The Evolutionary Development Model for Software

The traditional waterfall life cycle has been the mainstay for software developers for many years. For software products that do not change very much once they are specified, the waterfall model is still viable. However, for software products that have their feature sets redefined during development because of user feedback and other factors, the traditional waterfall model is no longer appropriate.

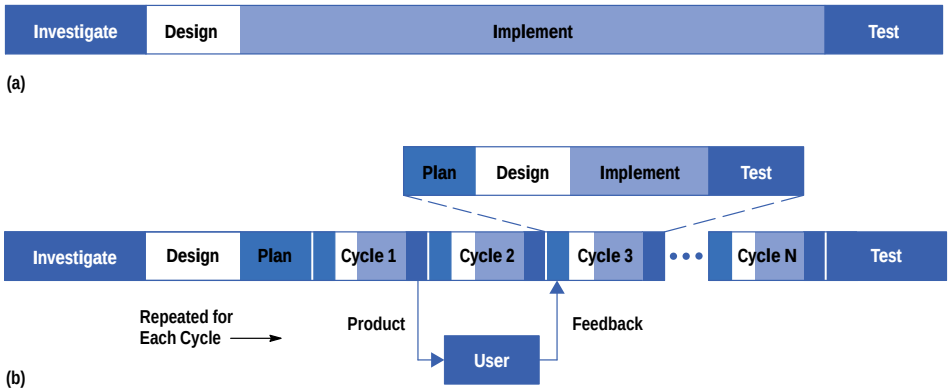
by Elaine L. May and Barbara A. Zimmer

Hewlett-Packard, like other organizations developing software products, is always looking for ways to improve its software development processes. One software development method that has become quite popular at HP is called *Evolutionary Development*, or EVO (see reference 1 and ***Article 3***). EVO uses small, incremental product releases, frequent delivery to users, and dynamic plans and processes. Although EVO is relatively simple in concept, its implementation at HP has included both significant challenges and notable benefits. This paper begins with a brief discussion about the EVO method and its benefits, then describes software projects at three HP divisions that have used EVO, and finally discusses critical success factors and key lessons about EVO.

The EVO Method

Fig. 1 shows the difference between the traditional waterfall life cycle and the EVO life cycle. The EVO development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle. The users provide feedback on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plans, or process. These incremental cycles are typically two to four weeks in duration and continue until the product is shipped.

Fig. 1. *Software development life cycles. (a) Traditional waterfall model. (b) Evolutionary (EVO) development model.*



At Hewlett-Packard, we have found that it is possible to relax some of our original ideas regarding EVO. In particular, it isn't absolutely necessary to deliver the product to external customers with customer-ready documentation, training, and support to benefit from EVO.

Benefits of EVO

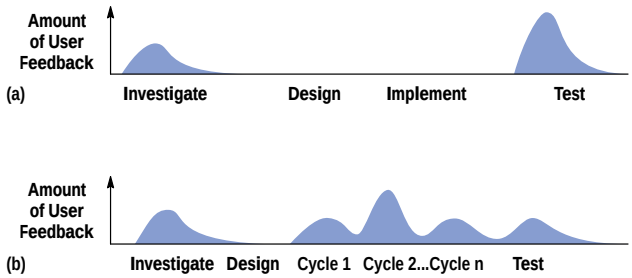
Successful use of EVO can benefit not only business results but marketing and internal operations as well. From a business perspective, the biggest benefit of EVO is a significant reduction in risk for software projects. This risk might be associated with any of the many ways a software project can go awry, including missing scheduled deadlines, unusable products, wrong feature sets, or poor quality. By breaking the project into smaller, more manageable pieces and by increasing the visibility of the management team in the project, these risks can be addressed and managed.

Because some design issues are cheaper to resolve through experimentation than through analysis, EVO can reduce costs by providing a structured, disciplined avenue for experimentation. Finally, the inevitable change in expectations when users begin using the software system is addressed by EVO's early and ongoing involvement of the user in the development process. This can result in a product that better fits user needs and market requirements.

EVO allows the marketing department access to early deliveries, facilitating development of documentation and demonstrations. Although this access must be given judiciously, in some markets it is absolutely necessary to start the sales cycle well before product release. The ability of developers to respond to market changes is increased in EVO because the software is continuously evolving and the development team is thus better positioned to change a feature set or release it earlier.

