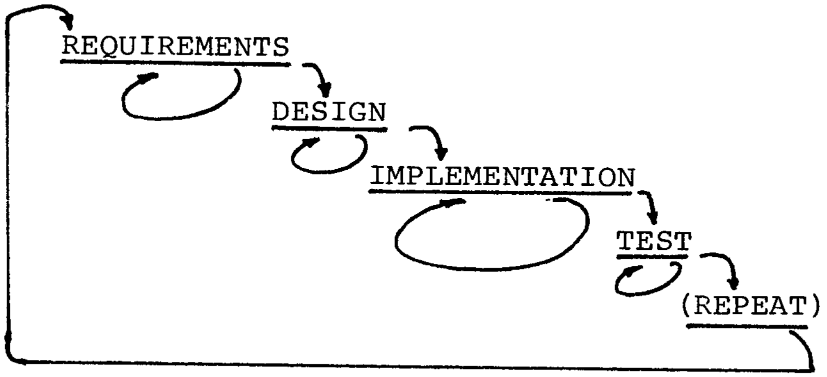
STOP THE LIFE-CYCLE, I WANT TO GET OFF

G. R. Gladden

The Life Cycle

I am of the opinion that the concept of a 'software life-cycle' is no longer helpful, indeed may be harmful to our software development profession. In its various forms the life-cycle has sought to describe the software development process as iterative events within the major tasks of design, implementation, test, etc. One begins to visualize the development process as a sequence of tasks 'waterfalling' into one another while within each task modifications occur iteratively as a better understanding of the system acquired is (Fig. 1). These iterations work together to extend project schedules, invalidate designs, alter test requirements, and to generally infuriate customers.



(Fig. 1) Typical Life-Cycle Representation

Most insiders continue to decry the failings of software. Its lateness, incompleteness, and error-proneness are topics for seminars and workshops as well as popular literature.

New areas of expertise are emerging to address the problems. Quality control and Configuration Management are two such areas. A great deal of attention has been focused on 'structured programming'.

Presently 'structured' has become the rubric under which all life-cycle tasks are grouped: structured testing and the like. The results of all of these efforts to correct the software 'crisis' (although inarguably producing better code) have not been satisfactory. On the contrary, if we are to believe some reports [1], they are abysmal failures.

What I would like to purpose in this message is not the abandonment of a professional and methodical approach to software, but rather to offer my perspective of the root cause of the problem and an alternate approach to undertaking software projects.

Frist, I contend the following:

(1) The chief villian in any software fiasco in a non-existant, vague, incomplete, or a poorly thoughtout set of requirements.

(2) The 'life-cycle' approach exacerbates the problem by encouraging eleventh hour alterations to whatever requirements do exist. Each modification to the requirements adversely effects the system by impacting each subsequent task. Conversely, each modification to tasks downstream adversely effects the preceeding tasks including the requirements. The result is a vicious (life) cycle.

(3) The elapsed time between requirements and a delivered product erodes a customer's confidence. Such eroding confidence manifests itself in new, altered or expanded requirements, or other modified task elements.

The above contentions generally chain together in the following sequence of events: system requirements are incomplete, however the project must proceed and so it does. During the course of development as new requirements emerge, the schedule is lengthened and customer confidence falters. New requirements are ladled on as the user seeks to assuage his growing fears that the developer does not have a firm understanding of his needs. Finally, when the product is completed (late) the requirements have changed to the point that the product no longer satisfies or even resembles them.

A New Approach to Development

I purpose a new view of the development process, especially the way requirements, or lack of them, affect it. I first state 3 propositions.

Proposition 1

System objectives are more important than system requirements.

Objectives can be set in a relatively short period of time and once they are set they are less likely to be subject to change. Objectives are set at the highest management levels from all the concerned system users. If objectives are changed the proposed project is clearly not the same and the need for a new system must necessarily be reexamined. Concentrating on objectives can go a long way to prevent a system from 'evolving' into one that the user does not want or need.

I believe that a design objective approach [2] is of paramount importance here. Our own experience bears this out.

Proposition 2

A physical object conveys more information than a written specification , (or a picture is worth a thousand words).

Nothing conveys more meaning or serves to congeal a system concept better than the system itself. We propose the liberal use of mock-ups of physical hardware early in the project. Similarly, 'mock-up' software should be encouraged. In a system that must interact with a variety of people, nothing can be more positively influencing to the success of the project than to see the proposed system in operation. Entire operating environments or scenarios may be staged in a room with mocked-up hardware and software interacting with live people. I believe that commercial artists, script-writers, film, and model makers will play a vital role in system developments of the future. Actors and hardware portraying system features under the direction of the project manager and his customer is not too improbable to imagine. (I am not advocating a great deal of disposable software which wastes programming efforts, but rather an amount necessary to demonstrate the attributes or goals of a system. See the letter on PNAMBICS [3] for instance.)

