Evolutionary Project Management

Adopting EPM in its entirety can be daunting. This case study describes the adaptation of EPM to a small-team environment and how the team obtained more quality from fewer resources in less time.

Stuart Woodward
DoubleHelix Software & Services Ltd.

As a developer of financial software, I have witnessed various project management strategies. As recently as three years ago, I participated in and managed two nearly identical order management systems for investment managers, undertaken at two different companies.

The first project, which I joined as project manager about three-quarters of the way through, used conventional project management techniques and failed without delivering a functional product after 21 months. Subsequently, I joined another company, Anite, and became the project manager of the OMAR (Order Management and Routing) system in July 1996. At that point I determined to learn from the first project's failure and decided to adopt the Evolutionary Project Management (EPM) model.[1] Our team completed OMAR ahead of schedule and provided significantly more functionality than that provided by the failed project undertaken by my previous employer. I'll refer to this earlier project as Project X.

This article describes how I adapted EPM to Anite's small-team environment, as well as EPM's advantages over conventional software project management.

OMAR

Even in its earliest stage, OMAR provided a working investment management infrastructure that included basic equity order and deal entry, manual and automatic routing of orders, broker commissions, basic security features, and a GUI front end. Eleven subsequent development cycles added the following coarse-grained requirements, each consisting of several or more functions: block orders and deal allocation against block orders, trading restrictions (portfolio or security stop lists, broker dealing restrictions, authorization, and so on), limits, amendments and inquiries, futures, options, debt instruments, switch orders, and Euro currency readiness. By contrast, Project X had provided functionality equivalent to only four of these 12 cycles when it was disbanded. Where conventional methods failed to deliver a system that was tested and installed, OMAR's functional units were delivered, tested, and installed at the end of each completed development cycle.

With an average team size of five (OMAR's 5.2 versus Project X's 5.77), OMAR delivered more functionality (61 versus 47 percent of that originally planned) than Project X, delivered it in approximately two-thirds the time, and used only 82 percent of the other project's personpower (104 versus 127 person-months). Figure 1 shows each project's cumulative delivered functionality and benefit-cost ratio over the periods of their execution. We calculated the benefit-cost ratio by dividing the cumulative delivered functionality by the cumulative personpower utilized at each point.

EVOLUTIONARY PROJECT MANAGEMENT

The OMAR development approach was based on the premise that customers and management judge projects and products on firm evidence of delivered results. Therefore, the basic idea was to plan for frequent deliveries of a production quality system that could be demonstrated and installed as a self-contained entity. The system would be delivered in cycles, each of a fixed duration lasting between 6 and 8 weeks. In this way, the requirements of the project as a whole would be more easily managed.

Evolutionary delivery

The approach enabled the team to

- focus on short-term, clear goals with a high degree of confidence of meeting them,
- ensure that regular feedback was obtained at the end of every cycle,
- keep in control of production,
- more easily perform estimation owing to short production cycles,

0018-9162/99/$10.00 © 1999 IEEE

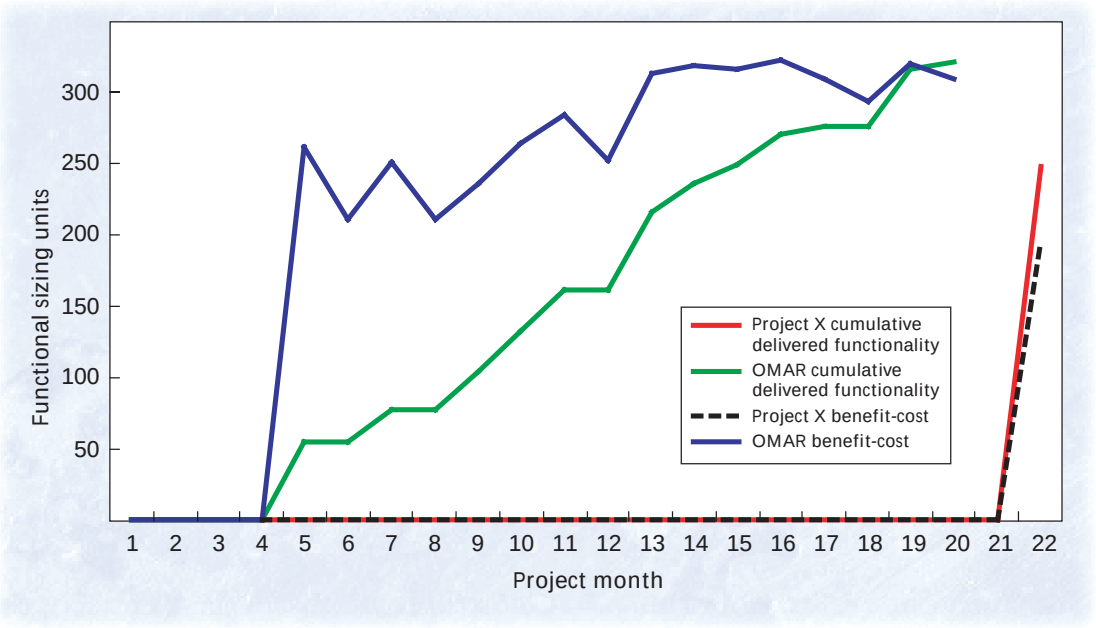


Figure 1. Comparative benefit-cost ratios for each project. The benefit-cost ratio is the cumulative delivered functionality—in functional sizing units (effectively a simplified Function Point measurement)—divided by the cumulative personpower utilized at each point.