Short, frequent EVO cycles have some distinct advantages for internal processes and people considerations. First, continuous process improvement becomes a more realistic possibility with one-to-four-week cycles. Second, the opportunity to show their work to customers and hear customer responses tends to increase the motivation of software developers and consequently encourages a more customer-focused orientation. In traditional software projects, that customer-response payoff may only come every few years and may be so filtered by marketing and management that it is meaningless. Fig. 2 illustrates the difference between the traditional life cycle and EVO in terms of how much user feedback can be expected during product development.

Fig. 2. *Amount of user feedback during (a) the traditional waterfall development process and (b) the evolutionary development process (EVO).*



Finally, the cooperation and flexibility required by EVO of each developer results in greater teamwork. Since scheduling and dependency analysis are more rigorous, less dead time is spent waiting on other people to complete their work.

While the benefits can be substantial, implementation of evolutionary development can hold significant challenges. It requires a fundamental shift in the way one thinks about managing projects and definitely requires more management effort than traditional software development methods. The next section examines how EVO was applied in three different HP divisions and what we learned from the experience.

Evolutionary Development in Practice

Some form of EVO has been used in at least eight Hewlett-Packard divisions in over ten major projects. Much of this has been done drawing on expertise from HP's Corporate Engineering software initiative, which is a central service group of consultants in software engineering and management (see ***Sidebar***). The software initiative group is currently leveraging existing experience and promoting the use of EVO at HP.

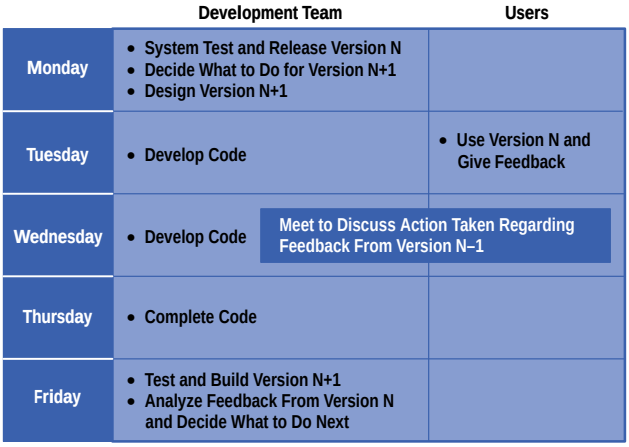
The three divisions described below are in three entirely different businesses. While all the product names used in this paper are fictitious, the case descriptions are real.

First Attempts

The first project was undertaken at HP's Manufacturing Test Division. The project (called project A here) consumed the time of four software developers for a year and a half and eventually was made up of over 120,000 lines of C and C++ code. Over 30 versions were produced during the eleven-month implementation phase which occurred in one- and two-week delivery cycles (see Fig. 3). The primary goals in using EVO were to reduce the number of late changes to the user interface and to reduce the number of defects found during system testing.

Project A adapted Gilb's EVO methods.[1] One departure was the use of surrogate users. The Manufacturing Test Division produces testers that are used in manufacturing environments. If the tester goes down, the manufacturer cannot ship products. Beta sites, even when customers agree to them, are carefully isolated from production use, so the beta software is rarely, if ever, exercised. Fortunately, the project had access to a group of surrogate users: application engineers in marketing and test engineers in their own manufacturing department. The use of surrogates did not appear to have any negative impact.

Fig. 3. *An example of a typical one-week EVO cycle at the Manufacturing Test Division during project A.*



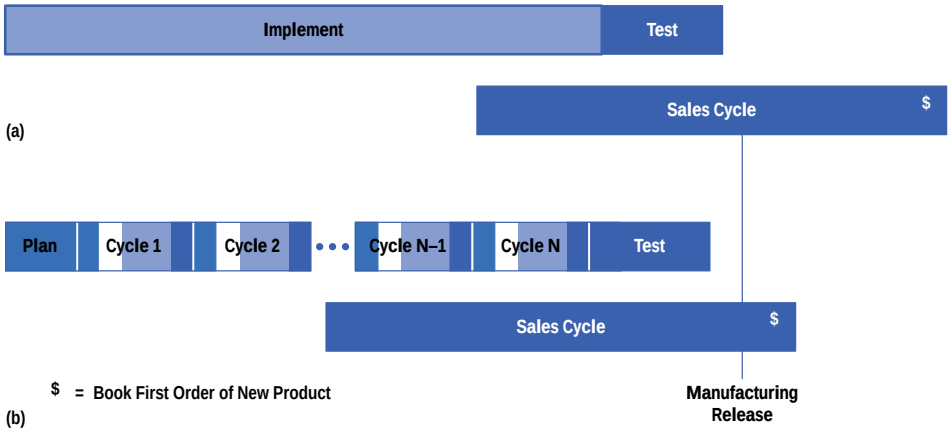
About two thirds of the way through the project, the rigorous testing and defect fixing that had been done during the EVO cycles was discontinued because of schedule pressures. The cost of this decision was quality. With all efforts focused on finishing, developers began adding code at a rate double that of previous months, and over half of the critical and serious defects were introduced into the code in the last third of the project schedule.

