Chapter (15) of Tom Gilb, Principles of Software Engineering Management (scanned in by the publisher without illustrations) 1988, 14th Printing in 2012
http://www.amazon.co.uk/Principles-Software-Engineering-Management-Gilb/dp/0201192462

Contact TomSGilb@Gmail.com, www.gilb.com for many more recent papers,slides, books and case studies

Background, this book, and certainly this chapter is credited by leading Agile Gurus such as Kent beck as their inspiraiton for understanding iteration and incremental project management.

See for example:
See back end of this chapter for documentation.

Some deeper and broader perspectives on evolutionary delivery and related technology

- Introduction

The objective of this chapter is to show the extent of understanding of the idea of evolutionary delivery inside and outside of software engineering, to show that it is not a new or unappreciated idea.

¯15.1 Software engineering sources on evolutionary delivery

These follow in alphabetical order.

15.1.1 Allman and Stonebraker

Source: Eric Allman and Michael Stonebraker, UC Berkeley, 'Observations 011 the Evolution of a Software System', IEEE Computer, June 1982, pp. 27-32. (Oc1982 IEEE)

The authors led the development of a 75000 line C database system, for over six years, in a research environment, but ultimately having over 150 user sites.

'It seems crucial to choose achievable short-term targets. This avoids the morale problem related to tasks that appear to go on forever. The decomposition of long-term goals into manageable short-term tasks continues to be the main job of the project directors.

Short-term goals were often set with the full knowledge that the longer-term problem was not fully understood, and were retraced later when the issues were better understood. The alternative is to refrain from development until the problem is well understood. We 'found that taking any step often helped us to correct the course of action. Also, moving in some direction usually resulted in a higher project morale than a period of inactivity. In short, it appears more useful to "do something now even if it is ultimately incorrect" than to only attempt things when success is assured.

As a consequence of this philosophy, we take a relaxed view towards discarding code . . . our philosophy has always been that "it is never too late to throw everything away."' (p. 28)

'Our largest mistake was probably in failing to clearly pinpoint the change from prototype to production system.' (p. 32)

15.1.2 Balzer

Source: Robert Balzer, USC/Information Sciences Institute, 'Program Enhancement', in ACM Software Eng. Notes, August 1986, Trabuco Canyon Workshop position paper, pp. 66-67

> 'There are two reasons for such enhancements. The first is that no-one has enough insight to build a system correctly the first time (even assuming no implementation bugs). The second is that the mere existence of the system, and the insight gained from its usage, create a demand for new or altered facilities.'

Dr Balzer comments on two of the main reasons that the waterfall model cannot work well in most high-tech environments. Software is different from hardware in at least one major respect. It can be more cheaply reproduced (copied, ported, converted reused). The consequence of this is that, like music composition, each effort is essentially an attempt to create something very new. This implies that we are bound to be working with more unknown factors than the bridge builder. So, we must have some processes for exploring the unknown, like evolutionary delivery.

Source: William Swartout and Robert Balzer, USC/Information Sciences Institute, 'On the Inevitable Intertwining of Specification and Implementation', Comm. of ACM, July 1987, pp. 438~0

'For several years we and others have been carefully pointing out how important it is to separate specification from implementation. . . . Unfortunately, this model is overly naive, and does not match reality. Specification and implementation are, in fact, intimately intertwined because they are, respectively, the already-fixed and the yet-to-be-done portions of a multi-step development. It is only because we have allowed this development process to occur unobserved and unrecorded in people's heads that the multi-step nature of this process was not more apparent earlier.' . . . 'Every specification is an implementation of some other higher level specification. . . many developments steps . knowingly redefine the specification itself. Our central argument is that these steps are a crucial mechanism for elaborating the specification and are necessarily intertwined with the implementation. By their very nature they cannot precede the implementation.'(p.438) 'Concrete implementation. . . insight provides the basis for refining the specification. Such improved insight may (and usually does) also arise from actual usage of the implemented system. These changes reflect (also) changing needs generated by the existence of the implemented system.' (p. 439)

'These observations should not be misinterpreted. We still believe that it is important to keep unnecessary implementation decisions out of specifications and we believe that maintenance should be performed by modifying the specification and reoptimizing the altered definition. These observations indicate that the specification process is more complex and evolutionary than previously believed and they raise the question of the viability of the pervasive view of a specification as a fixed contract between a client and an implementer.' (p. 439)