- reassess objectives at the beginning of every cycle,
- encapsulate quality requirements within the objectives of the project, and
- deliver fully tested, production-quality software with the right documentation.

EPM, or Evo, acknowledges that a software system is rarely "finished" and that, once delivered, it evolves. More importantly, it provides a mechanism for controlling this evolution from the outset. The project could be stopped in a controlled fashion at the end of any cycle. Regular and frequent feedback uncovers serious problems or needed "changes of tack." Moreover, it provides a stable position to "retreat" to, resulting in the loss or suspension of only one cycle.

We gained a high level of control over the implementation of functionality, but the implementation of Evo on OMAR with respect to the control of quality attributes such as maintainability, performance, and usability was less mature than it might have been. Although I specified a complete structure of these qualities, our process was not mature enough to measure and implement provable improvements. Rather, we used the specifications of these qualities in an intuitive fashion during software design.

Sequence of development cycles

The sales/support department was to be the system's initial recipient. This department maintained contact with prospective customers and provided us with market feedback about important issues in order management. In a sense, they were the owners of the

Table 1. Original versus actual sequence of cycles up to the end of OMAR's development.

Originally planned sequence of development	Actual sequence of development
1. Infrastructure and basic orders	1. Infrastructure and basic orders
2. Block orders	2. Block orders
3. Deal allocation	3. Deal allocation
4. Order amalgamation	4. Investment restrictions
5. Cash, stock availability, and limits	5. Limits
END OF ORIGINAL BUDGET AND SCHEDULE	
6. Switch orders	6. Amendments, inquiries, and placement
7. Order by value and placement	7. Futures
8. Order amendment	8. Options
9. Investment restrictions	9. Debt instruments
10. Debt instruments	10. Future deal by value
11. Import of orders	11. Switch orders
12. Reporting	12. Euro
	OMAR STOPPED HERE

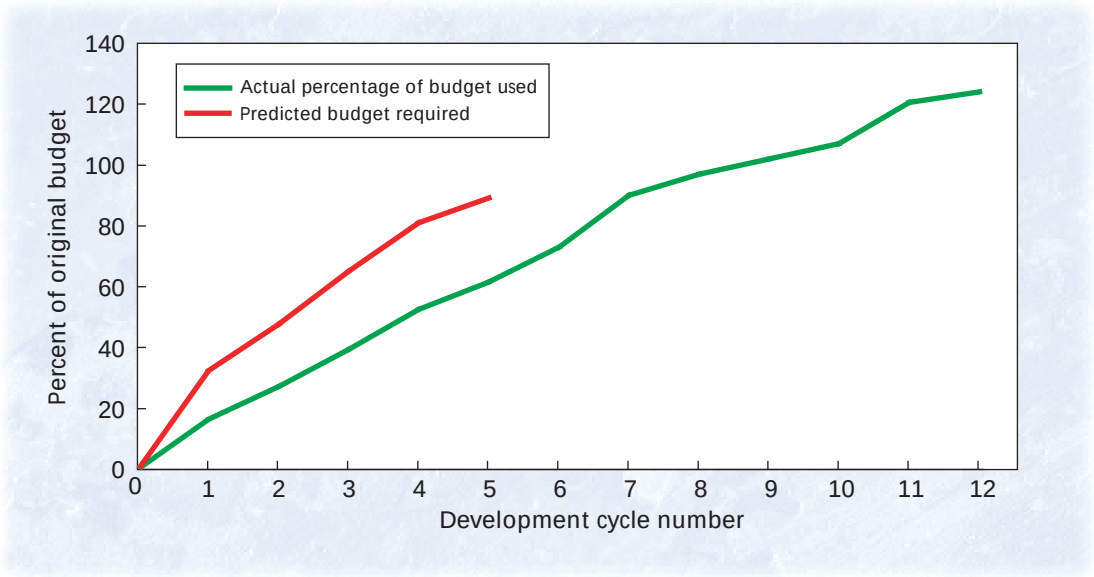


Figure 2. Actual costs compared to budgeted costs.

schedule, which itself constantly evolved according to a cyclic reassessment of priorities.

