

Prototyping

Systems Development in Record Time

Fred Nickols

© Fred Nickols 2009
All rights reserved



www.nickols.us
fred@nickols.us

This article describes a project led by the author and carried out at Monarch Financial Services in the late 1980s. The approach to the project was referred to at the time as “prototyping” and the author still uses that term to refer to the approach. Many of the elements of what is here called “prototyping” are found in what has become known as “agile software development.” Indeed, the project in question was probably an early instance of agile software development. In any case, what was learned on that project is still instructive and still of value to those who must develop systems in record time.

Executive Summary

Prototyping is frequently cited as a desirable step in many developmental efforts, not just those that are concerned with computer-based systems. Yet, the inner workings of the prototyping process remain a mystery to those who have never participated in such an effort. In this article, the author, who has led several prototyping efforts, shares his insights into the process of developing a prototype and his beliefs about what it takes to make that process work. What it takes is "a monomaniac with a mission." The key is intense focus. The chief benefit is systems development in record time.

The Practice of Prototyping

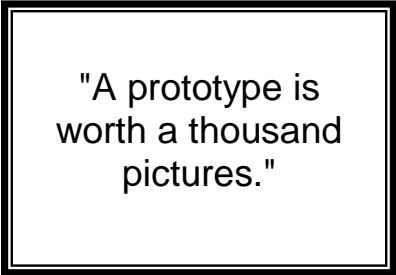
Prototyping refers to the activity of developing a prototype, and to the use of a prototype as part of a deliberate effort to simplify, speed up, and reduce the costs of development. The concept of a prototype or a working model is not new. Automobile designers have used it for years, as have aircraft designers. In these arenas, prototypes are typically referred to as "mock ups." In these cases, a prototype doesn't necessarily have to work; it just has to look good. Of more recent vintage is the notion of prototyping as a deliberate systems development strategy. Because of its comparatively recent origins, the practice of prototyping is not as well understood nor is it as widely accepted as the more conventional approach to systems development. And, the practice of prototyping is anything but standardized.

A Prototype is A Working Model

The essential concept to grasp is that a prototype is a working model. It might or might not be 100 percent complete, with all the bells and whistles that are desired, it might or might not be blessed with full functionality, and it most likely will have a glitch or two. But, essentially, it works as intended, and it typically works well enough to be used in a production environment, even if only temporarily.

The purpose of prototyping is to shorten development times and to reduce the costs of development. The fundamental premise on which prototyping proceeds is that a working model provides a much clearer picture of the system to be developed than an entire library full of user requirements, system specifications, data dictionaries, functional flowcharts, and memoranda disclaiming responsibility or pointing figurative fingers. To paraphrase Confucius, a prototype is worth a thousand pictures.

Personal computers (PCs), with their self-contained computing environments, offer the ideal prototyping mechanism for many business system applications. Their high-level languages simplify and make easier the programming task. Their ever-increasing computing wallop enables the tackling of non-trivial information and data processing tasks. PCs hold no mystery for many users; indeed, they are often useful in dispelling the mystique surrounding computing. Significant



"A prototype is worth a thousand pictures."

changes are sometimes made in minutes, often in no more than a few hours. This generates real interest and involvement on the part of users.

Collapsing lead times for system-supported products and services means that these products and services can get to market faster with less up front investment. Indeed, some systems are developed so inexpensively they are labeled "throw-away" systems. Regardless of the eventual disposition of the prototype, the objective of prototyping is always the same: to get on with it, to bring up the system in short order and use this quickly-developed version as a focal point for getting clear about requirements in a way that words can never convey and documents can never capture. And, if necessary, the prototype can be used as an early version of the production system.

The Advantages of Prototyping

There are several advantages associated with prototyping. Among them are speed, flexibility, simplicity, ease, and lower costs of development. Two examples from the author's consulting practice will provide some illustrations.

