Hold Me, Thrill Me, Kiss Me, Kill Me[1]

Patterns for Developing Effective Concept Prototypes

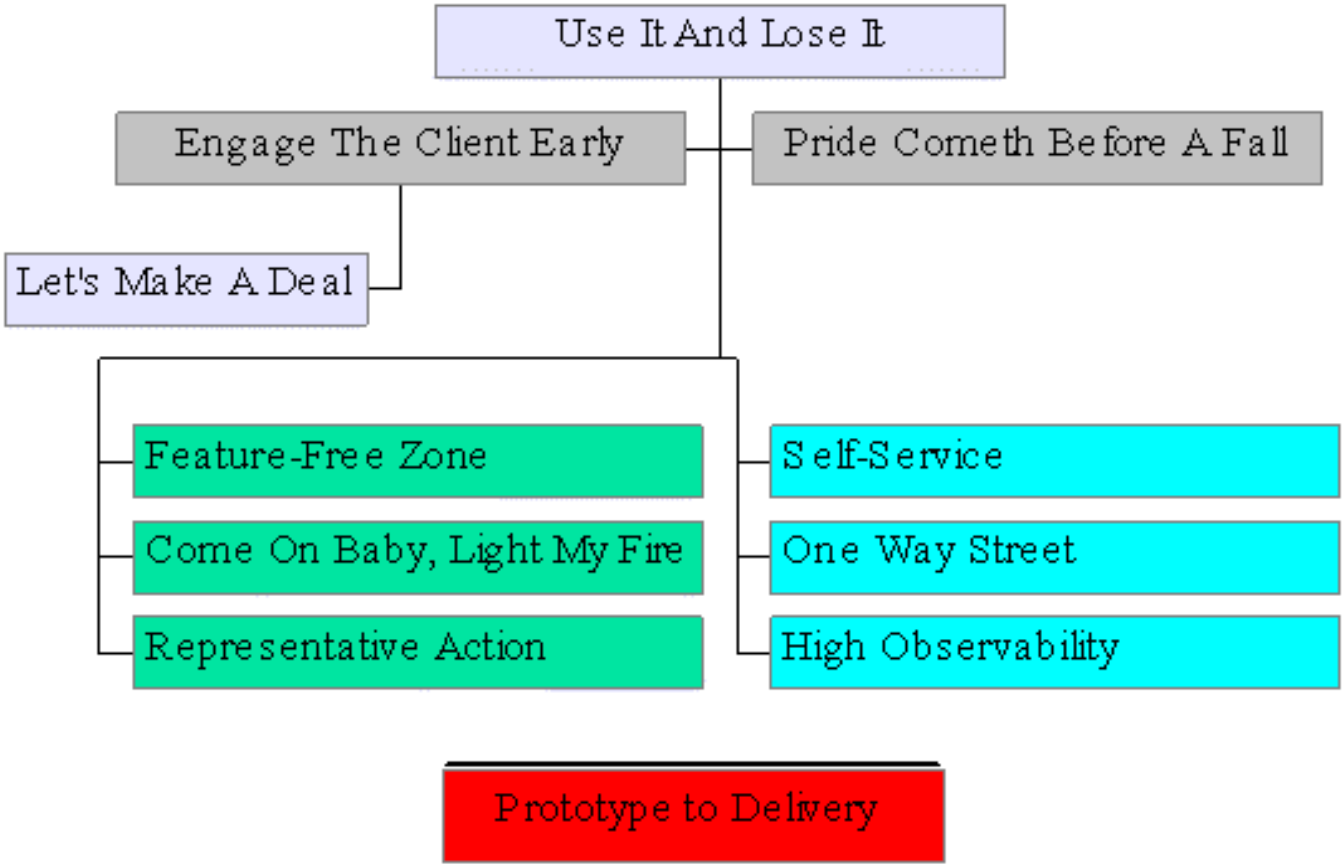
Carol L. Stimmel
Broadband Innovation Group
MediaOne Labs
10355 Westmoor Drive, Suite 100
Westminster, CO 80021
CStimmel@mediaone.com