Even though EVO was not used to complete the project, the product was successful and the team attributed several positive results to having used the EVO method for the majority of the project. First, EVO contributed to creating better teamwork with users and more time to think of alternative solutions. Second, the project still had significantly fewer critical and serious defects during system testing. Third, the team was surprised to see an increase in productivity (measured in KNCSS per engineer-month). The project manager attributes this higher productivity primarily to increased focus on project goals.

Despite having abandoned the EVO method in project A, many in the division felt that because of the benefits derived from the method, they should give it another try. Project B, the second project to use the EVO method, involved creating custom hardware with a team of three project managers and 20 engineers. The project required significant changes to over 1.5 million lines of code. One project manager coordinated the efforts of three development teams.

The primary reason for using EVO for this project was to demonstrate the feasibility of the product's new test technique through the use of beta sites. Internal users were used early in the cycle and external customers became involved with later versions. This decision resulted in actually selling systems to beta-site customers. Further, the traditionally long startup time for the division's sales cycle was shortened significantly by the use of EVO and validation of the product by users (see Fig. 4). This created a major business impact since it typically takes nine to fifteen months before the market believes a product of this type can live up to its claims. Even more time passes before customers will actually buy the product. Because EVO encourages early exposure of a product to users, sales started even before the product shipped!

Fig. 4. *An accelerated sales cycle in (a) the traditional waterfall life cycle and (b) the EVO cycle.*



Start Small

The experience of HP's Personal Software Division with EVO also began with some startup problems, followed by remarkable success. The first project to use EVO was chosen because it was felt that EVO would help to prioritize new features, respond quickly to customer needs, and, because of EVO's many release cycles, enable the release of the software product at any time in response to competition.

The six to eight project managers and approximately 60 engineers on this project were all new to the EVO method. The plan was to do a complete release, including customerready documentation and support, every month. Unfortunately, the first release consisted of paper prototypes and the users were not able to provide good feedback. The second release used real code, took six weeks rather than the four weeks scheduled, and EVO was generally thought not to be worth the integration and logistical effort. For this reason, the division decided to abandon EVO.

Although EVO had a bad reputation at the division after the first project, in a smaller follow-on project, one project manager and eight engineers decided to try their own variation of EVO, calling it "phased development." During the first one-month phase, the development team worked from static visual designs to code a prototype. In focus group meetings, the team discussed users' needs and the potential features of the product and then showed a demonstration of its prototype. The excellent feedback from these focus groups had a large impact on the quality of the product.

After the second cycle of focus groups, the feature set was frozen and the product definition complete. Implementation consisted of four-to-six-week cycles, with software delivered for beta use at the end of each cycle. The entire release took 10 months from definition to manufacturing release. Implementation lasted 4.5 months. The result was a world-class product that has won many awards and has been easy to support.

The success of phased development for this second product led to the use of a similar process in the second release. The project manager concluded that the phased development process was the best approach for projects with an aggressive, user-driven schedule. Team experience and confidence were definite contributing factors to the product's success, and a compelling product vision proved to be absolutely necessary.

Several potential issues arose during the project. EVO can add overhead, particularly in small one- or two-person components. This is mainly because of the need for rapid context switching between various activities. Another potential problem is the amount of time consumed by evaluation. The team is investigating how to make evaluation feedback more timely. A third issue is the need to schedule enough time for front-end activities like design and inspections. Scheduling longer evaluation cycles at the beginning of a new release could accommodate this, as could setting aside intermediate cycles for design, inspections, and code cleanup.

