INTRODUCTION

"Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the Second Law of Thermodynamics: i.e., it always increases."

(Augustine 1983)

Software Engineering methodologies are reported to be responsible for saving millions of dollars and have made programming projects for things such as Space Station Freedom feasible. Additionally, many corporations have or are getting ISO 9000 certification, which mandates software engineering procedures be followed in developing software. Contractors are also increasingly seeing corporations and government agencies require that they have software engineering methods in place before they are awarded business. Plus, in this day of fancy GUI interfaces, users require more of our programs and as programmers we also demand more out of our efforts. Lastly, as SAS has moved from a niche statistical tool to a major application builder for client-server environments, developers need a way to handle the complexities of major software development projects.

A plethora of models exist in the software engineering literature to fit a myriad of development environments. However, software engineering principles have mainly been designed for large projects using lower-level programming languages. A good example of a software life cycle, is the model which SAS Institute (1993) adheres to in their development process. Additionally, Phillips and Hall (1995) have shown where they applied software engineering principles in a SAS-based development environment.

LIFE-CYCLE MODELS

"The main reason the process matters is that in software, quality must be built in from the first step onward. This flies in the face of the folk wisdom that you can code like hell and then test all the mistakes out of the software. That idea is dead wrong. Testing merely tells you the specific ways in which your software is defective. Testing won't make your program more usable, faster, smaller, more readable, or more extensible."

(McConnell 1993)

The "waterfall" model is probably the oldest software life-cycle model and most widely known models. There are plenty of reasons why this is not a good model to use for software development; document driven, little room for interactions, specifications are frozen too early, no room for management activities (Charette 1986, Sommerville 1992). Though this model has faults, it is a very good conceptual model of how software should be developed. Plus, if we were in a perfect environment the waterfall model would probably be the fastest and most successful way to develop software. One of the more remarkable aspects of the waterfall model is that it recognized that there are distinct stages that occur in the development of software, e.g., Requirements, Design, Implementation, Test, Maintenance. Other life cycles may divide and slice these stages, but it is hard to really develop new categories for these stages. Also, most life cycles assume that these stages of development are tied together in basically a linear fashion.

Unfortunately, in my experience's software development efforts do not usually occur in a linear manner. Also, since I work in numerous types of environments, a model that would allow diverse types of development patterns was required. As a consultant, my expertise and career lie largely with efforts that have consisted of working on small development teams. Therefore, my experiences have been that the amount of effort available to applying software engineering is limited to non-existent. In addition, I typically work with and for groups who have little or no background in software engineering. Additionally, I am also usually responsible for data quality and statistical suitability. Therefore, SAS is a natural choice for developing a majority of the projects I face. However, unfortunately though, since I am using a 4GL-type tool I do not have the luxury of utilities and tools that normal programming and software engineering environments utilize (i.e., GREP, CASE, Library Control Tools, Automatic Testers). A model that would assist me in conducting business and not hinder my efforts was therefore required.

SW-PONG

"If software development were a deterministic, algorithmic process, you could follow a rigid methodology to your solution. Software development isn't a deterministic process, however. It's heuristic which means that rigid processes are inappropriate and have little hope of success."

(McConnel 1993)
One of the first computer games that came out was a computerized version of table tennis, called PONG. The object of the game was to bounce a ball off a wall using a paddle that was controlled with keys on the keyboard. More advanced versions of this game broke the wall into bricks, which would disappear as the ball hit them. The object of this game was to break the wall apart as fast as possible by hitting the ball so to remove as many bricks as possible and break through to the other side. If one uses a little imagination, software development can be considered to be like the computer game PONG. However, to more aptly describe, the software development process, the game will now be called SW-PONG for Software developed through Progressively Ordered Noetic-based Goal-making.

The basis for SW-PONG is that software very seldom is developed in a linear fashion. Furthermore, the concept behind SW-PONG is that development bounces between the various stages of software development, much like the ball bounces in the computer game Pong. Linear models are possible using SW-PONG. However, because of the flexible nature of SW-PONG numerous development configurations can be followed within the same framework.

SW-PONG, as figure 1 illustrates, is made up of a board that consists of two walls. The upper wall is called the Development Bank. This Development Bank is further broken into sections based on the recognized stages of development; requirements, design, implementation, testing, and maintenance. The bottom wall, which controls the paddle (and thus the direction of the ball), is called the Quality Bank. The Quality Bank is a fusion of all the quality and management activities that are involved in developing software, e.g., software quality, cost estimating, scheduling, planning, risk analysis. The ball will carry information back and forth between the two walls. Additionally, if the Quality Bank is capable of operating in expert mode, more than one ball may be in play at any given time.

A software project being developed with SW-PONG, begins with the formation of the Quality Bank. This bank is responsible for starting the project. The first task of the Quality Bank is to build the Development Bank. It is up to the bank to choose exactly what stages of development the project will likely experience. Therefore, the Quality Board must tile each stage’s wall with bricks that represent development units within each stage. For example, requirements may be broken into such divisions as user interface requirements, data analysis requirements, and network security requirements. If a brick extends all across a stages wall, then a project ball is forced to complete that step before it hits that particular stage. A brick may also just occupy part of a wall. Therefore a project may break through a stage without completing the step that exists in the bank. The next step that the Quality Bank must conduct is to prepare and arrange all of its functions so it is ready to hit the ball back as fast as possible. Lastly, the Quality Bank loads the ball with information and launches it towards its first target on the development bank.