Abstract

The following collection of patterns and one anti-pattern target developers and their managers who are involved in creating and demonstrating concept prototypes and using the ubiquitous GUI-development tools to make them (such as Visual Basic, Delphi, and Web-based development tools, like Sapphire Web). With an increasing emphasis on quick product turn-around and fast to-market pressures, these concept prototypes feed the illusion that development cycles can be drastically shortened because, "the GUI is already done!". These patterns addresses how prototypes can be used as a tool to inform the developers, managers and clients, and even venture capitalists, rather than an arbitrary baseline to which all further development must conform.

Pattern Diagram:

This diagram describes how the patterns in this language fit together. *Use It And Lose It* is the main theme for this language, with the anti-pattern *Prototype to Delivery* containing it's mirror image. The two gray boxes containing *Engage the Client Early* and *Pride Cometh Before A Fall* are messages to the developer about working with clients and containing their hubris in forging those relationships. The green colored branch containing *Feature-Free Zone*, *Come On Baby Light My Fire*, and *Representative Action* describe to the developer a way to handle the visual characteristics of their prototype. The blue branch containing *Self-Service*, *One Way Street* and *High Observability* show the developer how to effectively relate the characteristics of the prototype to the client. Finally, the purple box containing the pattern *Let's Make A Deal* is an exit strategy from the world of the prototype to a true development project.


Use It And Lose It
Engage The Client Early
Pride Cometh Before A Fall
Let's Make A Deal
Feature-Free Zone
Come On Baby, Light My Fire
Representative Action
Self-Service
One Way Street
High Observability
Prototype to Delivery

Use It and Lose It

Richard Teague, legendary creator of the AMC Pacer demonstrated that a car could be round and actually become airborne at 100MPH. An example of something that should never have made it out of the ough sketch stage? [2]

Problem:

Prototypes live beyond their intended purposes. How can you ensure that a concept prototype is perceived and utilized in the manner for which it was intended?

Context:

Developers and their managers are using GUI prototypes as a method for getting client buy-in and funding for a project.

Forces:

- Developers can quickly create extensive GUI's using Rapid Application Development tools
- Clients and Managers are looking for ways to compress the development cycle
- GUI prototypes are an effective means of communicating a potential solution and gathering input
- Feature rich prototypes look production ready
- Feature rich prototypes generate excitement in the client

Solution:

Design the prototype in a manner that underscores the transient nature of it. Discard the prototype after you establish client buy-in.

Use the supporting patterns *Feature Free Zone*, *Come On Baby Light My Fire*, *Engage The Client Early*, and *Representative Action* as ways to generate excitement for your idea, while keeping the prototype simple and lightly functional. Make your idea work for you, using *One Way Street*, *High Observability*, and *Self-Service*. Move beyond the prototype phase by employing *Let's Make a Deal.*

Resulting Context:

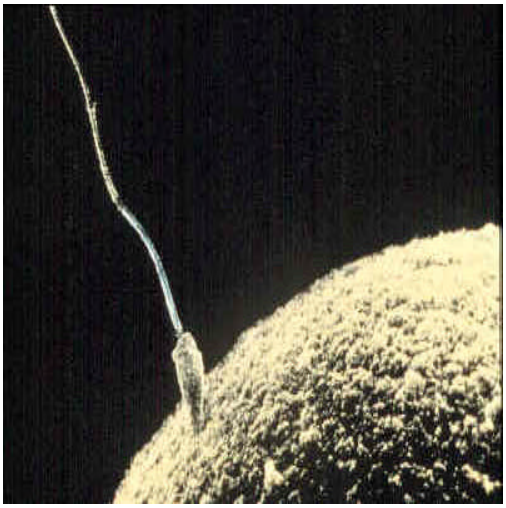
The prototype is not confused for a nearly finished product, nor does it live on like an albatross around the necks of the developers who feel the need to conform to any ill-conceived constraints. This may also be confusing to clients who don't necessarily understand that buggy products result from a prototype-driven development process and he or she may be downright unimpressed with your efforts. Use *Light My Fire* as a way to combat this lack of enthusiasm.