The project postmortem listed a number of benefits from using EVO. The team particularly liked seeing the results of their work often. Other benefits included:

- Long-term vision broken into short-term steps
- Prioritized implementation within component teams
- Cross-functional, empowered component teams (decision making pushed down to the project engineers)
- Early results—good communication tool inside and outside the division
- External customer feedback
- Six-week planning at the system level
- Excellent for incremental improvements to existing products
- Early realism about how much can be done.

A New Platform

The last project described in this article involved one project manager and eight engineers from HP's Microwave Instrument Division. It was a firmware project to build a platform for developing new products.

Since a reusable platform design was a new way of working for the division, the EVO method was expected to get more visibility (via frequent delivery dates) for construction of the prototype. The platform team expected to get good feedback from the teams developing the follow-on projects. The project manager had strong reservations about using EVO and about being able to produce a verifiable slice of the platform architecture in six weeks. The project manager decided to give EVO a try even if it seemed that it would take ten weeks to complete the feasibility demonstration work. The team completed the bulk of the feasibility work six weeks into the project and finished it all by the ninth week.

Some engineers initially struggled with breaking their work down into two-week chunks. Eventually, they not only learned to do it but saw some real value in doing it, such as getting better estimates so they could meet their commitments and handle coordination and linkages with the rest of the team.

The team members also benefited from using EVO to develop their software development environment. Because of the improved infrastructure, the project team was able to train new engineers quickly on the new platform development paradigm. The project manager reported much greater insight into the progress of the team and felt better able to manage the project. EVO helped to uncover key issues early and focus attention on the right things. One-third of the way through the

project, the team was able to verify and meet their first performance goals. Traditionally, this didn't happen until at least halfway through a project.

Unfortunately, after completing more than half of the planned cycles, the project was cancelled because of a shift in the division's short-term R&D strategy. The lab currently plans to use EVO in two new projects.

Critical Success Factors

Based on accumulated HP experience in evolutionary development, including the three division experiences described above, we have compiled a list of critical success factors. Since not every project is suited for evolutionary development, the following success factors provide some indicators for deciding if a project is a good candidate for EVO.

Clear Vision

Perhaps the most critical success factor in using EVO is having a clear and compelling vision of the product. The perceived vision or value of the product is the reason why someone would buy a given product rather than another, or buy no product at all. Whether adding incremental functionality to an existing product or developing major new components or functionality, the project team needs to understand and accept this vision.

This vision will help guide prioritizing and decision making and will make it easier for users and developers to understand why some changes are approved and others are not. The project manager at the Personal Software Division noted that the lack of clear focus for the second version of the product made the project much more difficult to manage than the first release. A clear vision is critical to convergence on a releasable product.

Project Planning

Three factors need special consideration in planning EVO projects. First, managing an evolutionary development project requires substantially more effort than managing a traditional waterfall development project. The contents of each delivery should be planned so that no developer goes more than two releases without input to a release. The goal is to get everyone on the project team developing incrementally. Although it is difficult and time-consuming, the work breakdown structure and dependency information must be done and done correctly.

In addition to more management effort, EVO also requires a fundamental shift in how we think about software development. Traditionally, the first third of a project is spent getting the infrastructure in place before developing any customer-visible capability. This is a problem for an EVO project because EVO requires earlier development of customer-visible functionality to elicit customer response. Delaying customer interaction with a product until the second third of the project is incompatible with this objective.

The solution typically lies in finding some existing code to leverage. Although there is almost always resistance to using this approach, it is usually possible to find something to leverage. If this is not possible, then think about implementing the infrastructure in an evolutionary manner. In any case, the first delivery should be released in no more than two EVO cycles (see Fig. 1) from the start of implementation. The first reason for this is that many of the concerns developers have with evolutionary development are best addressed by doing EVO. Second, if the start of EVO deliveries is delayed too long, the risk that delivery will never happen (until manufacturing release) is increased.

The final planning recommendation is to create a standard development plan that can be used for each cycle. Having the same activities occur at the same time within each cycle helps team members get organized and makes process improvement easier. The activities of three groups should be planned: developers, users, and the group who will make decisions about user feedback. Keeping the cycles short helps keep the developers motivated to make changes in response to customer feedback.