At Community Mutual, the Ohio Blue Cross & Blue Shield plan, three people developed a complete medical claims entry system in 90 days. This system included all the edits necessary to ensure a "clean and complete" machine processable claim. It also included a nomenclature to code look-up system for medical procedure (CPT 5) and diagnosis (ICD 9) codes, as well as a batch transfer program for sending records to the mainframe. The team of three consisted of a consultant, a programmer, and a trainer of claims examiners. The cost of developing this system was approximately \$100,000. It was never used in production but it did form the basis for the subsequent design of a major modification to the mainframe system.

At Monarch Resources, (later Monarch Financial Services, Inc.), a team of five people developed a complete, full featured variable life insurance administration system in 89 days. This system included new business, underwriting, and policy service functions. The core members of this team consisted of a consultant, three programmer analysts, and an actuarial assistant. The cost of developing this system was roughly \$250,000. This system was used to support the roll out of a new product and, although much modified since then, it is still in use.

Interestingly, the speed with which these prototype systems were developed proved problematic in a very strange way. To this day, there are people who refuse to believe the systems in question were developed in the stated time frames. It is instead the belief of some that the development of these systems was undertaken months before, in secret, and the systems were then later unveiled at an opportune moment.

To understand and appreciate the speed with which a prototype system can be developed, it helps to understand how the prototyping environment is created and maintained.

To Create and Maintain A Prototyping Environment...

Rely on Small Teams

A small team is exactly that, from three to five core members. This provides a very flat and a very lean structure. It is also conducive to shorter, more direct, and hence much faster communication channels. Besides, having too many people simply means they will get in each other's way. Contrast, for example, the small software development teams used at Microsoft with the "masses of asses" approach IBM is accused of using.

Exploit the Inherent Productivity Advantage of High Level Languages

The high level languages associated with PCs relieve programmers of many of the more tedious and troublesome aspects of programming (e.g., file handling routines). The use of these high level languages contributes greatly to improved speed, ease, and productivity of the prototyping effort. They are their own form of CASE tool.

Do It Now, Document It Later and Make the Users Do It

The prototype system can be documented after the fact. This is not as cavalier as it might sound. It's pointless to document a prototype unless and until it's up and running. The high level languages used in prototype development result in very readable and understandable source code and require less systems documentation. Moreover, the people who build a system are not necessarily those who are best suited to the task of documenting it.

Documentation is not prepared for the people who develop a system. They already understand it. Documentation is prepared for the people who don't understand the system. System documentation should be prepared by the intended users; in other words, by people who are unfamiliar with the system but who have a stake in the quality of the documentation from the users' perspective. The kinds of things they'll try to find out during the course of developing the documentation are precisely the kinds of things others will want to know, and the explanations they provide will be more understandable. Explanations offered by experts rarely make sense to the uninitiated.

More important, if the prototype is truly a prototype, that is, a prelude to some larger effort, then the source code of a working prototype constitutes the best possible set of system specifications one could have. Logic, calculations, relationships, screens, database structures, arrays; all these and more are present in the source code of a working prototype. I have on more than one occasion seen business analysts take six months and more to produce a set of specifications that subsequently proved useless. The source code of a working prototype is hardly what one could call useless. As a matter of fact, "gussying up" the prototype can yield a working production system. Document it, train the users, and it's done.

Foster Intimate User Involvement and Contributions

A lot of lip service is paid to the notion of user involvement, but it is rarely a reality. Frequently, the users are too busy with their own problems to become really involved, or they sense disdain on the part of developers, or they simply aren't given any really meaningful work to do. It is absolutely essential that at least one highly qualified user or expert be intimately involved in the project, a full-fledged member of the small team. And, it is important to get a real user, not an occasional representative of management. A "real user" is someone who in fact will use the system regularly.