Design Rationale:

The idea for this pattern stems from the oft heard complaint about a software development cycle expressed in the Anti-Pattern *Prototype to Delivery*; the result of a client (who holds the purse strings) seeing a prototype, getting excited about it, and wanting it right away. It's my belief that the polished nature of these prototypes create the illusion that the system is 'practically done', when often the requirements for the system haven't even been generated yet. The developer is then in a jam, and will use the prototype as a proxy for actually doing the necessary work to create useful requirements for both the user interface and the underlying system itself.

The result, especially for complicated or real-time systems, is a poorly designed system that reflects badly on the development team and an unhappy client who is left with an unreliable, difficult to use piece of software. Or perhaps just as likely, a dead idea.

Engage the Client Early



In the world of your idea, the prototype is potential life – an unfertilized concept that is waiting for an external force, the client, to give it meaning.[3]

Problem:

You are using a very simple GUI-based prototype to explain an idea. How can you get the client involved in developing the idea further?

Context:

Developers are building concept prototypes to describe a new concept or to solve a perceived problem.

Forces:

- Extensive prototypes distract the user from seeing other possible solutions
- GUI models help the client visualize the solution that you have in mind for solving the problem
- Functional GUI prototypes are simple and quick to design with RAD tools
- Clients that are engaged in the process early on lead to overall project success

Solution:

Use the prototype as a baseline for discussions with the client about how he can help you solve his problem, rather than as a platform for demonstrating your personal programming prowess. Consider creating a family of prototypes that express different ways to solve the same problem and demonstrate your openness to various solutions, using *Representative Action* to show you are working to understand their needs. Specifically ask the Client if you are understanding, and be prepared to actively listen to his responses. Take notes.

Resulting Context:

If the client can be hooked into your solution, you can use the client to help drive your development efforts. They will be more patient in helpful throughout the whole process, and they become an integral part of the solution rather then just an "end user". Use *Let's Make A Deal* as the next step in cementing a relationship with your client.

Design Rationale:

There are reasons other than the psychological reasons for having a client engaged in your development process. They are absolutely essential to developing a tool that fits the computing environment and user's skill set and ultimately the acceptance of your solution. Successfully using *Engage The Client Early*, means your team will have an early start at building a relationship with the client that allows for more frequent check-pointing during the design and development process. Of course, client involvement is risky if the client attempts a hostile takeover of the whole process and is generally difficult to work with.

Pride Cometh Before a Fall

Jimmy Swaggert learned that there are times to be humble.[4]

Problem:

High Tech people are motivated by new technology. How can you, as a developer, create simple, feature-light prototypes and still remain excited about your work?

Context:

You are a developer (or a savvy manager of developers) who understands the need for a *Feature-Free Zone* and *Self-Service* to help *Engage the Client Early*.

Forces:

- Developers need to stay on the cutting-edge to remain competitive
- Clients and Managers don't always understand or care about the latest technology
- Developers have a desire to show how bright and clever they are
- Using the newest technology often means dealing with unexpected events

Solution:

Avoid the desire to use the latest tool or technology just for the sport of using it, when creating concept prototypes. You may be undermining your own efforts to secure a new project or understand the product domain fully to the client's benefit. Focus on choosing the most appropriate tool to accomplish your task.

Resulting Context:

You may spend some up-front work time doing research on the problem domain and some very unglamorous prototype building, but your up-front efforts will bring you sanity and time for creative thought down the line. Developers love to try the latest technology and often don't mind if their clients become beta-testers for a new technology, but may lose the

credibility gained in appearing very clever when the product fails to perform as advertised.