The milestones on OMAR were not the activities within each development cycle but the delivered production-quality systems at the end of each cycle. I only maintained a very high level Gantt chart that showed the sequence and dependencies of complete cycles. I considered the dependencies between activities within a cycle too complex to manage using Gantt; thus, my charts showed deliverables, not activities.

How was the sequence of development cycles actually determined? Initially, the team decided on a "route" through what would first give us a product that provided basic functional breadth but not functional depth. We then planned on providing richer functional areas, one by one, in a sequence that we considered being typical of the system's use. However, this picture changed quickly with feedback from sales/support as Table 1 shows.

Further, each cycle did not always exhibit a clear-cut identity. For example, although development stopped before order amalgamation could be implemented, OMAR already provided a built-in facility for the user to immediately see which orders could potentially be amalgamated with other similar orders. This would have been part of the Order Amalgamation cycle but was a simple spin-off from existing development, so we included it anyway. We failed to predict some functions at the outset, though, including a complete cycle, Future Deal by Value. Yet because we tried to accommodate every eventuality by constructing an open business and technical architecture, the functionality developed in Future Deal by Value fit in seamlessly.

Speed of delivery

I based the original global schedule and estimate on what I perceived would be necessary after my experience with Project X. Although I organized OMAR differently, I thought that a sensible start would be to assume the same approximate level of resourcing to deliver the same basic amount of functionality. I had nothing else to go on at that time. At no time during OMAR's life did I obtain the eight staff I originally specified. However, as early as the first cycle of development I warned of an understaffed project's consequences—that with only four staff, for example, some less-than-critical functionality would not be completed within the original schedule.

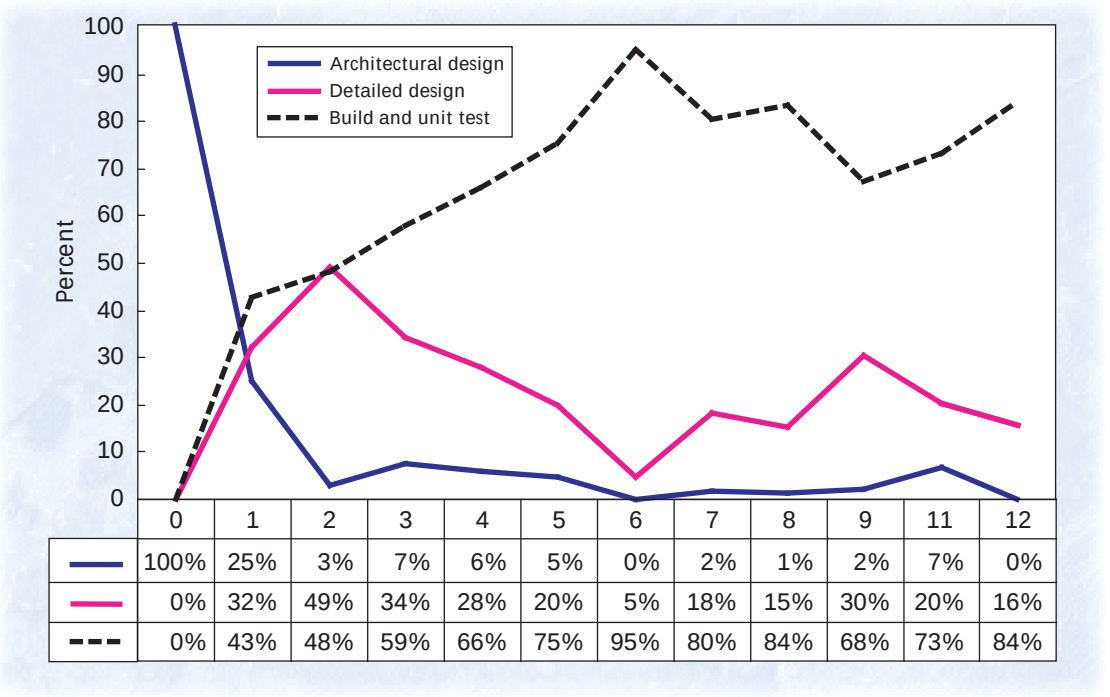
The company had initially authorized the project to continue to the end of the first five originally planned development cycles, at the cost I had originally estimated those cycles to be. The events shown in Figure 2 resulted. We used up the original budget by the end of the seventh development cycle, not the fifth as originally expected. Anite's management then allowed the team to continue working on the project.

Cycles and steps

Tom Gilb suggests that an Evo step should typically use 2 to 5 percent of both budget and schedule. Clearly, the cycles of development above do not conform to this. Could the cycles have been further subdivided? At the time they were devised for the purposes of producing a global plan, I did not know how to achieve it. Also, the team did not consider it sensible to add one function without adding another to which it logically related. For example, a system that allowed order entry without a corresponding deal entry function would have been stable but probably unusable.