Users can contribute significantly to the productivity of a prototyping effort. At Blue Cross & Blue Shield, Citicorp, Community Mutual and at Educational Testing Service (ETS) I trained claims examiners and financial aid assistants to apply a simplified form of algorithmic analysis. They then developed algorithms expressing the claims adjudication or suspend resolution logic. At Monarch, an actuarial assistant was persuaded to develop a set of plain language actuarial or calculation specifications (as opposed to the usual actuarial notation). The point here is that much of the work of prototype development can and should be shifted to the users and other experts.

Let Specialization Emerge, Don't Impose It in Advance

The division and coordination of labor is a fact of life in any effort involving more than one person. But, in a prototype environment, it is better to let specialization emerge than it is to impose it

in advance. The members of a small team will very quickly figure out who wants and likes to do what and who is good at what. And they will allocate the work among themselves accordingly. The team members also know the task dependencies and interdependencies better than anyone else and will see to the coordination of the work among themselves. Of crucial importance here is the merging of the functions of analyst, designer, and programmer into one role: programmer analyst or system builder.

Force the Use of Plain Language

Systems people have their own jargon. So do users and experts. One of the more important and more difficult tasks in a prototyping effort involves extracting the information buried in various specification and source documents and expressing it in plain language so that all, regardless of their technical specialty, can understand it. (I am convinced that the major contribution I make to most of my projects boils down to translating jargon into plain English.) One approach is to give the team members a common language and a shared set of referents (e.g., the algorithms used in analyzing and documenting claims adjudication and suspense or reject resolution operations). Another is to collect, compile, condense, and synthesize information from a number of different sources into one centrally maintained, easily read, and widely-distributed document. Finally, get in the habit of forcing everyone to say what they have to say in plain language. If they can't, they might not know what they're talking about.

Concentrate Dispersed Knowledge

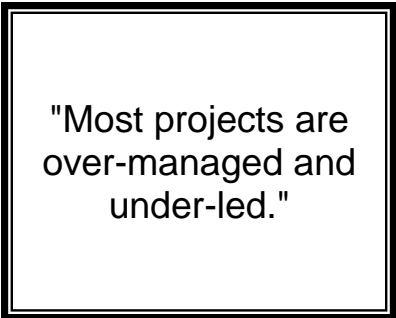
Closely related to the preceding point is the notion of concentrating dispersed knowledge, of bringing it all together. At the health insurers and ETS, this was accomplished via the development of algorithms. Not everyone knew how to adjudicate a given claim or resolve a given suspend. The algorithms served to concentrate this dispersed knowledge. At Monarch, dispersed knowledge was concentrated through the preparation and continuous updating of a product specification document. This regularly updated document represented a compilation and condensation of information about the product that the system would have to support. Its contents were drawn from actuarial specifications, administrative procedures, marketing plans, legal filing documents, and endless cycles of review and revision.

Conduct a Careful Analysis, but "On the Fly" Don't Rely on "Up Front" Studies

The analysis of business and functional requirements must indeed be carefully and correctly accomplished; after all, a system will be built based on this analysis. But, not all the analysis has to be done up front, and not all of it has to be documented nine ways from Sunday. Much of it can and should be done on an as-you-go basis, and by the people who are building the system, not by people who specialize solely in analysis.

Rely on Minimal and Informal Reporting Requirements

Very little time should be spent in reporting on progress and problems and the reports themselves should be informal. The overhead involved in keeping management informed, and at times for no other purpose than to maintain the illusion of managerial control, is extraordinary. In one systems shop with which I was familiar, I determined that this managerial overhead amounted to 20 percent. That's right! A full 20 percent of the shop's resources went into preparing reports for management or in actually reporting to management!



"Most projects are over-managed and under-led."