Design Rationale:

It is important to do creative work when you are a developer – for many of us, that is our only source of personal satisfaction in our field. The prototype phase, though, is really a time to be collecting information on the problem domain for which you are providing a solution. I have seen developers, like myself, who just want to get back to the keyboard, bang away and throw the results over the wall and forget about it. In a world where so many of our development efforts end up in the bit-bucket, through cancelled projects or design changes, perhaps we can save a few personal cycles and have a more enriching professional experience if we take the time up-front to listen and understand our client's needs. Consider the projects you may have worked on that have been cancelled, because ultimately, they didn't meet the requirements (that were never fully defined in the first place). Managing expectations starts with setting the bar at a realistic height, unless you like the pain of falling mid-span on a hurdle.

Feature-Free Zone

The Classic Beetle: A limited set of functions, but it met the needs and desires of a generation of drivers. And you could fix it. [5]

Problem:

Engaging a client's interest with a flashy GUI prototype is a valuable way to plug a new concept. How can you keep the prototype from distracting the client into thinking you are displaying a nearly completed product?

Context:

You are using tools to build GUI-based concept prototypes to demonstrate your new idea to a new or potential client.

Forces:

- Clients want fast turn-around from project conception to delivery

- Feature-rich models create expectations
- GUI models look finished and well thought out
- GUI models generate excitement for the product
- Clients hire you because you generate excitement

Solution:

Introduce a prototype that demonstrates a limited set of functions that demonstrate the core functionality. Provide supporting documentation and consider packaging your prototype using a presentation tool, such as Macromedia Director or even HTML, as opposed to a compiled executable.

Resulting Context:

The client's expectations are managed by the fact that he doesn't see everything he desires in your prototype, but sees the potential for a system that may meet his needs. As *Use It and Lose It*, you may seem unimpressive or seem to demonstrate a lack of understanding for the client's problem domain. *Use Come On Baby, Light My Fire* and *Representative Action* to address these concerns.

Design Rationale:

Products that are developed from the GUI down (using the prototype as a baseline for the system design) result in spaghetti code that is difficult to maintain and may be unreliable. These concerns are exacerbated in real-time systems that require high availability and resiliency. Clients may assume, after viewing a functional prototype, that someone can flip a switch somewhere and the system will become available – when in actuality the system does not even exist, and there is no evidence that it could actually be delivered!

One of the most effective concept prototypes I've seen was developed at MediaOne Labs and is a prototype of video email using a high-speed data connection and your television. The prototype was done completely in Director, using pre-generated video. The prototype is small, highly visual, simple and can be carried on a floppy disk.

Come On Baby, Light My Fire

In the world of rock, sex appeal creates attraction. But true excitement for music comes from sharing something in the music that touches you personally. [6]

Problem:

You are using a very simple GUI-based prototype to explain an idea. How can you excite the client about your idea?

Context:

Developers are building prototypes to describe a new concept or to solve a problem. You understand how to employ *Representative Action* in creating your prototype.

Forces:

- Clients are excited by flashy, rich graphical environments
- Extensive prototypes distract the user from understanding the problem you are trying to solve
- GUI models help the client visualize the solution that you have in mind for solving the problem
- Functional GUI prototypes are simple and quick to design with RAD tools

Solution:

Develop the prototype GUI, knowing that you are working to convey the essential core principles of your solution. Use a structure that lends itself to *Representative Action* and that avoids the need for the user to switch context, such as occurs with multiple window pile-ups, pull-down menus or any need for extra work to get going, as described in *Self Service*. A prototype doesn't need a splash screen, or even a logical starting point; cut to the quick and provide just a few screens that show the user how a core session with your product might progress. If you are developing your prototype for a known creature, spend

some time finding out what turns that person on -- do they think browsers are the windows to the soul? Deliver you concept via a web interface. Whatever platform you choose, ensure that it makes sense in terms of how the ultimate product will be delivered -- Don't use a Macintosh if your company has banned them.

An example that demonstrates a core principle; the user clicks on a button that says get my data, and the system gets the data and displays it in a list. The user clicks another button that sorts the list. If the client has never been able to access data like this without going to 5 different data sources and poring over printouts, she will be excited, and the fire will have been lit.