15.1.3 Basili and Turner

Source: Victor R. Basili, University of Maryland, and Albert J. Turner, Clemson University South Carolina, 'Iterative Enhancement: A Practical Technique for Software Development', IEEE Trans. on Software
Engineering, December 1975, pp. 390-396. (OC1975 IEEE)

'Building a system using a well-modularized top-down approach requires that the problem and its solution be well understood. Even if the implementers have previously undertaken a similar project, it is still difficult to achieve a good design for a new system on the first try. Furthermore, the design flaws do not show up until the implementation is well under way so that correcting problems can require major effort.

One practical approach to this problem is to start with a simple initial implementation of a subset of the problem and iteratively enhance existing versions until the full system is implemented. At each step of the process, not only extensions but also design modifications can be made. In fact, each step can make use of stepwise refinement in a more effective way as the system becomes better understood through the iterative process. This paper discusses the heuristic iterative enhancement algorithm.' (p. 390)

They recognize that evolutionary progress is made by a combination of function ('extensions') and solution ('design modification') enhancement.

'A "project control list" is created that contains all the tasks that need to be performed in order to achieve the desired final documentation. At any given point in the process, the project control list acts as a measure of the "distance" between the current and final implementations.' (p. 390)

'The project control list is constantly being revised as a result of this analysis. This is how redesign and recoding work their way into the control list. Specific topics for analysis include such items as the structure, modularity, modifiability, usability, reliability and efficiency of the current implementation as well as an assessment of the goals of the project.' (p. 391)

From this it is clear there is a dynamic revision of the design based on a multi-dimensional quality goal analysis. This is therefore quite close to the method described in this book. It is worth noting that Basili cites Harlan Mills and Parnas, both at one time colleagues of his.

'A skeletal subset is one that contains a good sampling of the key aspects of the problem, that is simple enough to understand and implement easily, and whose implementation would make a usable and useful product available to the user.' (p. 391)

This last sentence is explicit recognition of the value-to-cost step selection heuristic we recommend.

'The implementation itself should be simple and straightforward in overall design and straightforward and modular at lower levels of design and coding so that it can be modified easily in the iterations leading to the final implementation.' (p. 391).

This sentence is recognition of the factor that we have called 'open-ended design'.

'It is important that each task be conceptually simple enough to minimize the chance of error in the design and implementation phases of the process.' (p. 391) 'The existing implementation should be analyzed 'frequently to determine how well it measures up to project goals.' (p. 391)

It is clear that Basili and Turner are of the 'small is beautiful' school.

'User reaction should always be solicited and analyzed for indications of deficiencies in the existing implementation.' (p. 391)

Thus user experience played a major role not only in the implementation of the software project (i.e. the compiler) but also in the specification of the project (i.e. the language design). No doubt that the process is designed to make use of real user feedback. The authors go into some detail about a case study and even present a full table of preliminary numbers regarding the effectiveness of the technique!

'The development of a final product which is easily modified is a by-product of the iterative way in which the product is developed.' (p.395)

This is explicit recognition of the observation that the mere use of an evolutionary development process promotes frequent designer awareness of the practical need for open-ended and otherwise easily modifiable design.

'Thus, to some extent the efficient use of the iterative enhancement technique must be tailored to the implementation environment.' (p. 391)

15.1.4 Boehm: the spiral

Source: Barry W. Boehm (TRW Defense Systems Group), 'A Spiral Model of Development and Enhancement', ACM SIGSOFT Software Eng. Notes,
Vol. 11, No. 4, August 1986, pp. 14-24 (Proceedings of International Workshop on the Software Process and Software Environments, Trabuco Canyon CA 27-29 March 1985, ACM Order 592861)

Barry Boehm has a simple 'incremental step' evolutionary delivery model included in his Software Engineering Economics book. In 1985 he presented his spiral model to give more detail to this idea. The spiral model is not, however, in any sense identical to the evolutionary delivery model explored in this book. It is, it seems, a framework for including just about any development model which seems appropriate to the risk levels in the project at hand, or in particular components at particular points in the development process. The spiral model could be viewed as a framework for choosing evolutionary delivery as a strategy, or deciding not to choose it and to choose a traditional waterfall model, or other alternative instead. The spiral model, as befits the author's industrial background in military and space contracting in the US, shows due consideration to current political considerations and traditions or standards to which a large contractor might be subjected. The spiral model might also offer a politically viable way to convert from a waterfall model dominated environment into a more evolutionary environment, without having to make a major formal shift of direction. Here are Dr Boehm's own words on the subject:

'The spiral model['s] . . . major distinguishing feature . . . is that it creates a risk-driven approach for guiding the software process, rather than a strictly specification-driven or prototype-driven process.' (p. 14)

'One of the earliest software process models is the stagewise model (H. D. Benington, 'Production of Large Computer Programs,' Proc. ONR Symposium 011 Adv. Prog. Meth. for Dig. Comp., June 1956, pp. 15-27, also available in Annals of the History of Computing, October 1983, pp. 35~361). This model recommends that software be developed in successive stages (operational plan, operational specifications, coding specifications, coding, parameter testing, assembly testing, shakedown, system evaluation).' (p. 14)

'The original treatment of the waterfall model given in Royce (W.W. Royce, 'Managing the Development of Large Software Systems: Concepts and Techniques', Proc. WESCON, August 1970. Reprinted in Proc. 9th International Software Engineering Conf., 1987, Monterey, Calif., IEEE) provided two primary enhancements to the stagewise model:

- Recognition of the feedback loops between stages, and a guideline to confine the feedback loops to successive stages, in order to minimize the expensive rework involved in feedback across many stages.
- An initial incorporation of prototyping in the software life cycle, via a 'build it twice' step running in parallel with requirements analysis and design.'

The waterfall approach was largely consistent with the top-down structured programming model introduced by Mills (H.D. Mills, 'TopDown Programming in Large Systems', in Debugging Techniques in Large Systems, R. Ruskin (ed.), Prentice-Hall, 1971, pp. 12~137). However some attempts to apply these versions of the waterfall model ran into the following kinds of difficulties: the 'build it twice' step was unnecessary in some situations . . . ; The pure top-down approach needed to be tempered with a 'look ahead' step to cover such issues as high-risk, low-level elements and reusable or common software modules.

These considerations resulted in the risk-management variant of the waterfall model discussed in B.W. Boehm, 'Software Design and Structuring', (1975) in Practical Strategies for Developing Large Software Systems, E. Horowitz (ed.), Addison-Wesley, pp. 10~128 and elaborated in B.W. Boehm, 'Software Engineering', IEEE Trans. Computers, December 1976, pp. 122~1241. In this variant each step was expanded to include a validation and verification activity to cover high-risk elements, reuse considerations, and prototyping. Further elaborations of the waterfall model covered such practices as incremental development in J.R. Distaso, 'Software Management: a Survey of the Practice in 1980', IEEE Proc. September 1980, pp. 110~1119.

Boehm continues to note further alternatives to the waterfall model developed to cope with its weaknesses, but he finds weaknesses with each of these approaches, which he tries to resolve using the spiral model.

How does the spiral model relate to this book?

Note that Boehm is suggesting doing the kinds of activities which in this book we would call impact estimation and impact analysis, high level inspection of design, as well as what we would also try to discover by means of actually delivering small evolutionary steps, to see how things worked in practice, and to identify possible risk elements. Boehm suggests that any appropriate techniques can be used for this risk analysis phase. His model is open to all useful tools. His basic advice is to choose the appropriate next step based on 'the relative magnitude of the program risks, and the relative effectiveness of the various techniques in resolving the risks.'

I would argue that the evolutionary delivery process together with the set of software development and software project management tools and principles in this book is a complete set of tools for making the decisions about risk which the spiral model attempts to tackle. I cannot see that the spiral model adds anything necessary to the development process. This is not to say it is not useful, especially in the environmental context which Boehm was in where a large bureaucracy is emerging from the waterfall model situation. Boehm was trying to 'patch' the existing culture and to be diplomatic with his professional peers. There is necessary virtue in this, of course, but it is a problem with which only some of our readers must contend.

What does the spiral model not specifically incorporate?

