Evolutionary Delivery versus the "Waterfall Model"

by Tom Gilb, Independent Consultant,
Box 102, Kolbotn, Norway

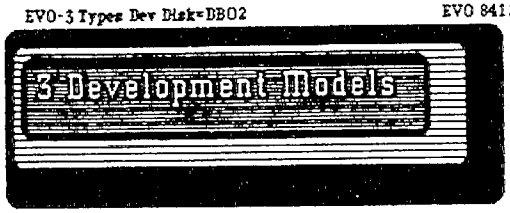
Short Abstract:

The conventional wisdom of planning software engineering projects, using the widely cited "waterfall model" is not the only useful software development process model. In fact, the "waterfall model" may be unrealistic, and dangerous to the primary objectives of any software project.

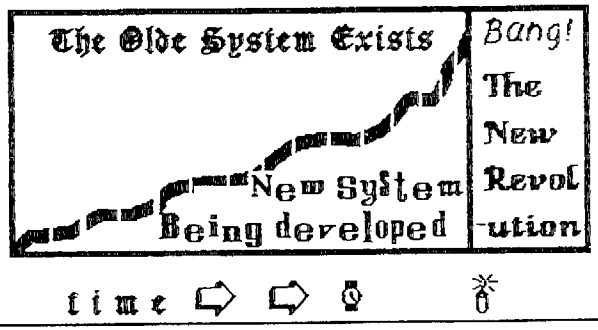
The alternative model, which I choose to call "evolutionary delivery" is not widely taught or practiced yet. But there is already more than a decade of practical experience in using it, in various forms. It is quite clear from these experiences that evolutionary delivery is a powerful general tool for both software development and associated systems development.

Almost all experienced software developers do make use of some of the ideas in evolutionary development at one time or another. But, this is often unplanned, informal and it is an incomplete exploitation of this powerful method. This paper will try to expose the theoretical and practical aspects of the method in a fuller perspective. We need to learn the theory fully, so that we can apply and learn it completely.

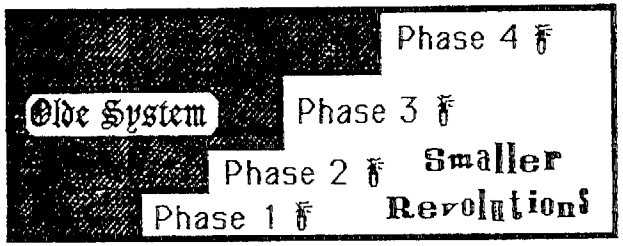
EVO-3 Types Dev Disk:DB02 EVO 841225.1700 TxG



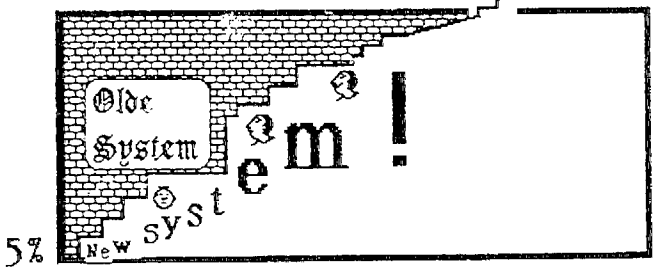
The "revolutionary" or "big bang" model.



The Phased delivery approach.



The "Evolutionary Delivery" Model



INTRODUCTION

Most textbooks, and most software engineering models today are based on the "waterfall model". This can take several dialects. But, the principal characteristic is that;

- all planning is oriented towards a single delivery date,
- all analysis and design are done in detail, before coding & test.

The delivery date is typically one or more years after project start. There may be some effort to improve the design by means of prototypes. But, these prototypes will typically be "throw away", and may see little or no real useful work by real users being done on them.

The evolutionary delivery ("EVO" method for short) is based on the following simple principle:

a. Deliver something to a real end-user;
b. measure the added-value to the user in all critical dimensions;
c. adjust both design and objectives based on observed realities.

This "eternal cycle" (Deming-85) starts early. It usually will attempt to modify some existing system, rather than to build a total new system. This solves the problem of "critical mass", of getting enough of a system together to give a design idea a realistic try.

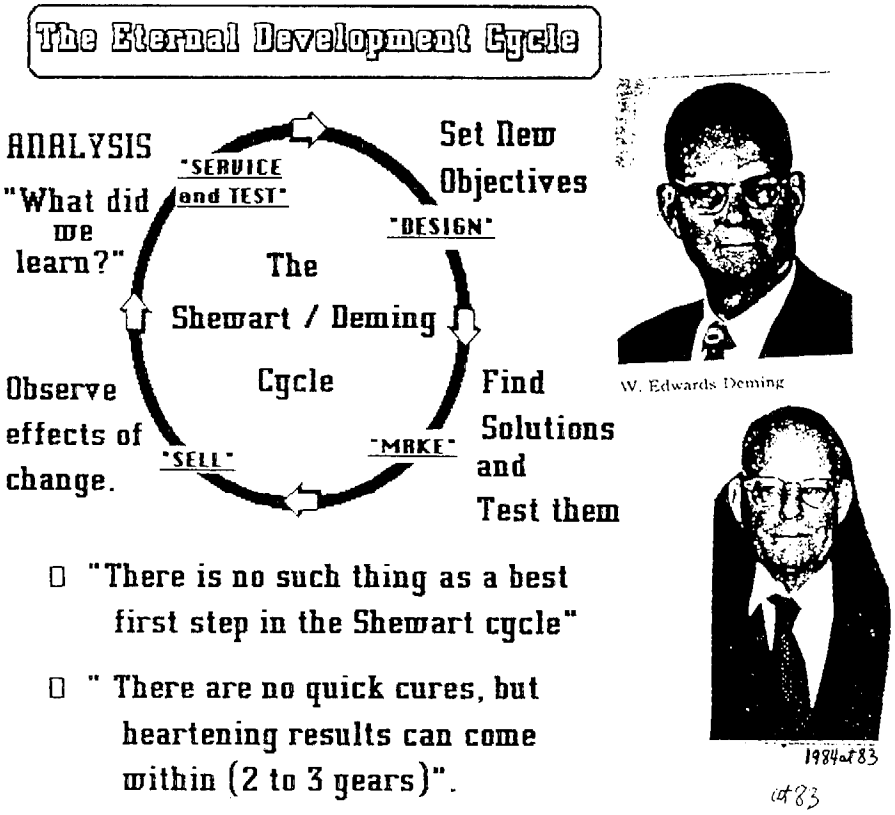
The basic EVO concepts are firmly rooted in other engineering literature, and in engineering practice. It is the software community which has been slow to recognize the potential of the EVO method and to exploit it fully.

