Rapid Prototyping: A New Philosophy for Application Development

by Arthur B. Smith

Rapid prototyping (RP) is one of the buzzwords of the nineties, there's no denying it. Like many of these buzzwords, it's not really a new concept, but one that has finally found its time in the sun. So, what is rapid prototyping? It is a development methodology; that is, it is a way of approaching the design and implementation of an application (or any other "made" thing, for that matter). It is not a specific tool (like a debugger, or GUI builder), it is not a specific language, it is not a technique to plug into an existing system—it is a whole new philosophy of development. Some tools will help development in a rapid prototyping environment, and some languages (such as M!) are well-suited to rapid prototyping. There are definitely techniques that are helpful, but they alone do not give you the new methodology.

Traditional "Waterfall" Development

Traditionally, application development has followed the "waterfall" methodology shown in Figure 1. In this process, the product moves through a number of specific stages. The product is only in one stage at a time, and each stage must be completed before the next is started. There are numerous software engineering standards for this methodology, each with its own set of stages and with carefully defined deliverables to be produced at each step. In normal development, the product flows smoothly down the waterfall from Initial Concept through each of the stages until finally reaching Deployment.

Of course, things don't always work so smoothly. Problems are found at each step, and these problems may cause the project to move back to a previous stage, possibly backing up more than one stage. This is always an unfortunate experience, as the product must revise all the deliverables for each of the previously completed stages before it can again move forward. The farther back the project must be taken, the more it will cost in time and effort. Thus, there is a lot of pressure to find the problems early and avoid these expensive mistakes. When the mistakes are found, there is often a lot of finger pointing, back stabbing and other invidious aspects of office politics.

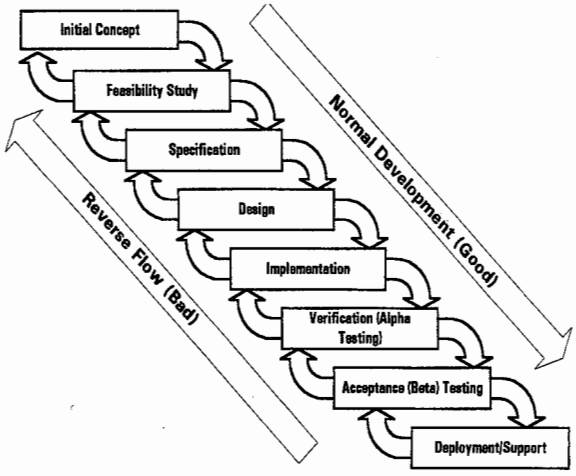

Fig. 1 Traditional "Waterfall" Design Methodology

Rapid Prototyping Development

Rapid prototyping, on the other hand, does not rely on this stepwise progression from initial concept to released product. Rather, it is based on an evolutionary response, in which the "problems" of the waterfall approach become triggers for a new evolutionary cycle. This is an important aspect of the RP philosophy: removing the negative connotation from what is traditionally considered "failure."

Soichiro Honda, the founder of Honda Motors stated this very well: *"Many people dream of success. To me success can only be achieved through repeated failure and introspection. In fact, success represents the one percent of your work which results only from the ninety-nine percent that is called failure."*

Notice that "failure" (for lack of a better term) loses that bad quality. It is no longer a thing to be avoided, rather it is a necessary component for success. Some people have even described rapid prototyping as the "Method of Fast Failures." Tom Peters, noted business writer, states *"There is an almost irreducible number of failures associated with launching anything new. For heaven's sake, hurry up and get them over with."*[1] This is the essence of rapid prototyping.

The rapid prototyping evolutionary cycle is shown in Figure 2. Note that there is no backward movement with the accom-

panying nasty connotations that we saw in the waterfall methodology. Rather, each cycle through the evolutionary loop refines the product until the analysis phase determines that the product is ready for deployment.

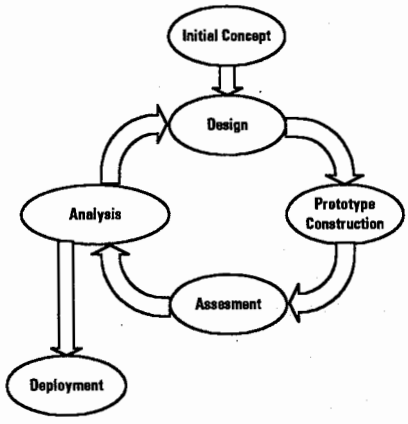

Fig. 2 Rapid Prototyping Methodology

This does not mean that there are no feasibility studies, specifications, formal designs, alpha test regimens, etc. All the meat and potatoes deliverables of the waterfall methodology are present, though they may not be labeled as such. Rather, each of these evolves along with the product. Some of these aspects (such as the feasibility study) come into being in the early cycles and probably change very little in the later cycles. Others, such as testing regimens, probably don't exist at all during the early stages and may change quite a bit during the later cycles. Thus the net effect is similar to the waterfall methodology.