Rely on Simple Controls and Short Interval Scheduling

Closely coupled with the preceding comment is Peter Drucker's view that controls should be few in number and simple in nature. Also, because prototyping efforts are generally short fused, the formal trappings of managerial control (e.g., plans, budgets, and schedules) all but disappear. They don't go away, actually, they just move upstairs, where they belong. Daily, informal progress meetings are more useful and more appropriate. At Monarch, for instance, the New York-based development team found itself in Springfield, Massachusetts for several weeks during the implementation phase. While in Springfield, I generally took the team to dinner two or three times each week. Before, during, or after dinner, I asked the following three questions, recording the answers on a table napkin:

1. What did you get done today?
2. What are you going to work on tomorrow?
3. What do you need that I can get for you?

Give the Team End to End Accountability

It is important to make sure the team members understand they have end to end or complete accountability for the total system. This goes a long way toward relieving them of anxiety about the extent to which they will be forced to modify the system to satisfy some irrelevant or absurd standard. In effect, the team reigns supreme in matters of design and architecture. This necessitates being willing to protect the team when it runs afoul of the established order.

Staff the Team with People Who Don't Know Why "It" Can't be Done

At all the companies where I've led prototype efforts, the system developers were new to the business. They were also young and ambitious. They thought they could do anything and, being new to the business, they knew of absolutely no reason why it couldn't be done. This fresh perspective is essential. It promotes a "can do" attitude, it eliminates the biases of past experience, it leads to all kinds of very "dumb" and very useful questions, and it absolutely prevents the phenomenon known as "paving the cow paths."

Set Up a "Bull Pen"

Central to the success of a prototype operation is clear, direct communication. One physical way of promoting this kind of communications environment is to set up a bullpen, an open workspace housing all the members of the team. This also promotes informality, coordination of effort, and rapid resolution of bugs, snags, glitches, and other developmental problems. In a word, it promotes teamwork.

Allow No Secrets

It is essential to the success of a prototype effort that there be no secrets among the members of the team. Secrets are divisive. Everyone involved can and should know anything they want to know about the project, and anyone can and should be able to go anywhere and talk to anyone about anything. When people start squirreling away information the project is in trouble.

Maintain Flexible Priorities

Things change. Information thought to be available isn't. A day scheduled with a user gets re-scheduled. New functionality requirements are uncovered. On and on the changes go. The developers must have the ability to drop what they're doing and tend to something more important or more pressing. The general principle here is that you work on what you can work on when you

can work on it. Hence, also, the general avoidance of formal and rigid plans and schedules; the situation is simply too fluid and fast changing for those artifacts of a more stable and ordered world.

"Hole Up," Off Site, If Possible

It is important to get prototyping projects out of the mainstream. Set up a "skunk works." Avoid politics, procedures, and protocol. Shelter the prototype team from the day to day demands of the production environment. Take them off site if at all possible. Interruptions are the kiss of death. Focus and concentration are vital to success.

Rely on Leadership, not Management

Pick your team leader carefully. Don't confuse him or her with the project manager. Don't bother with a project manager; good people manage themselves. They have to; they're knowledge workers and no one else can supervise them. They have to supervise themselves. Far too many developmental projects are over-managed and under-led.

Keep the Pressure On

There is very little slack in a prototyping effort. Deadlines, in addition to being tight, offer an interesting way of keeping the pressure on. Let's suppose a deadline is drawing near and it's clear that the project won't be completed on time. Let's suppose, also, that the deadline was unrealistic to begin with. When the deadline arrives, find or invent some reason for extending it. But, make sure the extension is for a short period and make sure the reason given has nothing to do with the fact that the work couldn't be completed in time. This technique allows management to insist that the deadline is very, very real (which it is), it does not diminish the general utility of deadlines (which would happen if the deadlines were not taken seriously), it keeps the pressure on until the very last moment and then provides only a slight breather, and it affords the project leader the opportunity of coming up with creative and imaginative reasons for extending a deadline that was not supposed to have any give in it. On one of the two projects cited here, the initial deadline was 30 days from the commencement of the project. When this preposterous target was proposed, I shrugged, smiled, and said, "Sure, why not." I knew the deadline was unrealistic and that it would become a moving target. The real deadline might have been phrased as follows: "As soon as humanly possible." That deadline we met.