At that time, though, I was being too restrictive in my interpretation of the cycle concept. An EPM cycle is not necessarily one that results in a final delivery to an end user; it could simply be the addition of something that increases the functionality within the system but that, nevertheless, leaves the system in a stable state. The full or "external" cycles themselves would be treated as the milestones or delivery points to end users.

As it turned out, we were probably making this assumption without realizing it and were even reporting the step progress within cycles to senior management so that they gained confidence in the process,



	0	1	2	3	4	5	6	7	8	9	11	12
Architectural design	100%	25%	3%	7%	6%	5%	0%	2%	1%	2%	7%	0%
Detailed design	0%	32%	49%	34%	28%	20%	5%	18%	15%	30%	20%	16%
Build and unit test	0%	43%	48%	59%	66%	75%	95%	80%	84%	68%	73%	84%

Figure 3. Cycle "shapes," or relative proportions of actual work done per development cycle. (No data was available from cycle 10.)

too. Plenty of evidence in the team meeting minutes substantiates this claim.[2]

TIME MEASUREMENT AND ESTIMATION

EPM is superior to the conventional project management model with respect to estimating. For a start, you get more practice. Whereas Project X provided a single "up-front" project estimate that inevitably became an immovable target, OMAR provided a detailed estimate for every cycle within a global project plan. Each cycle estimate used the experience gained in previous cycles.

The biggest advantage of doing a sequence of project estimates is that you can learn from the experience. The literature on estimation and software metrics advises us to record all past estimates and actuals from old projects. I agree with the need to do this, but consider the following.

- You must establish a reasonably stable culture in which the data is recorded and managed.
- Experience has taught me that it's difficult to make comparisons between projects: Rarely do you have the same personnel on more than one project; meanwhile, technology advances, work practices mature, and so forth.

So how did I manage the estimation of the cycles? My objective was to make the method reusable throughout the life of the project and beyond. I therefore needed to create an easy and quick-to-use framework that could also be transported between cycles, making comparison and learning effective.

Sizing functional areas

My work on Project X taught me that the time records kept by the technicians were of little or no use. Team members recorded time only against the functional area being worked on—for example, programmers would simply book time against Order Form. This practice made it impossible to compare different areas because I had no way of "sizing" the functional areas. I therefore established a structured and controlled time recording system based on both the activities each person would perform during a development cycle and the functional areas within the product. So, for example, a programmer would book time against the detailed design activity of the order form functional area. This approach let us compare the development cycles' "shapes" in terms of their activities, as shown in Figure 3, and helped us adjust for future estimates.

Estimation consisted of deciding how much of each activity—architectural design, detailed design, and build and unit test—would occur for each piece of functionality to be delivered. Since these activities dominated and characterized each cycle, I assumed that other activities correlated to these and therefore derived an overall estimate from a formula that evolved, based on actual data, cycle by cycle.

Time codes

To make these methods work, I had to ensure that the team took time recording seriously. In previous companies, time sheets were practically useless for analyzing actual work against estimates because staff was under pressure to remain as highly billable as possible, and thereby minimize unpaid work. To charge customers for work actually done, these companies used a very simple set of time codes that ensured that work descriptions appearing on invoices were not questioned. Inevitably, this led to a lazy approach to booking time; essentially, it did not matter what type of work was being booked to the client as long as it

Table 2. Time sheet codes.

Time sheet code	Name	Description
ADD	Architectural design	A reassessment of existing interfaces between software components or the design of new ones performed by the developers
DDD	Detailed design	The specification of the implementation of software components
BLD	Build and unit test	Software construction, unit, and integration testing. Performed by the developers
Other codes		
ATT	Acceptance test	Acceptance testing by sales/support of the completed cycle's output
EST	Estimation	Detailed estimation performed by the development team and project manager
ISP	Inspection	Verification and validation by means of formal inspections
MAD	Adaptive maintenance	For example, the porting of the system to a new database
MCO	Corrective maintenance	The fixing of bugs reported during system testing
MPE	Perfective maintenance	The improvement of specific algorithms or software components
RTT	Regression test	Regression testing
SRD	Software requirements	Functional specification—the detailed specification of the functions from the perspective of the user in use case format
STT	System test	System test specification and testing by a tester
URD	User requirements	The update of the requirements definition and corresponding functional specification

looked realistic and customers paid. Furthermore, management frowned upon unpaid work, which made staff members defensive about what they did outside billable work. Time was habitually booked to "bucket" codes that did not truly describe their work or were used in the absence of other useful or authorized ones. This practice made analyzing the actual work done impossible.

I overcame this obstacle by providing a detailed, but restricted, list of time sheet activity codes. I provided codes that acknowledged the technicians did more than simply programming or testing. I checked every time sheet, a practice that encouraged recording time responsibly because team members now felt they could be honest and that the time sheets were contributing to the project's success.