Resulting Context:

The client can clearly see how the direction you would take to solve her problem, without distracting her with particular interface issues that aren't appropriate at this stage. By knowing the kind of environment your client is most comfortable with, she will immediately perceive you as an ally or even brethren in her personal struggles. It helps to remember that *Pride Cometh Before A Fall*, when you are tempted to flex your coding muscles during prototype development.

Design Rationale:

Concept prototypes are a great way to get people's juices flowing about the possibilities of solving a problem The prototype does more to help express the nature of the problem, than provide a solution in and of itself. Developers are tempted to churn out complicated interfaces to describe their idea, when they really only serve as distractions and impediments towards the real task at hand, which is to provide a workable solution to a real problem that the client has.

Representative Action

In the US, we elect our representatives to address complex problems. But, like a concept prototype, it helps if they look good. [7]

Problem:

Your don't want your prototype to over-simplify the complexity of the problem you are trying to solve. How can you communicate this?

Context:

You understand the need for a *Feature Free Zone* within your prototype. You are trying to *Engage the Client Early*.

Forces:

- Effective prototypes make the solution look easy
- Clients want fast turn-around for their products
- Everyone thinks their problems are tough

Solution:

Demonstrate a solution for a single problem from the client's domain, which while limited in scope, demonstrates that you understand the true nature of the problem. Finding this problem involves taking the time to understand the issues that the Client is actually facing and most importantly, what she is doing NOW to solve the problem. This often involves investigative work and sleuthing. Analyze your discoveries, and find something the client will immediately recognize to use as the heart of your prototype.

Resulting Context:

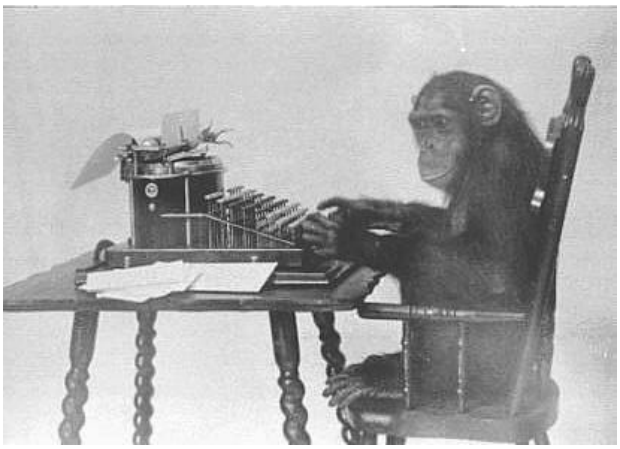
The client trusts that you understand their biggest worries and can help them solve her toughest problems. You have demonstrated how your concept will help her solve her problems, by showing how you will solve one of them. She can understand that, because

you are using their own domain to establish common ground. No one thinks his or her own problems are easy, and subsequently with your use of *Let's Make A Deal*, she won't over-estimate your prototype. With the use of *Come On Baby, Light My Fire*, she won't underestimate it either.

Design Rationale:

I worked on a project where the client had many problems. One of them was a process that required paper to flow back and forth and required each party to change numbers in a spreadsheet. Our client would apply certain formulas to the original spreadsheet and then pass the spreadsheet back. This resulted in a number of iterations. We simply took her formulas and put them into a GUI web-based calculator that allowed her customers to solve the problem themselves and then push a button that would submit them to her for approval. By demonstrating how we could solve one small problem in her problem space, we suggested a change that improved a 2-week bottleneck and encouraged the client to trust our ability to solve tough problems.

Self Service



In 1714, Henry Mill was granted a patent for "…an artificial machine or method for the impressing or transcribing of letters, singly or progressively one after the other, as in writing...." He hit on an incredibly simply way of solving a complicated problem, that still lives on today in the context of word-processing. And he never needed to look for an electrical outlet. [8]

Problem:

You are using a concept prototype to help sell your solution for solving a complicated problem. How can you ensure that your prototype will run well anytime, anywhere?