Proposition 3

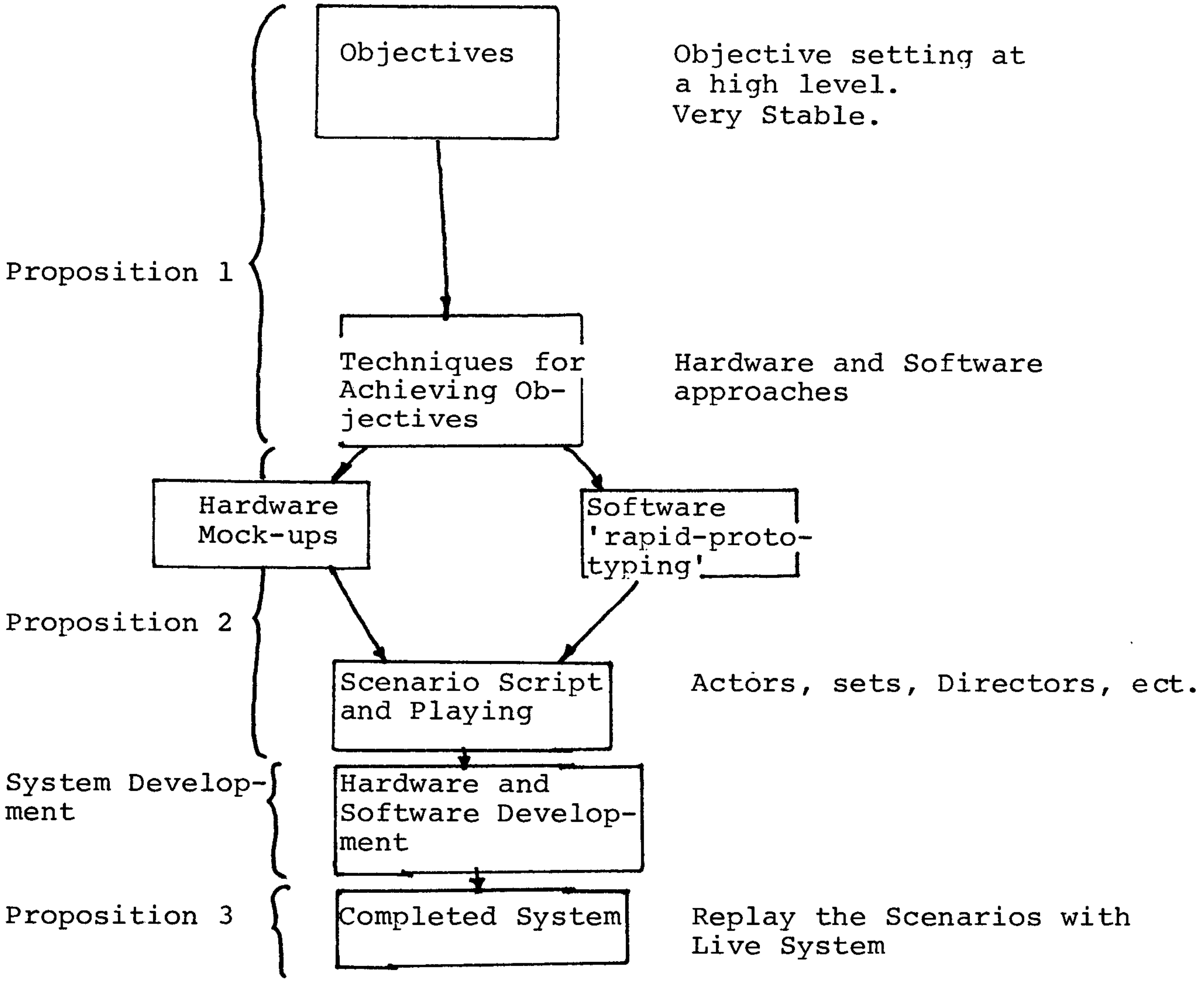
System objectives plus physical demonstrations will result in a successful product.

By a successful product I mean one that: (1) performs the function intended, and (2) satisfies the customer's percieved need. All parties are convinced of this early on because they have seen the 'system' at work.

I believe that most users do not have a concrete idea about their automation needs. If they could experience a live, albeit mocked-up system, the problem of wholesale requirement changes and delivered but unused software would soon disappear. Once a user has a 'warm' feeling for what he will recieve at the end of the system development, his confidence is increased. When the user feels this confidence only a philistine would care how the system was implemented. Once Proposition 1 is satisfied and Proposition 2 is executed, Proposition 3 is a natural result.

The Non-Cyclical (Hollywood) Model

If I were to construct a model of this software development process (which I have dubbed the Hollywood model in deference to 'Tinsel Town') I would render it as follows:


Objectives
Objective setting at a high level. Very Stable.
Proposition 1
Techniques for Achieving Objectives
Hardware and Software approaches
Hardware Mock-ups
Software 'rapid-prototyping'
Proposition 2
Scenario Script and Playing
Actors, sets, Directors, ect.
System Development
Hardware and Software Development
Proposition 3
Completed System
Replay the Scenarios with Live System

(Fig.2) Hollywood Model

Model Benefits

1. Objectives are more stable and change resistant than requirements.

2. All objectives can be stated succinctly before a project begins.

3. Customer anxiety and therefore a tendency to expand requirements is ameliorated by providing a working model of the system early in the game. A model is also easier to change if need be.

4. Flexibility in implementing a system is enhanced because the customer is convinced of what the system will do when delivered. Implementation is a 'don't care'.

5. Schedule is reduced along with errors because all participants understand the system objectives and changes to that understanding are reduced.

Notes and References

[1] Some statistics from the DPMA Software Management Conference earlier this year:

- 75% of the software development undertaken was never completed or not used if completed.

- Of the 75%, 25% was never delivered and 47% was delivered but not used.

[2] System Attribute Specification, Tom Gilb, Software Engineering Notes, July 1981, p. 78.

[3] Sam Harbaugh in Open Channel, Computer, February 1982, p. 97.

Building Services Division
Honeywell Inc. Technical Center
Irvine CA 92714

G. R. Gladden
Supervisor Quality Assurance