A typical development cycle consisted of the activities shown in Table 2. Obviously, I used these same codes in the estimation formula itself.

Having to provide a monthly report to senior management, including details of actuals against estimates or progress against plan, prompted me to be especially scrupulous in ensuring the figures' accuracy. This discipline paid dividends in improving the team's estimating skills. It also gave the management a good deal of confidence in our ability to forecast delivery dates, allowing them to publish confidently the planned delivery dates of each area's functionality. In Figure 4, I attempt to derive a correlation between the size of a development cycle in person-days and the accuracy of our estimates. The estimates' quality hints at an optimum size of cycle that the team could control.

REQUIREMENTS DEFINITION AND FUNCTIONAL SPECIFICATION

Project X only had one completed document—the Business Domain Definition, generally referred to as "the Spec." The Spec was the equivalent to the Software Requirements Document (SRD) on OMAR, which documented—by means of use cases—the functionality to be available in the product.

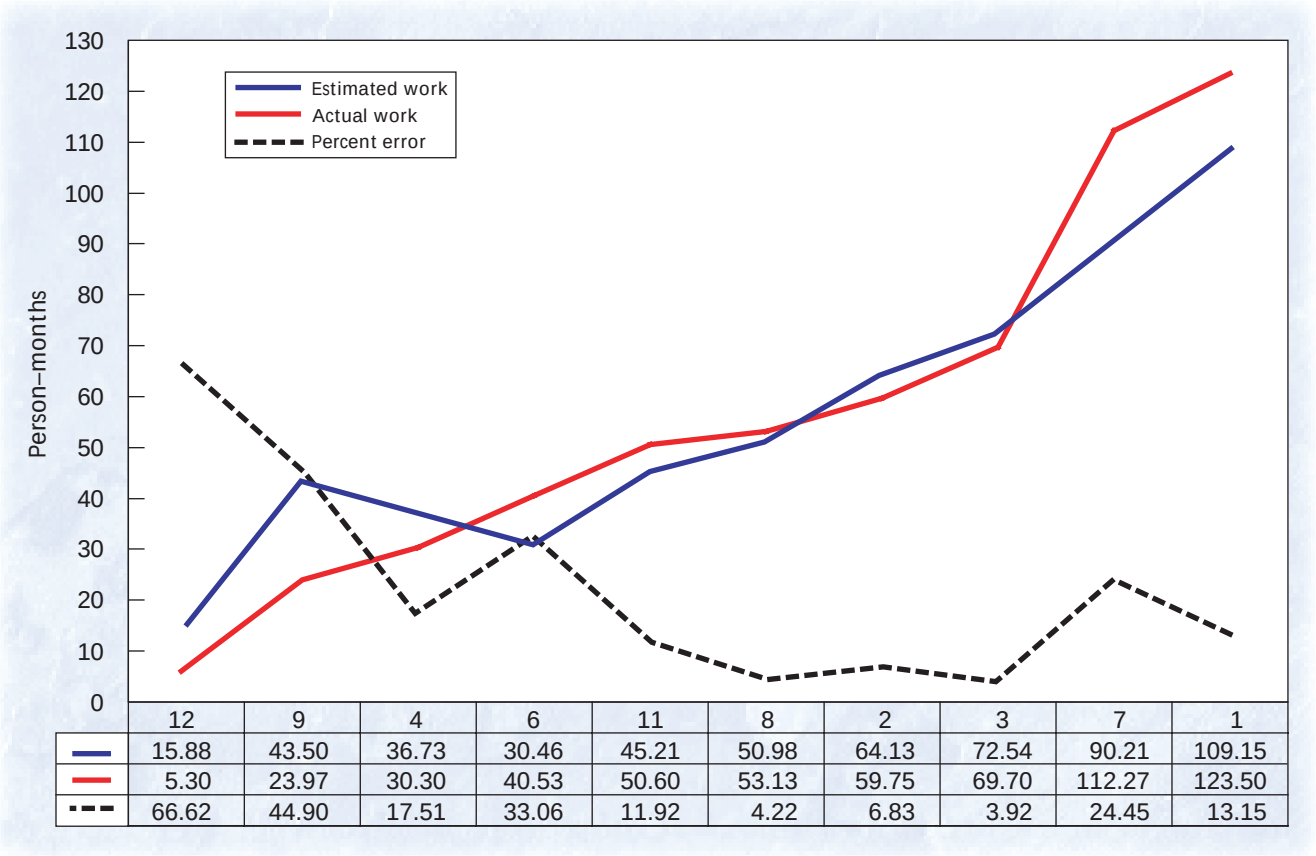
Project X's Spec

The organization running Project X insisted on controlling the scope of functionality within the product. One consequence of this approach was that the system's customers had to sign off on every Spec page. Further, the Spec formed part of the contract between the two parties; at the outset of development they had agreed on the exact scope and form of functionality to be delivered. This decision gave rise to a bizarre set of circumstances.