A SIMPLE MODEL; A PC PROGRAMMER-USER.

The simplest software-area model of EVO is a personal computer user building an application for themselves interactively. The personal computer programmer is then both the user, the setter of objectives, the analyst, the designer, the coder, the tester. There is constant and early iteration as the system is built up. The iteration cycle is typically measured in minutes, not years. The system being built is real and evolving. The user can modify both design, coding detail and even their final objectives.

THE COMPLETE EVO MODEL.

There are many concepts which can be put in the basket we call evolutionary delivery. Each individual has a set of their own concepts. I will give my personal list of the factors which I believe are vital to



- "There is no such thing as a best first step in the Shewart cycle"
- " There are no quick cures, but heartening results can come within (2 to 3 years)".
- "By working in the cycle, everyone will see what he can do, and what only top management can do". (Team effort).
- " In use in Japan since 1950".
- "it is hampered in America by the annual rating of performance "(kills teamwork).

Modified from: Manuscript by W.Edwards Deming.
Out of Crisis"(working title) MIT Press 1985.See also Walter E. Shewart, STATISTICAL METHOD.. (GRAD. SCHOOL, DEPT.AGRIC.1939)

full exploitation of the method. A planner can choose to ignore some of these concepts, but I believe that the method will lose some of it's power if they do.

A planner may well find additional ideas to those which I mention here, which will increase their power in using the method. I find that I am constantly learning, by experience and by experimentation, new ways to use the basic concpts better. Thus, I do not pretend that my model is complete. It is however richer than most models which I have found in the literature or in practice.

Here is a list of the main critical concepts; this may be taken as a superficial definition of what I mean by "EVO" or evolutionary delivery planning.

1. MULTI-OBJECTIVE DRIVEN.

2. EARLY, FREQUENT ITERATION.

3. COMPLETE ANALYSIS, DESIGN, BUILD AND TEST IN EACH STEP.

4. DESIGN BY OBJECTIVES.

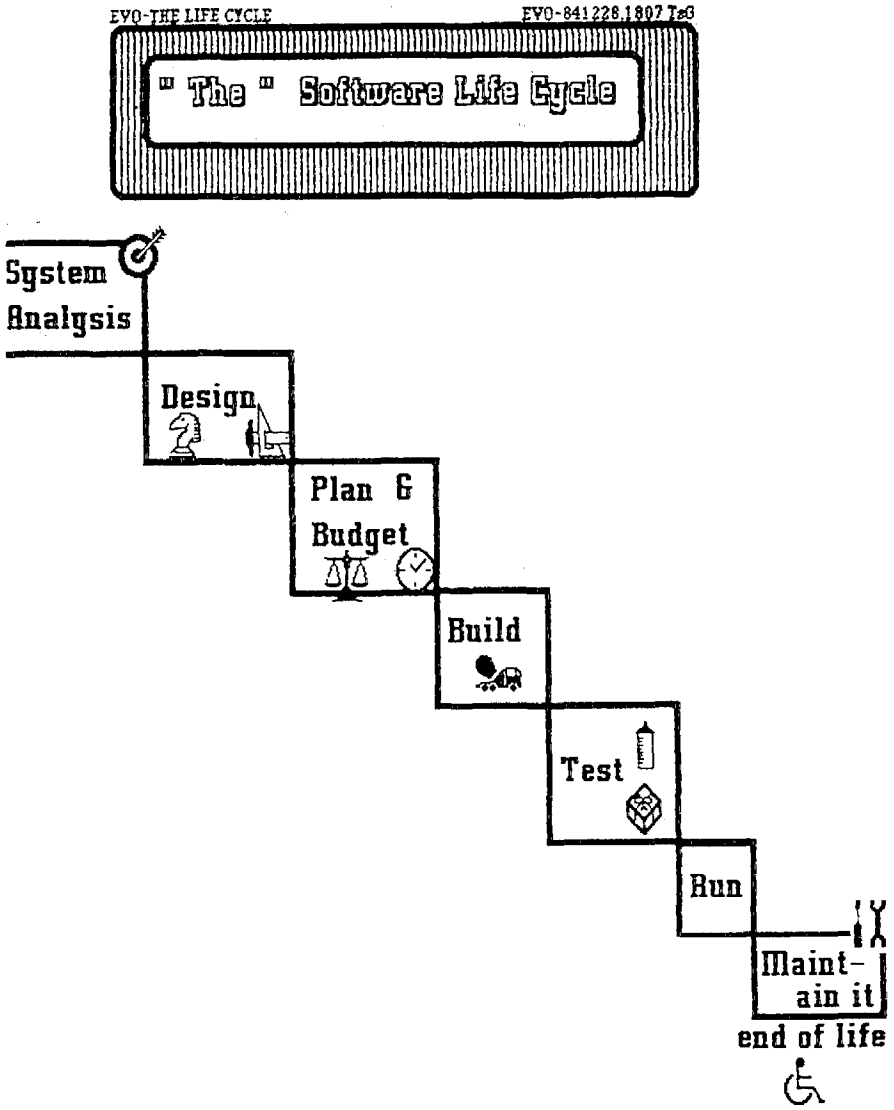
5. USER ORIENTATION.

6. SYSTEMS APPROACH, NOT MERELY ALGORITHM-ORIENTATION.

7. OPEN-ENDED BASIC SYSTEMS ARCHITECTURE

8. RESULT ORIENTATION, NOT PROCESS ORIENTATION

Let me treat each of these subjects in somewhat more detail below.