Fill Key Organizational Roles

The unique needs of EVO projects require two additional key project roles to be filled: technical manager and user liaison. Because of the extra management burden imposed by EVO, it is useful to have one of the engineers act as a technical manager. This person is responsible for developing the EVO plan (with the project manager) and keeping track of the progress of the group. The technical manager is key to moving the project forward, resolving daily tactical issues, and handling most coordination activities, with the management team as backup. Typically, technical managers also have development responsibilities, so they tend to understand the dependencies between tasks and can relate more freely to the other engineers.

The other key position, user liaison, trains users, collects their feedback, and communicates changes in the status of the project. The user liaison is the single point of contact for all the users and developers. This person should be the first one to use each delivery to see what has changed and if there are any problems in the code. The liaison's job is to make it easy for users to be involved in the project. A strong user advocate in this role can also contribute to management decisions. Without this role, communication between the developers and users can be haphazard, inefficient, and a major energy drain.

Manage the Developers

Although evolutionary development may seem intuitively obvious, implementing it in a traditional software life cycle environment should not be undertaken lightly. Much of the challenge has to do with managing people. The following steps have also contributed to success at HP:

- Establish a credible business reason for using EVO
- Discuss the method and the rationale for using EVO with the development team
- Ask for feedback
- Develop an initial plan that addresses as many concerns as possible
- Ask the development team to try EVO for a couple of releases and then evaluate future use.

Two major concerns arise in managing developers. The first is a concern that the development effort will degenerate into hacking. To prevent this, the software architecture must be well-partitioned and loosely enough coupled to enable easy modification. This is why object-oriented programming techniques are particularly well-suited to evolutionary development. In addition, one or more persons must be assigned to maintain architectural integrity, and if substantial redesign is required, time must be scheduled. This should be a major consideration in determining if a project is appropriate for EVO methods.

A second concern is that it will be too difficult to make so many releases. If it is difficult to make one release every 9 to 18 months, how much more difficult will it be to release every two weeks? The answer is that when you make frequent releases, you get better at it (if this is not the case, EVO becomes too inefficient). Further, the small chunks in each cycle keep things to a manageable size.

Be aware of a few potential problems that could make managing developers difficult in the implementation phase if not addressed properly. First, users tend to focus on what they don't like, not what they do like. To keep this from being discouraging for the developers, it might help to provide a standard feedback form that elicits "Things I liked" followed by "Things I didn't like." Because many more project management decisions need to be made in EVO, handling decisions can also become a problem. If the decisions are not timely or cause dissension, progress can be delayed. Participatory decision-making techniques have been one solution at HP. Finally, developer overwork or burnout is a potential hazard. Most developers overestimate the amount of software they can write in one or two weeks. While working long hours may seem attractive for a short period, it will ultimately be destructive.

Select and Manage Users

Evolutionary development requires users to exercise each delivery. Many potential users have been alienated in the past by the inability of developers to respond to their feedback in a timely manner. Additionally, users are usually being asked to do a task above and beyond their regular jobs. Consequently, the selection, care, and treatment of the user base is a key issue for an EVO project manager.

The source of the user base is the first issue to address. External customers (through field organizations), internal customers, marketing or field people, and temporary workers have all been used successfully to test products. The closer the project team gets to external customers, the more accurate the feedback tends to be, but the more difficult the customer-relations situation becomes. Several projects satisfactorily used internal surrogate users for early releases and then shifted to external customers.

The user group should have a mix of customers that are representative of the target market. The group must be big enough so that one person doesn't skew the results, yet not so big that managing users overwhelms the project team. Among the user expectations that need to be set are:

- Time commitments to use the product and give feedback
- The possibility of critical problems with the software
- The possibility that the software may or may not change substantially during the project
- Prohibition against discussing the software with anyone outside the project.

If the user is an external customer, the field organization must also be comfortable with their involvement.

In addition to setting expectations correctly, keeping users satisfied during the development process is the other main challenge of managing users. An obvious way to keep users happy is to give them code that works reasonably well. If the code keeps failing, they will get frustrated and tend to stop using it. A second key to customer satisfaction is to take their comments seriously and let them know what changes resulted from their feedback. If a suggestion can't be implemented, explain why to them or, better yet, have one or more of the users involved in the decision-making process. Finally, streamline the software distribution and feedback collection process. Find out what mechanisms customers like to use for installing the software and providing feedback. Then accommodate those desires as much as possible.