As business analysts and users continued to consult on the finer points of detail, the scope and nature of the project's functional requirements evolved. This phenomenon almost inevitably occurs on long projects. However, no one ever formally documented these changes, and thus Project X's staff never updated the Spec.

One reason for this lapse was that it took so long to obtain sign-off on the original Spec that the organization felt it could not go through the same painful process each time something changed. The "straitjacket" of Project X's conventional model, together with its laborious sign-off process, led to serious problems. The first undocumented change of requirements set a precedent that users felt they could repeat, which they did, typically at each monthly meeting between themselves and the project staff. It was simply assumed that the analysts would communicate the resulting undocumented changes to the technicians who designed and built the system. The functional scope of the product soon spiraled out of control.

There was worse to come. The users expected to receive and use Project X's test specifications. The Spec was a lengthy (127-page) and wordy document of poor structure. This made it impossible to generate



	12	9	4	6	11	8	2	3	7	1
Estimated work	15.88	43.50	36.73	30.46	45.21	50.98	64.13	72.54	90.21	109.15
Actual work	5.30	23.97	30.30	40.53	50.60	53.13	59.75	69.70	112.27	123.50
Percent error	66.62	44.90	17.51	33.06	11.92	4.22	6.83	3.92	24.45	13.15

Figure 4. Estimation errors per cycle in order of actual work-size of cycle. (No data was available from cycles 5 or 10.)

test specifications that could be easily traced back to it. The chief tester solved this quandary by redrafting the Spec to make traceability possible.

Project X therefore had the official Spec, undocumented analyst changes communicated by word of mouth to the technicians, and a new version of the original Spec from which the testers specified and performed testing. None of these procedures were documented or reviewed; the analysts, testers, and technicians communicated informally at best.

OMAR's functional specification

On OMAR I made it top priority to avoid repeating this situation. I adopted some simple tactics to start with, including the following:

- specifying a basic set of requirements in the User Requirements Document (URD), but allowing these to evolve;
- specifying requirements in short summaries instead of lengthy paragraphs, which were then expanded into full functional specifications at the time of implementation; and
- tagging every requirement uniquely to aid traceability throughout the development process.

I resisted the immediate inclusion of every suggested enhancement to the product, however small, until its effect on the schedule could be estimated. Once under way, no development cycle was allowed to creep in scope, although weekly team meetings provided a forum for discussing the inclusion or exclusion of specific product features.

We considered how the full Functional Specification, the SRD, would be published, bearing in mind it would grow incrementally and be reissued at the end of every cycle. I insisted that at every step its contents must be directly traceable to the URD by having the same basic sectional layout with identical requirements tags. We then gave each requirement yet to be implemented an additional status tag, as follows:

- <<P>> Requirement is to be implemented during a future development cycle.
- <<IF>> Requirement forms an interface and is to be implemented during a future customer installation cycle.
- <<SDM>> Requirement is to be implemented during a future, standing data-maintenance development cycle.
- <<U>> Requirement is to be implemented according to user demand.

This simple device allowed any version of the SRD to be agreed upon and, if necessary, signed off. The functional specification therefore always reflected the functionality of the version of the system that was actually in existence, delivered, and tested. Also, the quality department, sticklers for version control, loved it.

Other Contributing Factors

In addition to the EPM, other factors contributed to OMAR's ultimate success.

Quality management system

A quality management system (QMS) can provide the framework in which projects are initiated, executed, managed, and controlled. For example, a great deal of time and effort can be saved if documented procedures and templates exist from which specific project documents begin.

The organization undertaking Project X had established a full QMS several years previously but it had not been maintained. There was nobody to enforce the project's quality standards, the lack of adherence to which were clearly evident in Project X.

Anite has a full QMS that also received ISO 9001 certification. Its online availability promotes its active use. Most of the documents created within OMAR began or evolved from standard templates available from the Anite QMS. The project team was saved considerable time and effort by having this system in place.

OMAR underwent a periodic formal audit conducted by the Quality Department to ensure its compliance with the QMS. The project therefore received a great deal of support from the company, its management, and its infrastructure, into which the QMS was integrated.

Project standards

As project manager, I viewed my role as one of facilitator to the team members, enabling them each to do their own jobs better and to improve their productivity when working together as a team. I searched for ways to improve the communication between everyone involved in the project.

One problem EPM recognizes is the poor clarity that often occurs in requirements definition. I extrapolated this principle to enforce as much clarity on all documents as possible.