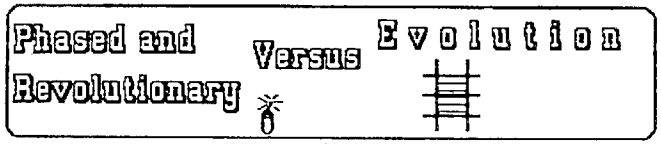
SOME DETAILS ON THE CRITICAL CHARACTERISTICS OF EVO-DELIVERY

1. MULTI-OBJECTIVE DRIVEN.

Conventional software planning is overwhelmingly "FUNCTION" oriented. The planning is in terms of the functional deliverables. These are more concerned with WHAT the software will do, rather than "HOW WELL?" (quality attributes) and "AT WHAT COSTS?" (the resource attributes). It is my firm belief that software engineering currently places too little emphasis on control of the critical quality and resource attributes of the system; and thereby loses control of these attributes. A simple example is the general ignorance among software engineers and teachers on such elementary subjects as how to define critical attributes like "usability" (see Gilb-IFIP-84 for a detailed example) or "maintainability (see Gilb-SSD-79). These subjects are today so vital to the success of most software projects, that ignorance of how to specify measures of them is roughly equivalent to an electonics engineer not knowing what volts and watts are.

Evolutionary delivery, in my view must be based on iteration towards extremely clear and measurable multi-dimensional objectives. The set of objectives must contain all functional, quality and resource objectives which are vital to the long-term and short term survival of the system being developed. (See Gilb-DBO, and Gilb-SM-76).

If this discipline is missing, and it is overwhelmingly missing from most every international software project I have encountered, though some are better than others, the evolutionary process becomes meaningless. The project is not, then, related to the vital real world needs of the user.



Initial budgets years/mill	1/10 -> 1/100th complete stp
Months->Years then deliv.	Days, weeks to results.
Short changeover to new.	Build away from present syst.
Human shock at change.	Time for adjustment.
Changes threaten system.	Adaptation is "native".
Big capital cost before ROI	Excellent cash-flow.
Req. changes unwelcome.	New requirements natural.
"Goldbricking"; uncritical	Small steps evaluated by ROI
Early years unprofitable.	Earliest steps deliver Hi ROI
Must finish to get any ROI	We can "stop in the middle"
The Bottom Line	
MGT. AT MERCY OF IMPL.	MGT. IN FULL CONTROL

2. EARLY, FREQUENT ITERATION.

In most software engineering projects, the planning schedules delivery of practical and useful results, one or more years away. There are a series of plausible excuses for this lack of confrontation with reality. But, in my view the excuses are invalid and due to professional ignorance. I have found that the original planners of such projects, are themselves the first to agree that there is, in fact, a real possibility of earlier delivery, which they had not yet considered. Their problem in finding early and frequent software delivery cycles is one of both "lack of motivation" and "lack of method".

The management involved would, of course, dearly love to get some results on the table as early as possible. But, even that same management, accepts the conventional wisdom of the long initial cycle, before even the first useful phase is delivered.

Often they believe that their "Phased" project does in fact give them the earliest possible delivery of something useful. My experience is that most first phases can be sub-divided into ten to one hundred smaller and earlier steps of useful delivery.

Phased planning asks a dangerous question: "How much can we accomplish within some critical constraint (budget, deadline, storage space)?"

Evolutionary Planning asks a very different question: "How little resource can we expend, and still accomplish something quite useful in the direction of our ultimate objectives?"

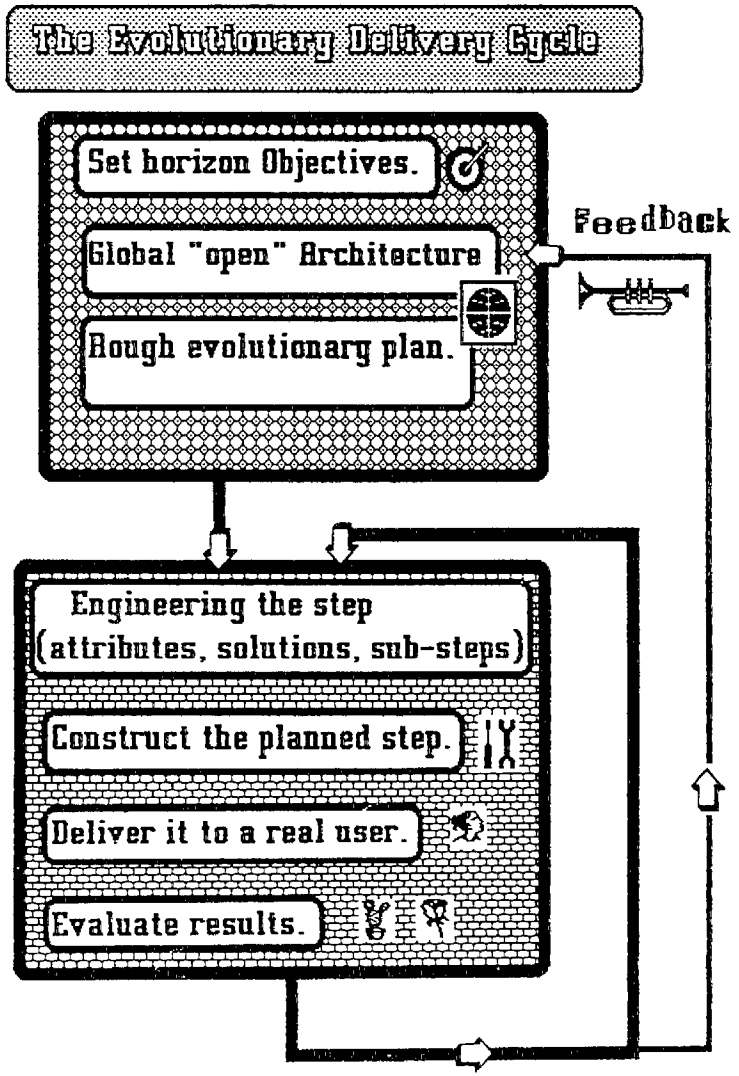
More formally, in EVO-planning today, we use the concept of selecting the potential steps with the highest **user-value to development-cost ratio** for earliest implementation. This is like skimming the cream off the top of the milk.