The ball now carries information to the Development Board. The task which the ball hits now takes control of the game. The first task that must be accomplished is to absorb the information in the ball. Next the Development Board tries to complete it's assigned task as soon as possible. Upon completing this task, the Development Board loads the ball with the results of the task and sends it back to the Quality Bank.

The Quality Bank now receives the ball from the Development Bank. The first function of the Quality Board is to analyze what the task did and store the results. Next management activities such as formal reviews and configuration management occur. The Quality Board will then determine where the ball should go next. Activities at this point will consist of such management activities as cost estimating, risk analysis, scheduling, and budgeting. Finally the ball takes with information and launched back to the appropriate stage in the Development Bank.

The bouncing of a ball continues throughout the life of the software. This ball can follow many patterns. A waterfall method would have the ball bounce from requirements to design to implementation to test to maintenance. However, because the ball can bounce anywhere on the Development Board numerous other patterns are easily constructed with SW-PONG. Eventually patterns of software development will emerge to fit particular projects and development groups. For example, many software projects today are essentially maintenance projects. If after doing requirements analysis it is decided that the problem is simple, the ball may bounce straight to implementation. On the contrary, if the problem is found to be significant then the ball may bounce to design. In either case the Quality Board has decided based on reasoning (noetic-based) on where the ball should go next, not the life cycle model. Therefore, software development is truly modeled through progressively ordered noetic-based goal-making.

**SW-PONG EXAMPLE**

"On a small project, the talents of the individual programmer are the biggest influence on the quality of the software. Part of what makes an individual programmer successful is his or her choice of processes" (McConnell 1993)

---

1 According to the Random House College dictionary, Noetic is defined as "1. of or pertaining to the mind. 2. originating in or apprehended by the reason"
The development environment for many data analysis teams must be extremely flexible. Additionally, regardless of the amount of advance planning, the majority of work never seems to be fully realized until a test actually begins. Furthermore, typically, software development is done in concert with the development of statistical test procedures. Figures 3 through 7 are an example of how the developmental stages might be setup in a SW-PONG board that meets the needs of a typical data analysis project. These stages are just a beginning; and new bricks may be added as needed. Also, it is extremely easy for the Quality Board to skimp on their duties. Therefore, it is important to have safeguards in place to insure that all management activities are implemented.

SUMMARY:

"Perhaps the best advice is to "fake it". What is important is not that a process model isn't perfect, but that one is used. It is better to use an incomplete model than none at all. This way, project participants can at least communicate, and management can have some measurement of project schedule and success. And using a model does work. Also, using a model and finding its weakness moves one closer to the ideal." (Charette 1986)

SW-PONG is a realistic model of software development. By allowing maximum flexibility within a controlled framework new technology and ways of doing business are possible. Unlike the waterfall model though, this structure does not guarantee that software will be developed successfully. Furthermore the use of SW-PONG is dependent on the Quality Board's ability to make good decisions at each bounce of the ball. Also, it assumes that the Quality Board will not have difficulty in deciding how to bounce the ball. For these reasons, this model would probably not be very feasible on large projects. However, it is a very good model for those working on smaller team-oriented projects.

SW-PONG is a life-cycle model that I have found works well in my environment. Finding a life-cycle model appropriate to your needs may seem like a daunting task. However, the literature is loaded with alternative models for all sorts of projects. While conducting your search, I would strongly suggest that a development team go ahead and implement both a software metrics program and adopt a formal design methodology. Having both of these programs in place will make the search easier and make the transition into a more structured environment smoother.

LITERATURE CITED


TRADEMARK INFORMATION

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

AUTHOR

The author welcomes your comments and suggestions at:
Roger Bolton
Avanti Corporation
102 Silver Creek Circle
Madison, Alabama 35758
(205) 552-2621
Figure 2. The SW-PONG Board

Figure 3. Requirements Block
### DESIGN

<table>
<thead>
<tr>
<th>Data Flow Diagram</th>
<th>User Interface Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Test Design</td>
<td>Data Reduction Plan</td>
</tr>
<tr>
<td></td>
<td>Data Report Design</td>
</tr>
<tr>
<td></td>
<td>Initial Software Design</td>
</tr>
<tr>
<td>Preliminary Specifications</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Design Block

### IMPLEMENTATION

<table>
<thead>
<tr>
<th>Deliverable Software</th>
<th>Beta-Version Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface</td>
<td>Data Reduction Plan</td>
</tr>
<tr>
<td>Statistical Design</td>
<td>Report Prototypes</td>
</tr>
<tr>
<td></td>
<td>Graphical Methods Prototype</td>
</tr>
</tbody>
</table>

Figure 5. Implementation Block

### TESTING

<table>
<thead>
<tr>
<th>Software Acceptance Test</th>
<th>Beta Software Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Final Data Quality Assurance Test</td>
</tr>
<tr>
<td>Data Analysis - Approval &amp; Quality Test</td>
<td>Data Reduction Proof</td>
</tr>
<tr>
<td></td>
<td>Statistical Methods Trials</td>
</tr>
<tr>
<td></td>
<td>Methods Acceptance Test</td>
</tr>
</tbody>
</table>

Figure 6. Testing Block
Figure 7. Maintenance Block