Of course the spiral model, in admitting the use of any ideas, past, present, or future, doesn't need to specifically incorporate anything, yet can claim that anything necessary is acceptable. However I find that the following elements of evolutionary delivery, as preached in this book are missing from the spiral model:

- The concept of producing the high-value-to-low-cost increments first. Cumulation of user value. (The spiral model is so dominated by risk consideration that value concepts are not directly mentioned, except in the form of objectives and constraints, yet risk is risk of not getting value for money.)
- The concept of actually handling over to users usable increments, at 1% to 5% of project total budget.
- The concept of intentionally limiting step size to some maximum cycle of a week, month or quarter of a year.
- The concept of constantly being prepared to learn from any and all of the frequent step deliveries, and in so doing, being prepared to change any requirement, or any technical design solution, or work process, or objective in order to satisfy the users' current priority needs.
- The concept that productivity is measured by incremental progress towards and planned increment of either function, quality or resource reduction.
- The concept of open-ended architecture as a desirable base for evolution.

15.1.5 Brooks

Source: F.P. Brooks, The Mythical Man-Month, Addison-Wesley, 1975

'Fred Brooks presented some thoughts on the traditional life cycle, arguing for "growing," rather than building software: making a skeleton run (attributed to Harlan Mills), and the progressive refinement of design (Wirth). He suggested that software projects must be nursed and nurtured, and that you should plan to throw one version away, even if you do so part by part. The traditional life cycle was useful primarily for building batch applications. Today most systems are interactive and they require changes in the life cycle. The life cycle should be divided into three segments, with iterations occurring within each of the segments. The first segment is a requirements segment, design specification, and user manual. The next segment is the design, coding of a "minimal driver," and debugging of this initial skeleton of the application. In the next segment, functional sub-routines are coded, debugged, and integrated with the main system.

Benefits of this approach: it supports a progressive refinement of specifications which is better suited to interactive systems. It facilitates the concept of rapid prototyping and much greater interaction with users. It is better suited to the idea of "throwaway" code since you can deal in smaller functional elements and can redo them more easily if some problem becomes apparent. This approach improves the morale of the developers since they can see results more quickly and more directly related to their efforts.' (from Data Processing Digest, 8/84 p. 11 and System Development, 4, May 84)

15.1.6 Currit, Dyer and Mills IBM FSD

Source: P. Allen Currit, Michael Dyer and Harlan D. Mills, 'Certifying the Reliability of Software', IEEE Trans. on Software Engineering, Vol. SE-12, No. 1, January 1986,, pp. 3-11. (Qc1986 IEEE).See also IBM Systems Journal no 1 1994 for an update on the 'cleanroom' method.

This work needs to be looked at in light of the work of Mills, Dyer, and other IBM Federal Systems Division authors in IBM Systems Journal, (4)1980, reported earlier in this book, on evolutionary delivery. Their work here shows the slow but predictable exploitation of the evolutionary delivery method (they prefer the term 'incremental development' as they are not releasing software to their real users at each increment) to control other aspects (in this case reliability) than the time and cost factors which dominated their earlier work.

'This paper describes a procedure for certifying the reliability of software before its release to users. The ingredients of this procedure are a life cycle of executable product increments, representative statistical testing, and a standard estimate of the MTTF (mean time to failure) of the product at the time of its release.

The traditional life cycle of software development uses several defect removal stages of requirements, design, implementation, and testing but is inconclusive in establishing product reliability. No matter how many errors are removed during this process, no one knows how' many remain. In fact, the number of remaining errors tends to be academic to product users who are more interested in knowing how reliable the software will be in operation, in particular how long it runs before it fails, and what are the operational impacts (e.g. downtime) when it fails.

On the other hand, the times between successive failures of the software as measured with user representative testing are numbers of direct management significance. The higher these inter-fail times are, the more user satisfaction can be expected. In fact, increasing inter-fail times represents progress towards a reliable product, whereas increasing defect discovery may be a symptom of an unreliable product.

To remove the gamble from software product release, a different life cycle for software development is suggested in which the formal certification of the software's reliability is a critical objective. Rather than considering product design, implementation, and testing as sequential elements in the life cycle, product development is considered as a sequence of executable product increments. . . . A life cycle organized about the incremental development of the product is proposed as follows: . . . increments (and product releases) accumulate over the life cycle into the final product.'