I would like to stress that selection of high value/cost steps is a fundamental difference in my formal conception of evolutionary planning, and that which I find elsewhere. I actually find that there is little or no conscious thought about this "cream" selection potential. Or, I find that the first phases are known to give little or no real value - but they are thought to be "pathways to the future". In one case which I experienced, in an European Airline project, the first 50 work-years of a 250 work-year project were concerned with building a database system, which gave absolutely no value in the direction of the critical objectives.

3. COMPLETE ANALYSIS, DESIGN, BUILD AND TEST IN EACH STEP.

One of the great time-wasters in software projects is detailed

EVO-loop Disk=DBO 2 EVO 841223.1910 TmG



requirements analysis analysis, followed by detailed design, followed by full coding and testing phases. We believe in it like a religion.
If we only had the intellectual capacity, and professional knowledge to really do those things accurately! We must admit that we cannot tackle such a task well, for any but trivially small projects. There are too many unknowns, too much dynamic change, and too complex a set of interrelationships in the systems we build. We must take a more humble approach!

We must set initial measurable **horizon objectives** (as far as we can reasonably have an opinion about the future. We must be prepared to modify these objectives, as soon as experience (of partial delivery !) dictates. We must design a suitable general architectural framework (I call it the "**Infotecture**") for enabling us to meet these objectives, in spite of obstacles along the way.

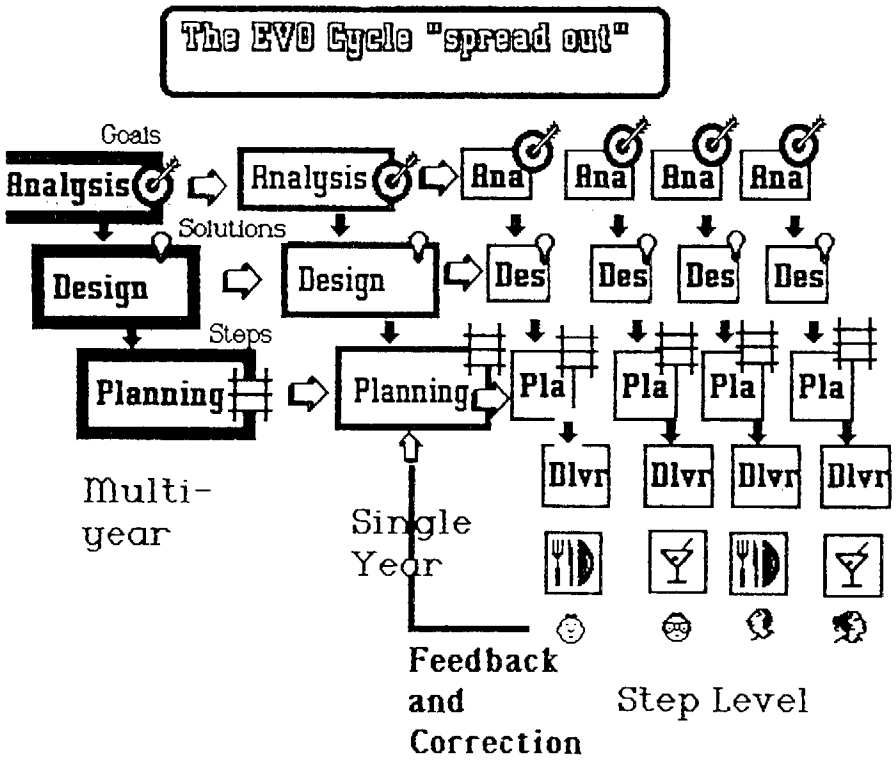
We must set measurable objectives for the next small **delivery step,** but even these are subject to modification as we learn. It is simply not possible to set an ambitious set of multiple quality, resource and functional objectives, and be sure of meeting them all as planned. We must be prepared for compromise and trade-offs. We must then design (engineer) the immediate technical solution. Build it, test it, deliver it - and get **feedback.** This feedback must be used to modify the immediate design if necessary, modify the major architectural ideas - if necessary, and modify both the short-term and the long-term objectives - if necessary.

It is silly to spend, nay "waste", so much time in the beginning of a project, to speculate on requirements and technical design attributes, which can be measured much more cheaply and reliably, if it is done while we implement a real system.

The major objection to this line of action is the fear, based on experience, that we will "paint ourselves into a corner", that when we get negative feedback, it will be too late. We will have committed too many resources to the wrong solution. This objection is not valid. Evolutionary delivery gives us early warning signals of impending unpleasant realities. Unpleasantries do occur, but never become too large. And, the key to reducing our major fear is that we must learn to design far more "open-ended" system architectures (I call them "open Infotectures").

We have perceived our problem as one of analyzing things in enough detail, before any construction takes place, to prevent construction errors. I assert that this is not as easy, or productive, as the evolutionary alternative. Start with a basic design which is easy to modify, adapt, port and change; both in the long and short term (we KNOW we need that anyway). Then jump in the water, and learn even more, even earlier - while being useful.

EVO-spread cycle Disk=DB02 EVO 841223.2110 TsG



Note: this illustrates the fact that with evolutionary delivery cycles, the various development activities are intermixed and spread out throughout the life cycle: each one feeding the others with practical insight and fact.

4. DESIGN BY OBJECTIVES ("DBO").

Evolutionary delivery can, in theory ,be practiced in splendid isolation from other methods. But, I would argue, as I have already begun to above, that there is a larger software development context in which it will be most fruitfully applied. For me personally, as a management and design consultant to large software producers, it is only one of several simultaneous methods which should be applied for fullest effect.

The major collection of methods which I recommend in this context, I give the umbrella title of "Design by Objectives" (See Gilb-DBO and other references by Gilb, most of which are related to this subject).