Context:

You realize from your use of *High Observability*, that your prototype needs to be available, even when your operating environment is not.

Forces:

- The availability of supporting infrastructure, like databases and network connections are not 100% reliable.
- You want your customer to concentrate on your ideas, strengths, and not idiosyncratic details.
- You have created a prototype GUI that doesn't rely on a strong, underlying infrastructure.

Solution:

Package your prototype in a manner that allows it to be self-sustaining and completely free of reliance on any other entity outside of its immediate operating environment. Avoid conditions that require your user to set-up their machine in any specific manner to operate your prototype, such as resetting screen resolutions or changing configuration files. You will need to use your imagination to fill in some of the 'blanks' – for example, if your prototype assumes that the user will have some sort of a visual streaming feed, mock it up with an inline animation or loop that shows how that would look.

Resulting Context:

When your prototype is run, it will behave in a completely expected way that allows you to concentrate on the ideas you are trying to convey rather than having unrelated distractions that are mere side-effects of using RAD tools to generate GUI. At the same time, this does not allow the customer to consider issues like latency or other real-time factors that would occur in a finished product. It is also sometimes impossible or prohibitive to simulate elements that exist outside of the prototype itself, such as a broadcast signal.

Design Rationale:

I worked on a team that developed a new way to deal with the radically disparate data feeds, sources and types inherent in weather data. The heart of the new product was a GUI that tied all these different forms together in one coherent interface. Since this was federally funded work, the client was the US Congress. You can imagine the embarrassment that occurred when our demonstration failed to connect to our database. The screen was blank; there was no data to depict. We learned from that fiasco, that canned datasets are not to be denied. Not only were we relieved of the pressure of ensuring a fast, reliable network connection, but we could hand pick data that looked exciting and our demonstrator always new what to expect. He was able to concentrate on describing the power of our solution, rather than having to worry about things going awry.

One Way Street

In software, as in driving, being able to follow the signs is the key to survival.[9]

Problem:

Letting a client drive a concept prototype helps engage his interest. How can you ensure that he isn't distracted or confused by the interface?

Context:

You are using a GUI to generate excitement, using the principles learned in *Come On Baby Light My Fire*, for your new idea and you want the client to give it a test drive. You know that hands on experience is a great way to get people to really understand a product. We know that we want to *Engage the Client Early* to aid our ultimate success.

Forces:

- A GUI can be confusing the first time it's used
- Using pull-down menus and pop-ups makes a user hunt for information
- An intuitive GUI makes the user feel good about using the system
- You sell software because it makes the user feel smart, not because you look smart

Solution:

Design your prototype so that it leads the user from action to action, by only providing one obvious direction. This can be accomplished by building a prototype that operates like a circular buffer; once you go all the way around, you automatically end up at the beginning. Think about designing the prototype like one might design a presentation, using simple, recognizable icons for going forward, backwards, and executing an action. Be mindful of always providing a way for the user to gracefully recover if they get in too deep; much like the "Home" button that a browser implements. One example of this, when successful, is the Wizard found in some Windows applications, that leads the user through potentially complex tasks in an instructional way.

Resulting Context:

Your client is able to immediately use your prototype to create an action. She feels smart and you can leverage the good feelings your client has for predicting the success of your product.

Design Rationale:

There are many clever things that we can do as developers in solving complex problems. It's always nice to be thought of as clever, and in this case, we want to make our prototype user feel clever and satisfied.

High Observability

The B-2 Bomber is the most survivable aircraft because of it's low observability. When the goal is getting noticed, take care to avoid stealth elements.[10]

Problem:

You are using a very simple GUI-based prototype to explain an idea. How can you ensure that your GUI doesn't need you around to explain it?

Context:

Developers are using rapidly developed GUIs as a way to describe a new system to management or non-technical clients.