Involve the Training People from Day One

Training people who are "up from the ranks," typically have a very good understanding of the work of their organization. They also know their way around. And their network of contacts is unparalleled. It is a wise move to involve them from the very beginning.

Do Your PR Work

Prototype efforts can incur suspicion, arouse resentment, and engender hostility. It is absolutely vital that the rest of the organization understand the aims of the prototype effort, and be sold on supporting it. Good PR work can go a long way toward "greasing the skids" of acceptance.

Make Sure You Have Friends in High Places

A prototype effort generally flies in the face of accepted practice and is sometimes seen as flouting it. In short, many prototype efforts are counter-culture. For obvious reasons, prototypes are threatening to some. For these and other reasons, it is essential that someone at the highest levels assume the responsibility for protecting the prototype effort. In short, PR isn't enough; a "champion" is needed.

Beware the "Retro-fitters"

Even if you succeed, you're not out of the woods yet. Beware the "retro-fitters." Retro-fitters are the people who want to put things back the way they were. There are two types of retro-fitters: system retro-fitters, and operations retro-fitters. System retro-fitters want to impose the old approach to systems development on future prototyping efforts. In at least one case I know of, they succeeded, and all the advantages of prototyping were lost, leading the CEO to ask, "What the hell happened to the savings?" And, more than a few of the users would like to see things put back the way they were. This class of retrofitter typically tries to get you to modify the new system so it will behave like the old one.

Be A Gracious Winner

If you do make it all the way through to the end, don't celebrate too loudly or publicly and, above all else, don't gloat. If you do, you'll trigger all kinds of envy and resentment and the retro-fitters will come swarming out of the woods.

Why Prototyping?

We all are faced with broad-based concerns: a rapidly changing economic environment, an ever-accelerating rate of technological change, and intensifying global competition. In response, we must be able to rapidly roll out new products, and even more rapidly develop the systems necessary to support them. In many organizations there is a requirement to get systems development off the critical path. Prototyping offers one way of accomplishing this. That a prototype can take only one tenth the time and cost only one tenth as much as other approaches is merely icing on the cake. It's the comparative speed with which products and services can get to the marketplace that adds real value.

Some Closing Thoughts

After reviewing the body of this article, I am reminded of two more, very basic points. One is something I wrote several years ago:

- "The fundamental task of management is to concentrate and channel organizational energy along productive lines."

The second point appears in Peter Drucker's *Adventures of A Bystander*:

- "Whenever anything is being accomplished, it is being done, I have learned, by a monomaniac with a mission."

The gist of both these statements can be summed up in a single word: Focus. I believe successful prototyping efforts are accomplished not as a result of any special or magical methods or techniques, but simply by eliminating most of the wasteful activity and the many obstacles and barriers that so often get in the way in established and successful organizations. I also believe it takes "a monomaniac with a mission" to bring focus to prototyping efforts and to knock down the barriers such efforts inevitably encounter.

Afterword: Crack the Tough Nuts First

Long after I published this piece it occurred to me that I had left out an important point, namely, the notion of "cracking the tough nuts first." In the prototyping effort at Monarch, the tough nut to be cracked first was a very complex funds allocation requirement. Within a few days, we knew that the requirement as stated couldn't be accomplished. However, we could come very close and the client accepted our approach.

Why crack the tough nuts first? Why not save them for later? Why not get some easy wins or gather the low-hanging fruit up front? Well, here's my thinking on that score.

If you crack the tough nuts first, it's a downhill run from there; the hard stuff is out of the way and confidence levels shoot up in all quarters. If you can't crack those tough nuts early on, you run the risk of getting halfway or close to the end only to discover that you can't get there. It might be counter-intuitive but it makes good sense to crack the tough nuts first.

Contact the Author

Fred Nickols can be reached by e-mail at fred@nickols.us. Other articles of his can be found on his web site at www.skullworks.com.