The major components of "DBO" are:

1. **System Attribute Specification**: Measurable multi-dimensional specification of critical objectives - as discussed above.

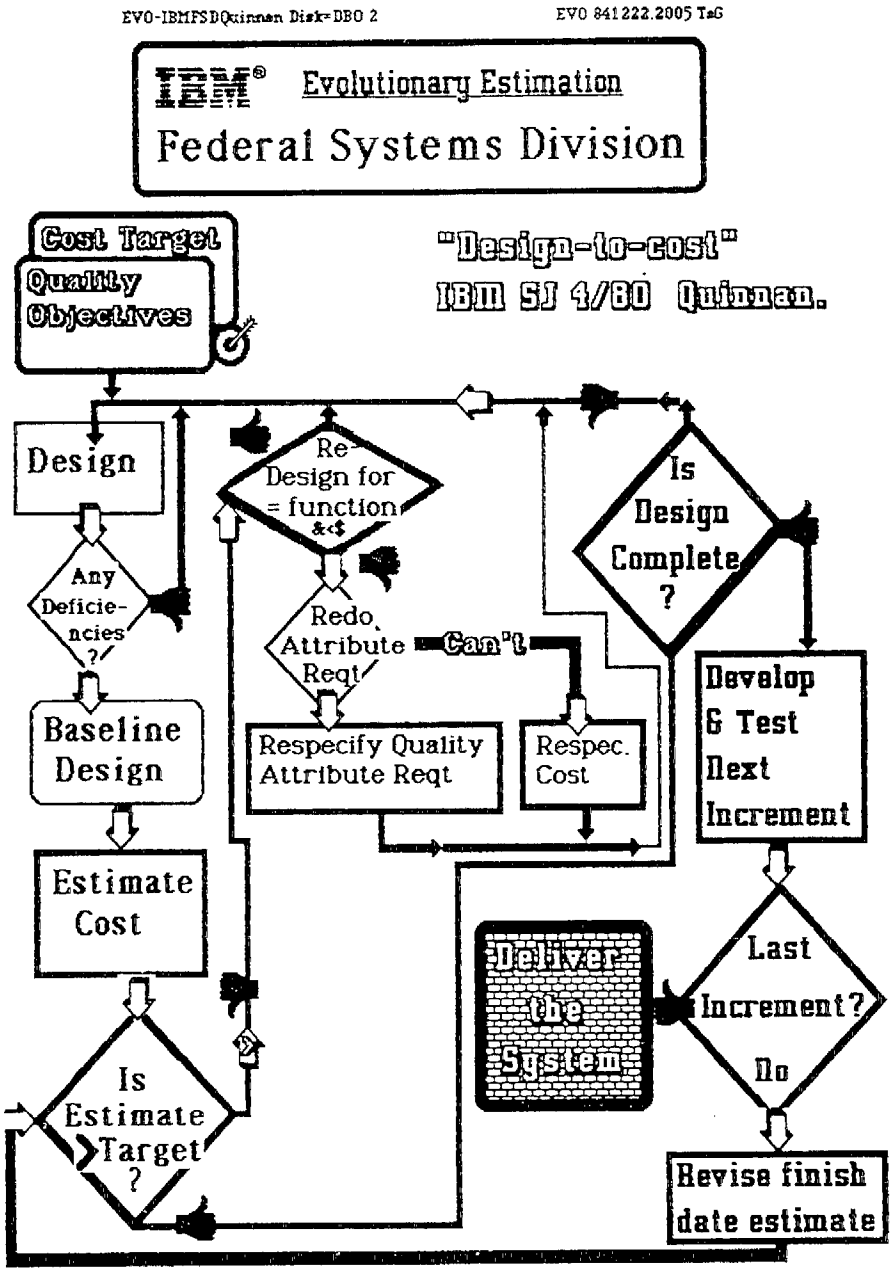
2. **System Functional Specification**: where functional concepts are broken down into a large number of possible deliverable ideas, and they are identified by "Tags" (for tracing deliverables, throughout the entire development process). Each deliverable has individually estimated value and development-cost factors, to help decide which deliverables should be done first, as discussed above.

3. **Software Engineering Handbooks**: are lists of all the methods, structures, techniques, languages and other potential design ideas at our disposal. Each idea has a documented set of expected values for attributes of potential interest to a design engineer. (See Gilb-SEN-SEH-81). This will increase our accuracy of making decisions about EVO steps and their design components.. It will reduce the probability of failure, and the need to re-work ideas.

4. **Fagan's INSPECTION:** Fagan's Inspection method needs to be applied to all parts of the software engineering process, in particular to the high-level design specification phases - where the evolutionary steps, and their underlying architectures, are planned. (Ref. FAGAN-76)

5. **Impact Estimation:** We need to apply a tool for estimating the qualititative impact of every design technique on every design objective. I have developed a simple table, which works well in electronic spreadsheet applications, for making and updating these estimates, and their uncertainty factors. (See Gilb-IFIP-84).

These are the central DBO tools to help EVO delivery work better.