Forces:

- Non-technical clients and managers are excited by flashy, rich graphical environments
- Extensive prototypes distract the user from understanding the problem you are trying to solve
- GUI models help the client visualize the solution that you have in mind for solving the problem
- Functional GUI prototypes are simple and quick to design with RAD tools

- Complex, unusable GUI prototypes are simple and quick to generate with RAD tools
- You want your GUI to be presentable without your presence
- Developers are protective over their work and like to show it with pride
- Developers are not around when business decisions get made
- Developers are motivated by technology

Solution:

Develop a few cohesive screens that can be dumped out, mailed around, or stuffed in a business plan, most importantly, each screen in your prototype should be able to stand on it's own merits without any further context being required. There are no stealth GUI elements in the prototype, such as pull-down menus or pop-ups (you can't see these in a screen dump) -- All elements are highly observable and immediately obvious.

It is important that the prototype is accompanied with enough of a framework (e.g. written problem statement, supporting documentation) that, especially if it is very complicated, the depth of the problem itself is not lost.

Resulting Context:

Other non-technical personnel are comfortable describing your idea to someone else, who may have some influence on whether your idea moves forward or not. Developing this kind of a prototype is personally risky to the developer whose pride is bound up in their work, resulting in the need to display his cleverness at problem solving. Use the pattern, *Pride Cometh Before A Fall* to help manage this feeling.

Design Rationale:

Developers can often excite technical people with their great ideas, but then they die there. A project may not get funded because the business people don't understand the work of the developer. If a non-technical person can describe the system, then it has a better chance of surviving. You have effectively grown your sales force, without spending a dime.

Let's Make a Deal

In architecture, clients may purchase the services of an architect based on preliminary plans of the structure they want built. A competent builder, on the other hand, will wait for plans with actual measurements before breaking ground.[11]

Problem:

You have successfully used a concept prototype to establish client buy-in for your solution. How can you ensure that you and your client both understand the expected functions of the new system?

Context:

You know from *Use It and Lose It* that you want to discard your prototype and avoid any substantial use of it.

Forces:

- Prototypes are sometimes used as a proxy for requirements gathering
- There is a desire for those funding the project to compress the project cycle

Solution:

In developing a successful concept prototype, you have avoided the traditional tasks of requirements generation, but now the time has come. Using what you learned *in Engage The Client Early* and *Representative Action*, you should have some sort of a rapport with your client; if not amity, at least a basic understanding of their needs. After the client has agreed that she likes your solution and wants to go further, sanctify the relationship. By agreeing to produce the appropriate Requirements documentation in a traditional reciprocal, iterative manner with the Client, you will have the basis for the building of a deployable system.

Resulting Context:

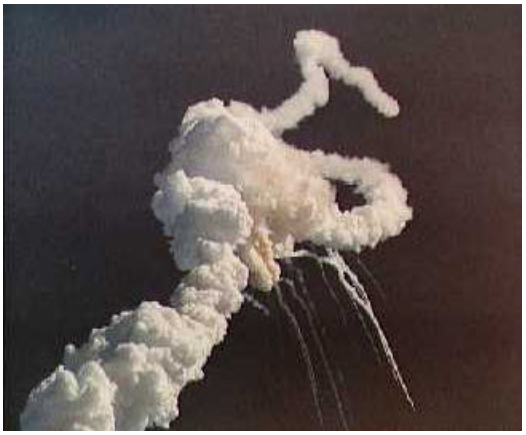
By generating a requirements document, you can be sure that you and your clients have an understanding about the features of your new system and you will have successfully avoided the traps often laid by the use of prototypes. You will also have a usable artifact that will contribute to the creation of a deployable system that meets the needs of the client.

Design Rationale:

How does one move successfully beyond the concept prototype stage, without getting caught in the trap you were working to avoid – your prototype being perceived as something more than it was every intended to be? It seems that while creating the opportunity to produce the more formal kind of documentation that every successful project requires, this juncture of the process allows you to move to a new level that casts both the client and the development team as actual stakeholders in the development process. Without a Requirements document process, the ontological framework for the project may become muddy and confusing, resulting in a failed product.