Further documents enforcing consistency and unambiguous specification included

- *Error Code names within the software source code.* These were maintained centrally and the appropriate source code files were generated automatically via a spreadsheet macro.
- *The definitions document.* This document defined all terms used in relation to the business area that OMAR addressed. For example, the team initially used the terms Client, Fund, and Portfolio in what seemed an interchangeable fashion, but I could not be sure. I insisted that we standardize on single terms for single meanings to avoid any potential confusion.
- *Visual Basic and C++ coding standards.* We applied a light touch and, in conjunction with the Naming Conventions, mainly enforced code architecture. The intent of this approach was to enable any team member, where necessary, to quickly locate specific routines, classes, or modules, and to understand where they traced back to the design and requirements.

Everyone was encouraged to contribute to the continuing evolution of these standards.

OPEN ARCHITECTURE

I determined at the outset to define an open architecture for OMAR. In particular, the team knew that OMAR must be able to interface to other software systems that were both functionally and technically different in nature. Since we had no external customer to fund the project at the time, we could only guess at—or go by our experience of—what these systems might be. Open-ended design therefore proved essential.

Architecture cycle zero

Open architecture can apply to product structure, parameterization, strategy in the best use of third-party components and tools, and so forth. Issues decided in the initial architecture stage included

- basic standing data maintenance method,
- GUI screen field validation methods,
- font size configuration,
- third-party tool use,
- third-party libraries,
- SourceSafe configuration, and
- GUI standards regarding look and feel of forms and screens.

Open architecture also allowed us to develop and implement some components independently of others.

All this may seem intuitively obvious. How many projects, however, attempt to address these issues downstream in the development process only to see the effects of late decisions or inconsistencies between the work of team members cause costly repairs or alterations to already developed code? I wanted to avoid this scenario at all costs. Executing an architecture cycle proved an excellent way to start a major development project.

Architectural design documentation

The architecture of the design documentation mirrored the architecture of the product itself. For each software component, we maintained an architectural design document (ADD), owned by the person responsible for its design. Detailed design documents (DDDs) for each component then related directly to the ADDs. Source code in turn related directly to the DDDs. The enforcement of the architecture aided traceability throughout the development process. Fred Brooks was convinced of the link between conceptual integrity and product quality.[3] So was I.

QUALITY CONTROL

Evolutionary delivery gave us the chance to test the complete system frequently.

Testing

On Project X, the specification and execution of tests occurred independently from the development team. Although not a problem in itself, it became a serious concern because the test specification was totally detached from the source document that drove development.

Fred Brooks was convinced of the link between conceptual integrity and product quality.

On OMAR, the specification and execution of tests also occurred independently from the development team. However, owing to the traceability of every specified test back to the requirements definition and functional specification, day-to-day cooperation between the development team and testers occurred naturally. Thus, the defined and stable environment in which we worked let us construct a successful test strategy, plan, and set of scripts.

OMAR was not without its own problems, however. Management never allocated sufficient personnel to our project, and consequently we never spent sufficient time on system testing. I knew this would happen at the outset and so concentrated on preventive measures that essentially equated to the quality of build. Nevertheless, I felt confident enough to allow prospective customers to inspect our test documentation, which did happen.

Inspection

Tom Gilb's and Dorothy Graham's book *Software Inspection*[4] contains a self-assessment test on how an organization or project rates in its implementation of inspections. On a numeric scale ranging from 0 to 45, OMAR scored 11, which places it in the Beginning class. On the face of it, this seems a particularly poor result, and I would agree that OMAR had a long way to go to establish a full inspection process. However, on the same scale, Project X scored just 1, indicating the true situation of having no inspection or review process whatsoever.

In the verification and validation plan from OMAR I defined a review (distinct from inspection) as an informal gathering of personnel to discuss the composition and construction of project components. Normally, a review would take place before inspection.

On Project X, the senior members of the development team and representatives of the user organizations individually reviewed the Spec. No other reviews of any kind took place, and the project team produced no records of any, including that of the Spec.

On OMAR, I wanted to engender a culture of high-quality work and continually improving processes. I wanted the team to understand that, just as you cannot test quality into software, you cannot inspect quality into documents.[5] I intended to use inspection as a tool to tell us how good—or bad—our technical processes were, and I wanted to be able to publish the results.

In terms of their everyday jobs, I encouraged the developers, nominally analyst-programmers, to think of themselves as designers, with coding merely being the design's implementation. This focus helped shift the emphasis of their work upstream in the development process and encouraged them to remove bugs before they actually got into the code. At the project's start someone suggested to me that "all code has bugs in it." I simply asked this person, "who puts the bugs into the code in the first place?" Imparting this attitude helped everyone take much more responsibility for preventing bugs.

Inspection process. As with EPM, I had to start with something and implement it quickly. I decided that it would be too difficult or too big a cultural change to try implementing the full inspection process. The essential points of the process I did implement follow.