5. USER ORIENTATION.

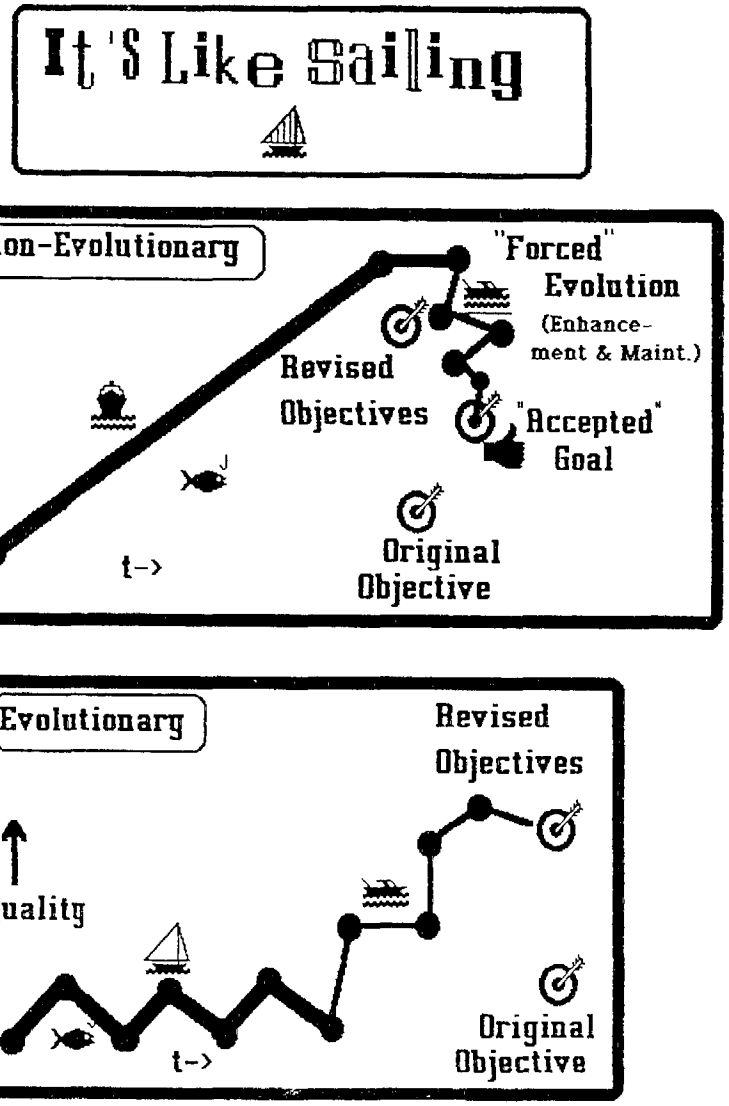
Software projects are not famous for adapting to what the market or user really needs or wants. The orientation is towards the machine, the algorithm, or the deadline - but too rarely the user. Many software developers literally never see their product in action with real users.

Most users never see the face of the designers and programmers. They don't even know their names. Even if the developers did want to make a product which the users were really happy with, it may be too late and too impractical to do so, by the time the developers find out what the user really wants.

With Evolutionary delivery, the situation is changed. The developer is specifically charged with "listening" to user reactions, early and often. The user can play a direct rôle in the development process. Neither the budget, nor the deadline are overrun. The overall system architecture is "open ended", and we are mentally, economically, and technically prepared to listen to what the user or customer wants.

The rule of selecting the highest available value-to-cost ratio step next is a dynamic one. User values can change or alter as they get experience, and user ideas of value can give the planners a flow of new ideas, not originally planned.

Every software developer experiences this feedback and changed ideas of value, design and know-how about real costs. We also know how this mechanism works once we get into maintenance phases. The question here is whether we are going to apply this learning and selection mechanism earlier, more consciously, and more frequently in the development process than we customarily do.



6. SYSTEMS APPROACH, NOT MERELY ALGORITHM-ORIENTATION.

Many of our software engineering methods have a common weakness. They are exclusively oriented towards current computer programming languages. They do not even treat software, in the broadest (i. e. "non-hardware") sense of that term. They do precious little about the Data Engineering (see Gilb-DE-76 and Gilb/Weinb-HI-83) aspects of software, and do even less about less-obvious concerns such as documentation, training, marketing, and motivation (See "Motivational Techniques for Reliability" in Gilb-DE-76). A contrast to all this will be found in the successful Apple Macintosh design effort - where everything was designed in a fully integrated manner (see Byte-2/84).

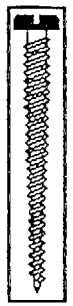
I frequently speculate that the real problem with software engineering is the lack of total architectural co-ordination of the total system design process, of which software is merely a part. I express my ideas by using the term "**Infotecture**" (for which I must thank the French). Infotecture is of course carried out by a trained "**Infotect**". The infotect is the leader of the entire design effort, and of course has a number of specialist **software engineers** working within the infotecture guidelines and under infotect control.

Evolutionary delivery is a method which is not merely limited to software, in the narrow sense of that word. It is admirably suited to the total process of creation in which we are involved. It insists that ivory tower programmers meet the real world early, often, and brutally - if necessary.

EVO-IBM FSD Mills Disk=DBO 2 EVO 841222.0835 T=G

IBM®
Federal Systems Division

- Harlan D. Mills, IBM SJ 4/1980
- "management has learned to expect on-time, within-budget deliveries"
- "LAMPS ...4 year ... 200 person-years in 45 incremental deliveries. Every one of those deliveries was on time and under budget."
- ".. NASA space program .... 7000 person-years software dev... few late or overrun .. in .. decade, and none at all in the past 4 years"
- "evolution in .. software eng. ideas... evolution in .. people using (them) evolution ... not without pain programming .. evolved to a precision design process ... software engineering has evolved from an undependable ..activity to a... manageable activity for meeting schedules..budgets..quality"



7. OPEN-ENDED BASIC SYSTEMS ARCHITECTURE

Software designers (most of whom we should not dignify with the title software **engineer**) seem to know only one single way, at most to build the functions we need. In some cases we know of no way, and fail. My vision of a really good design engineer, or of an architect is a person who knows a very wide variety of ways of solving a problem (see Frank Lloyd Wright's autobiography, Horizon Press NY, for an exciting insight into this).

A good designer can always list at least ten reasonable ways to solve a design problem. In addition, the design engineer has knowledge of the principle attributes of quality and resource which are the essential differentiators between the alternative solutions.

Among the principal attributes of any system, are those which allow it to survive and succeed under the inevitable changing conditions in the passage of time. A good software engineer, or infotect, should have made a detailed continuing study of the many available design technologies which lead to systems which are more adaptable than others. There are several measurable technical properties here - such as maintainability, portability and extendability (Gilb-SET-83).

We seem to me to be like a Greek-island house-builder who builds white-painted stone houses in the traditon of his forefathers, and will not admit to flexible technologies such as the sound-proof sliding-wall of the hotel conference-facility.

