the programmer than they might be in other frameworks. In the spirit of group ownership of the project and the process, the poststage discussion is more likely to focus on how the programmer could have been better supported, how planning could have improved our original time estimates, or how the process could change to improve performance in the future.

The productivity metrics collected during each module provide objective data on which to base improvements. When analyzing a troublesome module the project leader has time sheets, complexity measures (class and method counts, perhaps with counts of decision points), design documents, results of user tests, and meeting minutes at his or her disposal. Additionally, the team member responsible for the module has been trained to think not only about technology but also about process and communication. As the total-quality books suggest, accurate measurement and a desire to improve are two important ingredients for excellence.

Summary and True Confessions

We have described a process for planning and managing software development. Our experience has made us enthusiastic believers in the principles mentioned in this article and cautious believers in the specific embodiment of those principles in the PADRE process. We say "cautious" as a reminder to ourselves and our readers that this is only a snapshot of something that constantly changes. We can no more tell others they should run their projects exactly this way than we can predict how we will be running projects next year. The specifics will change to fit circumstances and personalities; the principles will not.

During the first two stages, we were eager to know how this approach would affect our productivity and quality, so everyone readily contributed detailed data about his or her work: metrics forms, time sheets, planning sheets, and so on. We were pleased with the results. We were more productive here than on previous projects, we were improving our ability to plan and estimate, and the process was becoming a habit rather than an exercise.

As the stages went by, it was all less new and exciting. We now collect far less data, having settled on a few key metrics. We are less formal in assigning roles during meetings. We sometimes skip review meetings, having learned a little about recognizing the difference between troublesome and benevolent modules.

For these reasons, and because the nature of our work is changing as we move from systems development to applications, our process is evolving. For example, after using the module planning sheets for a year or so, they fell into disuse—the team felt that the milestone progress chart was adequately meeting our need for tracking and accountability. Now the team leader makes the rounds once every week to check each team member's progress, and enters the findings on the chart.

The process documented here is the result of a recent refinement, an effort to incorporate the lessons of the past three years and to accommodate a broader range of activities. We wanted a process that would work for other teams in our organization—something we could easily teach to other managers and apply to new situations. The process continues to refine itself. C

[15] **Keep everyone informed about the project's status. We post progress charts publicly, including the ones that show the progress of individual team members. The simple act of posting this information builds morale, helps people feel included, and provides incentive to make progress.**

References

1. Arthur, L.J. *Improving Software Quality: An Insider's Guide to TQM*. Wiley and Sons, New York, 1993.
2. Brooks, F.P. *The Mythical Man-Month*. Addison-Wesley, Reading, Mass., 1978.
3. Connell, J. and Shafer, L. *Structured Rapid Prototyping: An Evolutionary Approach to Software Development*. Prentice-Hall, Englewood Cliffs, N.J., 1989.
4. Constantine, L. Building structured open teams to work. In *Proceedings of Software Development '91*, Miller Freeman, San Francisco, Calif., 1991.
5. Constantine, L. Decisions, decisions. *Comput. Lang.* (Mar. 1992).
6. Constantine, L. Consensus and compromise. *Comput. Lang.* (Apr. 1992).
7. Constantine, L. Negotiating consensus. *Comput. Lang.* (May 1992).
8. Constantine, L. The lowly and exalted scribe. *Comput. Lang.* (June 1992).
9. Constantine, L. Team harmony. *Comput. Lang.* (Sept. 1992).
10. DeMarco, T. and Lister, T. *Peopleware: Productive Projects and Teams*. Dorset House, 1987.
11. Gery, G.J. *Electronic Performance Sup-*

port Systems. Weingarten Press, Boston, Mass., 1991.
12. Rettig, M. Software teams. *Commun. ACM 33*, 10 (Oct. 1990), 23–27.
13. Rettig, M. Testing made palatable. *Commun. ACM 34*, 5 (May 1991), 25–29.
14. Rettig, M. Cooperative software, *Commun. ACM 36*, 4 (Apr. 1993), 23–28.
15. Rettig, M., Simons, G. and Thomson, J. Extended objects. *Commun. ACM, 36*, 8 (Aug. 1993), 19–24.
16. Scholtes, P.R. *The Team Handbook.* Joiner Group, Madison, Wis., 1988.

CR Categories and Subject Descriptors: K.6.1 [**Management of Computing and Information Systems**]: Project and People Management—*training*; K.6.3 [**Management of Computing and Information Systems**]: Software Development; K.7.2 [**The Computing Profession**]: Organizations

General Terms: Management

Additional Key Words and Phrases: Software teams

About the Authors:

MARC RETTIG is a former member of the technical staff at the Summer Institute of Linguistics and currently a performance support architect at Andersen Consulting. Current research interests include supportive applications and information design. **Author's Present Address:** 69 West Washington Street, Chicago, IL 60602; email: 76703.1037@compuserve.com

GARY SIMONS is director of academic computing at the Summer Institute of Linguistics. Current research interests include object-oriented conceptual modeling and the Text Encoding Initiative. **Author's Present Address:** 7500 West Camp Wisdom Road, Dallas, TX 75236; email: gary.simons@sil.org