Customer/User Centered Development

An important aspect of rapid prototyping is not shown in these diagrams, however. In the traditional waterfall methodology, the deliverables of each successive stage are sent to an ever-widening audience. Early-phase deliverables probably go no farther than the development team. Potential customers or users (i.e., those who will use this product or application, be they internal or external) never see anything until the Acceptance (Beta) Testing stage. In rapid prototyping, however, the customers are (or should be) involved in every cycle through the evolutionary loop. The Assessment step is performed by these potential customers in every cycle from the very start. Their assessment is then reviewed, and changes are planned by developers, working with the customers, in the Analysis step.

This customer-centered development is critical to the methodology. It helps to guarantee that the product matches the customers' needs and expectations, and just as important, it helps to guarantee that the customers' needs and expectations match the product. These may sound like the same thing, but they are very different. The waterfall methodology tries (often unsuccessfully) to achieve the first part, but does nothing to achieve the second.

It is clear that keeping the customers involved throughout the development process will make the product better match their desires. They will have the opportunity to shape the product to meet their needs, to add their input to the inevitable difficult and critical design decisions, and to provide an early end to misdirected development. The finished product will benefit from the continuous refocusing of the development process to meet the customers' desires.

Less obvious perhaps, is that the customers' needs and expectations will better match the product. Here we speak specifically of those customers who participate in the assessment of the prototypes. In a small internal development project, this may be most or all of the potential users. In a large-scale marketed project, this can be only a sampling of the potential customers. These customers, however many or few they are, have "bought in" to the product in a way never seen using other development techniques. These customers will believe that this product is better than any other—thinking anything else would, after all, reflect badly on their own input! They will gladly share their views with other potential customers and will gladly champion the product because they are now part of the development team; this product is "their baby."

The psychological advantage of this customer buy-in cannot be overemphasized. In an internal development project (one used in-house only), careful selection of the assessment team can virtually guarantee a favorable response from the top down. In an external project (one marketed to others) careful selection can provide a source of marketing testimonials and a solid initial customer base with good exposure to other potential customers. Either way, this customer participation and buy-in is an invaluable asset.

Two other aspects of rapid prototyping which are advantages over traditional development are the more rapid production of visible results and an increased ability to have the project come in on schedule. Because the whole idea of rapid prototyping is to continually refine prototypes, it follows that these prototypes will appear earlier in the development sequence. In fact, they do appear quicker, because the product specifications and design need not be completed first—they are, after all, refined by the assessment and analysis of the prototypes. This can ease pressures from management on developers , as progress can be easily demonstrated even at early stages.

Because the customers are participating in the entire design and construction of the project, time can often be saved by immediately halting work on misunderstood specifications

and avoiding costly rework. Furthermore, by having management participate in the evolution of the project, it becomes much easier to bring the project in on schedule. When the inevitable additional specifications or unforeseen delays creep in, management can take action knowledgeably, either by extending the timeline or by cutting requirements. Since they are participating in the development, they have personally invested in the project and will help to insure its appropriate timeliness.

The Down Side of Rapid Prototyping

Like everything, rapid prototyping has its failings as well. Because the design and implementation of the project is spread over a longer time period and is done before full specifications are available, it becomes harder to recognize modules which can be pulled out and used repeatedly. Developers must be especially careful to watch for opportunities to reuse work and properly modularize their programs.

The ad hoc nature of rapid prototyping also tends to lead to more "kludgy" programming. There is a temptation to write sloppy programs because "it's just a prototype." Unfortunately, there is no good way to predict what prototypes will be thrown away or thoroughly reworked and what prototypes will remain intact in the final product. It is important to keep excellent internal documentation at every stage, and it is essential that external documentation, specifications, and design notes be maintained in an organized fashion to counteract the inherent disorganization rapid prototyping brings.

Another possible down side is the inability to accurately predict development time. Since it is impossible to know the number of times through the prototype cycle that will be required and the amount of rework required each time, it is likewise impossible to predict the development time. Of course, most developers have found that time predictions based on the traditional waterfall approach do little better, and as we have already explained, rapid prototyping tends to bring projects in closer to schedule. It is difficult to assess just how much this difficulty is a real detriment to rapid prototyping.

One last drawback is rapid prototyping's inability to do formal verification. In software engineering, verification is the step following implementation which confirms that the project, as implemented, matches the project as specified. In rapid prototyping, the specification evolves through each cycle of the development (some may note that this is true in any methodology, but is deliberate in rapid prototyping and backsliding in the waterfall method). Thus, verification is just a check that you properly maintained the specification as well as the product. Since the end result of this evolving project, however, is increased customer satisfaction, it is arguable whether or not this inability to perform formal verification is a detriment to rapid prototyping.

Requirements for Rapid Prototyping

Rapid prototyping is not a simple variation on traditional program development. It represents a true paradigm shift, and as such, brings with it some new requirements for the developers. One of the most important requirements has been described by some as "egoless programming." If one listens objectively during the assessment phase, occasionally the customer will be heard to say, "This is junk!" Don't take it personally, it happens to everyone.