There is a clear need for a literature and teachings on the open-ended technological solutions - the hundreds of them. Not merely the "pop" approach of "standards", "de facto standards", and "structures". Old foxes learn by experience, but how shall we teach the young foxes what the old ones have learned so painfully - **in time** to avoid creating disasters ?

In terms of evolutionary delivery planning, open architectures are essential pre-requisites. Without them the effort will probably get caught in the all-too-familiar swamp of maintenance of today. The blame will be laid at the wrong door. "We didn't plan in enough detail before we strrarted coding!" they will say. But, the real truth is that it was always impossible to know so much complex detail in advance of real experience. The real truth is that they didn't have an open ended enough structure to tolerate the storms of change (see Alvin Toffler; Both "Future Shock" and "The Third Wave").

8. RESULT ORIENTATION, NOT PROCESS ORIENTATION

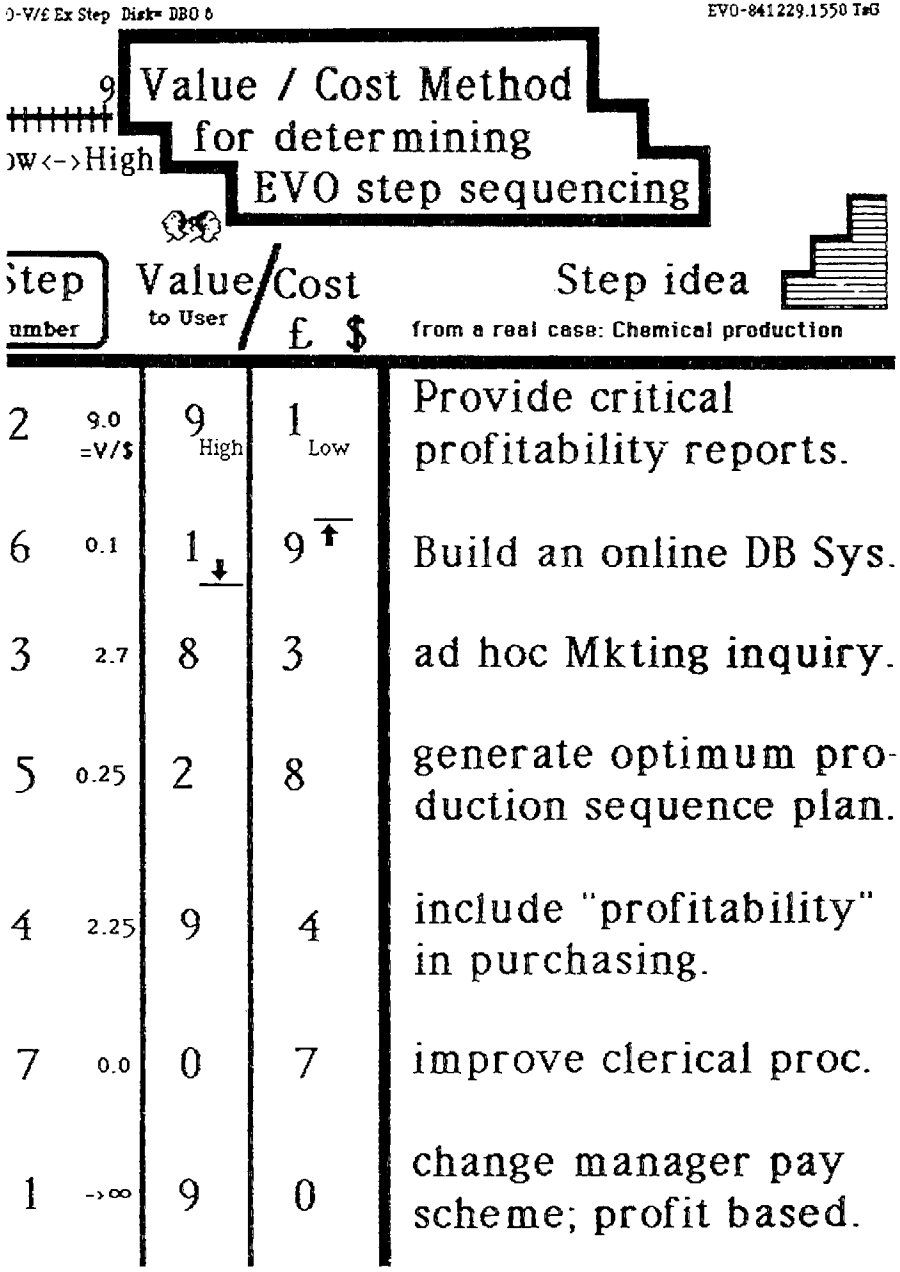
In traditional waterfall software development cycles, the process itself seems more important than the result. How often have I seen my clients software engineering people paralyzed by the formalities of a process, when there were clearly no clear objectives, towards which to steer that effort. The situation is so awful world-wide that I can be certain that all large software engineering efforts at present have extremely unclear, unmeasurable and unstated objectives in critical quality and resource areas. You can try "Usability" and "Maintainability" just for starters.

Evolutionary delivery forces the developers to get outside of the building process for a moment, frequently and early - and find out whether their ship is navigating successfully towards that port of call many cycles of delivery away. You cannot help be result oriented when confronted with the need to observe your designs and programs working with real working users.

CONCLUSION

Our software engineering community needs to take a long hard look at the evolutionary delivery method - by whatever name. They need to teach it to most praticing software professionals, to managers, and to the new people entering our ranks. We don't have to invent anything new - just to be a bit more humble in our approach to our task - and to apply wisdom which is thousands of years old. "Deal with little troubles before they become big" and "the longest journey began with a first step" (from "Tao Teh King" by Lao Tzu about 570 b.c., F. Unger Publishing, 1980). Unfortunately, I had to learn the theory by practical observation - and it took a long time. I hope others can get knowledge of this ancient wisdom at an earlier stage of their careers.