Author: Monica Marics as suggested to Carol Stimmel

Prototype To Delivery: (An Anti-Pattern)



On January 28, 1986 the space shuttle Challenger exploded. The cause was in part determined to be due to a hurried cold weather launch and a subsequent seal failure in the solid rocket booster. O-Rings do not seal properly in cold weather, even when commanded to by politicians.[12]

Problem:

After showing a prototype to a potential client, the client is excited about your idea and wants it as soon as possible. How can you use the work you've already done to give you a jump-start?

Context:

Developers are using tools and techniques that allow them to produce flashy looking prototypes with the appearance of extensive functionality and rich graphical elements in exceedingly short time spans.

Forces:

- Clients want quick turn-around on their projects
- Developers like to be fast and effective
- Speed of product delivery can supercede quality
- A compressed development cycle can squeeze out design and testing phases
- There are two kinds of developers: The Quick and The Dead

Solution:

Build prototypes that are as close to your vision of the finished product as possible. Use that as the baseline for your development effort. The development timeline will be compressed because you will have already created the user interface and you won't need to be concerned with a UI design phase, since you already have the client's buy-in.

Resulting Context:

The development team will look heroic for producing a product of such complex nature in a shorter-than-average period of time (hopefully). The client will look good for getting well within budget and for getting the most for the company's money. Although, the users will have inordinate troubles using your system that has such a minimal amount of thought put into the interface and how it interacts with your system (and it will probably crash and burn with great regularity), these issues can be addressed in further iterations of the software.

Design Rationale:

It's important to cater to the whims of your client, since they provide your bread and butter. Your development team should be proud of how quickly they can produce complicated software save cycles in the development process. Your client will learn to trust you as a real go-getter with the ability to get the job done on time. Anyway, trying to fix your code will be a great task for the new college graduate.

REFERENCES:

James O. Coplien, *A Development Process Generative Pattern Language* available at http://www.bell-labs.com/people/cope/Patterns/Process/index.html

ACKNOWLEDGEMENTS:

Don Olson, of AG Communications for his encouragement at ChiliPLoP '99 and shepherding of this paper. His ability to help find the nugget of truth in my disparate collection of thoughts is appreciated.

Mary Lynn Manns, of University of North Carolina, Asheville, for introducing me to patterns and her comments on the early iterations of this work.

Robin A. Seidner, The Copy Diva http://www.copydiva.com for her brilliant editing eye

Lyn Bain, of U S West Advanced Technologies, for introducing me to the fact that users even exist.

ENDNOTES:

---

[1] "Hold Me, Thrill Me, Kiss Me, Kill Me", is a single by U2, 1995 for the soundtrack to *Batman Forever*.

[2] Picture courtesy of Jeni, proud Pacer owner with a dedicated site at http://www.lightstream.net/~jeni/pacer/jeni.html

[3] This photo courtesy of Public Domain Images http://www.pdimages.com.

[4] Used without the permission of Jimmy Swaggart Ministries. http://www.jsm.com

[5] Picture courtesy of Dan Dunn, President of the Beetle Farm, http://www.thebeetlefarm.com

[6] This picture of Jim Morrison is ubiquitous on the Internet (and probably a lot of walls) and I have no idea who to cite for its usage, though I suspect Annie Liebowitz and Rolling Stone own this one. If you know, let me know.

[7] I found this photo at http://www.students.bucknell.edu/bbaker/jfk.html. It is unquoted and there was no contact information available.

[8] Photo courtesy of Public Domain Images http://www.pdimages.com

[9] This photo is from the movie, *Thelma and Louise*, MGM Home Entertainment 1991, Ridley Scott, Director.

[10] Photo Courtesy of the U.S. Air Force/Northrop Grumman.

[11] Photo from MediaOne Group collection of photos for Microsoft Word.

[12] Photo courtesy of the United States Government. Permission unobtained.