The originator created the item for inspection. The creation of each item could include informal reviews, walk-throughs, and brainstorming sessions. These latter activities would not normally be recorded. Verification and validation would be by means of formal inspections, as follows:

1. Items were presented for inspection to the project manager. The project manager acted as the moderator and coordinator of inspections and chose who would perform the inspection. Anyone associated with the project could act as an inspector. (The project manager, or others experienced in inspection, introduced new personnel to the inspection process as needed.)
2. Optionally, project staff held a meeting that let the inspectors become familiar with the item, possibly by means of a walk-through.
3. We used the Inspection Form to record defects in the item. The inspection should continue for a maximum of one hour (unless otherwise specified by the project manager), or until 10 defects had been located under a single category, or until 25 defects in total had been located, although discretion should be used in each case.
4. The originator removed any defects located in an inspected item, then resubmitted the item for inspection. This cycle would repeat until the item reached or excelled its planned levels of quality.

Typically, each first inspection of an item would expect to locate defects. Second and subsequent inspections that located defects would require discussion to fully resolve issues, or would indicate problems in the production of the item being inspected. The project manager and the team met to discuss the appropriateness of the planned quality levels and decide on a course of action. Meeting participants paid attention to critical objectives of the project as a whole, such as cost, duration, and quality.

After the third cycle, and in the event that the item had not been completed according to plan, the project manager, originator, and inspectors met to discuss problems arising during the production of the item

and how to avoid them in future. Participants avoided apportioning blame because project staff considered every item to be the product of the team as a whole. However, we did make the results of the inspections public.

RISK MANAGEMENT

At OMAR's outset, we set up a Risk Register and maintained it throughout the project's life. We also published the register in each monthly report to senior management.

In the register we listed each risk item, along with the probability the risk would occur, the impact it would have on the project, and the remedial actions we intended if the risk occurred. We informally reviewed the register at team meetings and updated it in each monthly report.

The Risk Register's biggest benefit to us was that senior management kept in touch with the most pressing concerns of the team. Further, because we presented these concerns in a structured and considered way, management took them seriously. All too often on Project X, senior management ignored the team's complaints because they represented risks to the project that had already occurred. The absence of planning for the risks' occurrence meant that senior management had no way of knowing how to deal with problems when they happened.

On the OMAR project, the Risk Register represented another aspect of removing uncertainty from our progress. I took a Murphy's law attitude of "what can go wrong probably will go wrong" and tried to instill in the team an attitude of attacking problems before they attacked us.

Adopting the Evolutionary Project Management method in its entirety can be daunting. Forced to choose a starting point, we took the path I've described. This approach, as evidenced by the OMAR project, resulted in only a partial uptake of EPM. Nevertheless, our efforts were a successful first step on the road to further uptake of the method and greater success from its use.

The adoption of EPM is itself an evolutionary delivery project. In the future, I intend to improve the way I implement each aspect of EPM on each new project. EPM brought beneficial changes at Anite, including an irreversible change in the mind-set of the personnel involved in the OMAR project. We no longer expect to deliver software via the traditional waterfall model; we plan to be able to change direction quickly and without negative impact on schedule or budget; we aim to be able to prove to what extent requirements have been met; and we commit ourselves to delivering quality on time, every time.

We now think naturally of executing software engineering projects by means of Evolutionary Project Management. ❖

Acknowledgments

I thank Anite's Finance Division, where OMAR took place, for permission to access the project records and also my current employer, Access Accounting, for their cooperation. Thanks especially to Tony Ruben and Neil Oag, who were involved in both projects, for valuable insights and recollections of events on Project X. I am indebted to Tom Gilb, who persuaded me to write the original case study and for his many constructive suggestions.

I dedicate this article to my long-time colleagues Justin Nahum, the architect of OMAR, and to Geoff Dixon, managing director of DoubleHelix and formerly my manager at Anite who made the OMAR project possible.

References

1. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Wokingham, UK, 1988.
2. S. Woodward, *Using Evolutionary Project Management to Get More Quality from Fewer Resources in Less Time*. (This is the original and full case study and is available from http://www.natural-metrics.freeserve.co.uk/download.htm.)
3. F. Brooks, *The Mythical Man-Month*, 20th Anniversary Edition, Addison-Wesley, Reading, Mass., 1995.
4. T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley, Wokingham, UK, 1993.
5. H.S. Gitlow and S.J. Gitlow, *The Deming Guide to Quality and Competitive Position*, Prentice Hall, Upper Saddle River, N.J., 1987.

Stuart Woodward wrote this article while a consultant at DoubleHelix Software & Services Ltd., which provides project management and business analysis services to London's financial institutions. Woodward's main interests are in project management, especially evolutionary project management, and the improvement of the software development process, specifically through improved techniques of estimation, risk management, requirements definition, and software architecture specification. He has a BSc (Hons) in software engineering management from the University of East Anglia. Contact Woodward at stuart.woodward@natural-metrics.freeserve.co.uk.