Finally, we must set this method into a wholistic context. It needs nourishment from clear goal setting, from Engineering Handbooks, and from multi-dimensional analysis techniques. It must function in a total systems environment. Software is not an island of self-sufficiency. Our dear algorithms can only produce interesting results in the company of a great many other parties. We must learn to evolve documentation, data designs, interfaces, hardware hosts, and not least people - their traditions and weaknesses. An evolutionary test cycle is nothing compared to the glory of improving the life of the user !



LITERATURE REFERENCES FOR THIS ARTICLE:

Boehm-SEE-81; B. W. Boehm, "**Software Engineering Economics**". This book covers incremental development "as a refinement of the waterfall model". Many of the principles of Evolutionary Delivery as described here, are present. But, it does not go as far as **Mill's(ref. below)**. It is more like a healthy phased delivery model.

BYTE-2/84: Cover story articles about the Macintosh development. Pages 30 and 58. Confirm the integrated, result-oriented, evolutionary process whereby a Mac becomes a Mac.

Deming-85, W. Edwards Deming. The man who brought successful US quality control methods to Japan, had as a principal idea the eternal cycle of development, which he credits his teacher, Walter E. Shewart ("Statistical method", 1939). Demings own recent work is "**Out of Crisis**" (MIT Press, 1985).

FAGAN-76: M. E. Fagan, "**Design and Code Inspections to Reduce Errors in Program development**", IBM Systems Journal, 15, No. 3 1976, pp 182-211. This article is cited because it laid the cornerstone, and is easily available. There is a wealth of follow-on literature available, especially from The Librarian, IBM, Kingston, New York. The essence of this article, and some additional technical data will be found in Gilb-SM. Inspection gives early control over design cycle products, even before evolutionary results appear.

Gilb-DBO: "**Design by Objectives**", Planned 1985, North-Holland.

Gilb-DE-76: Gilb, T., "**Data Engineering**", Studentlitteratur AB, Lund, Sweden. This book treats open-ended design ideas by directly and systematically exploring many attributes of data design including portability, extendability and maintainability. It also treats the subject wholistically, including "motivation" which is given a major chapter.

Gilb/Weinb.-HI: Gilb and Weinberg, G. M.,"**Humanized Input**:Techniques for Reliable Keyed Input", New edition 1983 QED Inc. MA, This book systematically describes the open-ended attributes of each technique.
Many of the techniques here have been used in evolutionary software projects by the authors to support ease of change from old to new.

Gilb-IFIP-84: Proceedings of the IFIP Conference on Human Machine Interactions ("Interact") Sept. 1984, London. My talk on "**The Impact Analysis (Estimation) Table in Human Factors design**" is included there. It gives a picture of the Impact Estimation tools mentioned here, and of a metric breakdown for "usablility", from my industrial practice.

Gilb-SEN-81-E :T. Gilb, "**Evolutionary Development**", ACM Software Eng. Notes, April 1981.

Gilb-SEN-SAS-81: ACM Soft. Eng. Notes. T. Gilb, "**System Attribute Specification**: A cornerstone of software engineering", p.78-79.

A List of things which must be delivered at every Evolutionary Delivery Cycle

GENERAL PRINCIPLE: Everything, without exception, which contributes to the completeness, usefulness and planned quality or resource attributes of the product at that stage.

1. All Functional configuration items planned for that cycle.

2. All quality levels planned for that cycle, at the planned level.

3. All resource objectives must be met at the planned level

4. Unpatched source code for all logic.

5. Compiled code in runable form (linked together)

6. Input test cases for all modules. Complete set; meeting test standards. All non-computer-readable cases (flipping switches etc.) to be recorded and included.

7. Module test output, for the above test cases, in magnetic form (for automatic compare when regression testing.

8. Cross-reference listing to all Configuration Items (CIs, Tags) which are exercised by the test cases. See test standards for cross-referencing.

9. Complete system specification, in detail, with CI Tags attached, in text processor format.

10. Complete set of user documentation,in text processor format, cross referenced with Tags of test cases which purport to test particular statements

11. Complete set of Integration Testing Test cases, outputs, Cross references; as indicated above for module testing. Anything necessary to easily and safely repeat the testing process, after modifications are made. All

12. Hardware configuration presumptions. A list of the exact hardware configuration for which this testing is applicable.

13. A signed statement that the testing for this cycle is complete and correct: by 1. Quality Control Officer. 2. Technical Controller., and 3. Project Leader.

14. [illegible] for whom this cycle is intended.

Software Engineering Notes, Oct. 1981, p. 30-31.

Gilb-SEN-84: T. Gilb,"**Software Engineering Using Design by Objectives**" ACM Software Engineering Notes, April 1984, pp 104-113.

Gilb-SET-83: "**Software Engineering Templates**", about 40 pages unpublished manuscript. Course documentation. Freely copiable. It defines a number of fundamental software metrics in hierarchical measurable form.

Gilb-SSD-79: Gilb, T. "**Structured Design Methods for Maintainability**" in Infotech State of the Art Report on Structured Software Development, 1979. See also Data Processing (UK) "**Maintaining Software Systems**", June 1984. Also extracts of it in **Gilbert-83**, pp186-192.

Gilb-SM-76: "**Software Metrics**", Winthrop USA (now out of print) and Studentlitteratur AB, Lund, Sweden (in print). Three pages (187, 214, 217) directly treat evolutionary development concepts. It is interesting as an early reference. Fagan's Inspection method is also treated extensively in the book (see Fagan-1976 reference here).

GILBERT-83: Gilbert, Philip, "**Software Design and Development**", SRA Publishers, 1983. Has several references to evolutionary development, and incremental delivery. He also has references to Design by Objectives - but, he does not quite put it all together, nor distinguish it clearly from phased delivery.

Mills, Harlan, Dyer, Michael, Quinnan: Articles on Evolutionary Delivery (also called iterative enhancement). IBM Systems Journal NO. 4, 1980.
IBM Federal Systems Division reports extensive successful use of the method.

Wong, Carolyn, "**A Successful Software Development**", IEEE Trans. on Software Engineering, Nov. 1984. The SDC experience with evolutionary delivery is stated in the context of a number of other successful methods used with it.