Shift Management Focus

Traditional software project management focuses 95% of the team effort on shipping code. With EVO, it is important to focus attention equally on all three components of the process, as shown in Table I.

Table I
Management Focus during Traditional and EVO Life Cycles

Activities	Traditional	EVO
Shipping Code	95%	33%
Getting Feedback	2.5%	33%
Making Decisions	2.5%	33%

Because of the need to radically shift the focus of all involved, getting feedback and making decisions in the early part of the project should be emphasized. Putting a lot of structure around those two activities by doing such things as scheduling regular meetings to review feedback and make decisions will help ensure that they get done. These two activities are prerequisite to getting real value from EVO.

Manage Builds

To do evolutionary development, a project team must have the ability to construct the product frequently. If the product will be released every two weeks, developers should be able to do a minimum of one build per week, and preferably a build every other night. The engineers must be able to integrate their work and test it, or they can't release it. Code that is checked into the configuration management system must be clean, and the build process itself must run in 48 hours or less. Identifying a build engineer or integrator can help the process.

Focus on Key Objectives

While there are many reasons to use evolutionary development on a project, focusing on one or two critical benefits will help optimize efforts. These goals will guide later decisions such as how to structure user involvement, how to change plans in response to user feedback, and how to organize the project. Regardless of what goals are focused on, it is critical to communicate the reasons for strategic decisions to both management and the development team.

Evolutionary development is a different way of thinking about managing software projects. Most groups will probably experience some of the pain that usually accompanies change, so start with a small pilot project first and then try a larger project.

Conclusion

The evolutionary development methodology has become a significant asset for Hewlett-Packard software developers. Its most salient, consistent benefits have been the ability to get early, accurate, well-formed feedback from users and the ability to respond to that feedback. Additional advantages have come from the ability to:

- Better fit the product to user needs and market requirements
- Manage project risk with definition of early cycle content
- Uncover key issues early and focus attention appropriately
- Increase the opportunity to hit market windows
- Accelerate sales cycles with early customer exposure
- Increase management visibility of project progress
- Increase product team productivity and motivation.

The EVO method consists of a few essential steps: early and frequent iteration, breaking work into small release chunks, planning short cycle times, and getting ongoing user feedback. Other components can be modified to accommodate the needs of specific projects, products, or environments. Examples where situation judgments are appropriate include selection of users and length of cycles.

Additional activities, like establishing a clear product vision, identifying a technical manager and user liaison, creating a standard development plan, and setting correct user expectations, will help optimize the benefits of using EVO.
The challenges in using EVO successfully are mostly, but not exclusively, human resource issues. These include the shift in thinking about a new project structure paradigm and perceptions that EVO requires more planning, more tasks to track, more decisions to make, more cross-functional acceptance and coordination, and more difficulty coordinating software and firmware development with hardware.

As noted earlier, many of these perceptions are valid but have extremely advantageous cost-benefit trade-offs. Since many software developers are no longer primary users of their products, they now need to be able to understand the primary users' needs, skill levels, and motivations. Finally, major changes in the customer-developer relationship can result in customer demand for more input and involvement in product definition and design.

HP is continuously improving the EVO process, building on our experience at different divisions. The software initiative team now offers a workshop and consulting expertise on the EVO method. Experience with the value of using EVO to develop the infrastructure and the need for management focus have framed recent implementation efforts.

The key lessons to remember when first attempting EVO are to start small, keep good records, and be diligent about doing the essentials.

Acknowledgments

The authors would like to thank Mike Teska at Manufacturing Test Division for his support of software development method improvement, the board consultant development team consisting of Sharon LaTourrette, John McDermid, Bob Illick, Kathy Withers-Miklos, and Charles Zeng, and the users of the board consultant EVO releases, especially Kent Dinkel. We'd also like to thank Todd Cotton and Beatrice Lam for discussing with us their EVO experiences on some HP projects, and Tony Dicolen and other members of his team at the Microwave Instrument Division for sharing their EVO experiences with us. Finally, we'd like to thank the other development teams within HP that have shared their EVO experiences with us and the software initiative management for sponsoring this work.

Reference

1. T. Gilb, *Principles of Software Engineering Management*, Addison Wesley Publishing Company, 1988.

▶ Go to Article 5
▶ Go to Table of Contents
▶ Go to HP Journal Home Page