But more importantly, don't try to defend it—don't try to convince the customer that you know what he wants better than he does (even if you think you do). The goal is to produce what the customer desires. If he thinks your prototype is junk, then it is junk. Throw it out, lock stock and barrel, and start over. Don't let your ego get in your way.

Akin to this, and just as important, is the ability to really listen to your customers. It is essential that the developers be able to have a productive interaction with the customers during the assessment phase. Training in communications skills and "active listening" is very helpful here. Some customers will know exactly what they want and will have no problem directing the developers appropriately. Other customers are unsure of themselves, or naturally shy, or have difficulties communicating. The developers must be able to draw an honest and useful assessment out of these customers without unduly influencing them or hearing only what they want to hear. Developers who chose programming so they wouldn't have to interact with people (and we all know some of these) have some unlearning to do before they can be effective in the rapid prototyping methodology.

A third requirement for developers is the need for good discipline. Rapid prototyping can quickly degenerate into an ad hoc free-for-all. It is essential that there be good shop-standards to reduce the tendency to produce "kludgy" code; to force the programmers to throw bad code out and start over rather than applying patches on patches; to keep the documentation (including specifications, design notes, internal and external documentation) up-to-date; and to thoroughly document the assessment and analysis steps to avoid treading the same ground over and over. Unlike the waterfall method, the discipline is not built in to the system in rapid prototyping. It falls on the developers, then, to maintain an appropriate level of discipline.

A final requirement is good prototyping tools. It must be possible to develop front-end (user interface) code with little or no "guts" behind it in the initial stages. Subsequent cycles must be able to add the application code and underlying database without requiring rework of the front end. Additionally, it must be easy to discard or change sections of

the application without affecting the rest of the application.

The M Connection

There are a number of aspects of M that make it particularly well-suited to a rapid prototyping development system. One of the main features is that it is an interpreted language rather than a compiled language. This allows sections of the system to be changed and tested immediately, without recompiling (or even re-linking) the entire system. In addition, there is no need to create module "stubs" as there often is in compiled languages. It is not an error in an M program to have references to routines that do not yet exist, as long as you don't execute those references.

In addition, M allows data structures to evolve along with the application. If it becomes necessary to add nodes to a data element in M, one simply does so. In strongly typed languages this will typically require a recompilation if the data structure changes size, unless it is completely handled by pointers (and sometimes even then). This ability to simultaneously grow the code and data aspects of the routine make M particularly well-suited to rapid prototyping using object-oriented design principles.

As a last point in M's favor, it is well-known for allowing rapid development of code. The reasons for this are complex and debatable, but the results are consistent—M allows rapid code development, which is necessary for rapid prototyping.

As with all things, M's very strengths can also be weaknesses. The ability to "grow" a routine or a data structure can often lead to applications that are unmaintainable. It is particularly important in a relatively unstructured language like M to carefully review the application (code and data) at each iteration, and make sure it is soundly designed, well-documented, and maintainable. It may often be necessary to throw an iteration out, not because the users found it unacceptable, but because the application has become unmaintainable. In this case, the next iteration should attempt to exactly reproduce the previous iteration (except for aspects that were deemed undesirable) but with a sound maintainable design.

Conclusion

Rapid prototyping represents a radical shift in development methodologies. A main feature that characterizes rapid prototyping is a recognition that "failures" are bound to happen and that they provide information crucial to development. These so-called failures are actually positive events and should be encouraged so long as the end result is an improvement in the design and implementation of the project.

Another characteristic of rapid prototyping is customer/user involvement throughout the development of the application. The only definition of quality that counts is the customers', and the only way to learn that definition is to allow the customer to evaluate and assess your work and to listen and act responsively and responsibly. If your oh-so-clever application does not match the customer's oh-so-perverse definition of quality, it is a failure. The sooner you learn this failing, the better you are able to adapt to it and produce an application which truly satisfies the customer.

Everything else about rapid prototyping derives from these two points. The techniques, the necessary skills, the advantages, and the disadvantages all stem from a desire to benefit from failures often and early and to involve the customer in the entire development process. As Tom Peters (again from "Thriving on Chaos") states: "Complexity + Need for speed = Make more mistakes (or else!)" Join the rapid prototyping revolution. Get out there and fail!! M

Reference

[1] T. Peters, "Thriving on Chaos", Harper and Row, 1987.

Art Smith is active with the MDC, works at the Veterinary Medical Teaching Hospital of the University of Missouri, and offers consulting services through Emergent Technologies. He may be contacted at (573) 642-8802, or Emergent@gnn.com.

MTA Europe Annual Conference

The MTA-Europe annual conference is provisionally scheduled for Friday, the 13th of December, 1996 in Calais, France.

Anyone interested in contributing to the conference program is invited to register their interest, with brief details, by email to: mtae@georgejames.com.

Further details concerning the conference program can be viewed at: http://www.georgejames.com/marina/mtae.