They suggest the use of an 'independent test group' who will be 'responsible for certifying the reliability of the increments . . .' This independent test group has the character of a user group, and indeed could be a real user of some friendly nature. They then go on to point out that they recommend testing from the standpoint of user frequency of operations.

They are aware of the narrow scope of their activity: 'There will be other properties - such as modularity or portability - that are not considered.' By modularity they probably intend to refer to modifiability and with typical current confusion of ends and means, mention one solution to it, modularity.

The article deserves to be read in its entirety by any serious manager of software engineering. My main point in quoting it here is to point out how the evolutionary delivery cycle can be combined with reliability management.

It seems obvious that any attribute of the system can be similarly controlled. It also is clear that the reader may choose to deliver increments directly to some real users at each increment, rather than to an independent in-house certification test team.

15.1.7 Dahle and Magnusson

Source: Swedish language article in Nordisk Datanytt 17/86 pp. 40-13, 'Programmeringsomgivninger' (Software Environments), by Hans Petter Dahle (Inst. for Informatikk, University of Oslo), and Boris Magnusson (Lund's Engineering University)

Resources: an English report Mjølner - 'A Highly Efficient Programming Environment for Industrial Use,' edited by H. P. Dahle et al., Mjølner Report No. 1, available from Norsk Regnesentral, Forskningsveien lb, Blindern, Oslo 3, Norway

Here is my translation of their remarks concerning evolutionary delivery:

'In traditional development environments we have created methods based on a "batch" mentality. These use names like "life cycle model" and "the waterfall model".

In each step one or more documents are produced which are then

```
REQUIREMENTS ANALYSIS
     REQUIREMENTS SPECIFICATION
          SYSTEM DESIGN
               IMPLEMENTATION
                    TESTING
                         MAINTENANCE
                              TERMINATION
```
The traditional software development model

used as the input to the next step. This model is coupled at times with more or less formal methods being used at each individual step. The model has been shown to bear fruit for problems which admit formalization, which can be specified in a formal language, which - in other words - can be fully understood in all its components.

The method is less useful for situations where the requirements are less clearly specified, for example by an inexperienced customer, or by vague specifications such as "the response time shall be satisfactory." The non-formalized requirements get discovered late in the development process. A completely different problem is that a change involves updating of a number of documents - which is often a time-waster and an unpleasant job which doesn't always get done.

The first integrated software development environments were developed at research centers. The environments usually supported a particular programming language. Smalltalk and Interlisp were among the first complete program development environments, both developed at Xerox Palo Alto Research Center.

These and similar systems are coupled to a software development model which aims to get an early "prototype" of the object system operational with limited function. On the basis of experience from using the prototype, one can incrementally improve and finally deliver a product which satisfies the (ultimately) clarified requirements. This method is occasionally called "explorative programming."

The fact that the software can have bugs is considered of less importance than the ability to try out changes.

This working environment is very fruitful when solving problems which are not perfectly defined, and where all requirements can not be formally specified.

Of course the methods can be combined. A prototype can be made initially to map the requirements, and the traditional development model can be used to produce a final version.'

This quotation is from a fairly narrow context of advanced programming environments. It is included because it recognizes explicitly the need for an evolutionary delivery model of some kind.

15.1.8 Dyer

Source: Michael Dyer, IBM Federal Systems Division, 'Software Development Processes', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 451-465

Michael Dyer is one of the core team led by Harlan Mills which implemented evolutionary delivery, and reported it in public literature, on a larger industrial scale than any other group. Here are some quotations from his article which shed additional light on the exact process used.

'Each increment is a subset of the planned product.' (p. 458) 'The software for each increment is instrumented for measurement of such system resources as primary and secondary storage utilization.' (p. 459)

'As these actual performance measurements become available, software simulations that may have been initialized with estimates should be continually calibrated to enhance their fidelity.' (p. 459)

This recommendation is a direct reference to the ability of evolutionary delivery to improve our estimating and prediction capability. It was also used in reliability estimation in later years (see Currit, in this chapter).

'Software integration plans are recorded in controlled documents containing the following minimum information:

- scheduled phasing of the integration increments;
- system functions included in each increment;
- test plans to be executed for each increment . .

- support requirements for each increment in terms of system hardware simulation, tools and project resources;
- criteria for demonstrating that the increment is ready for integration . . . the exit condition from the unit test; quality assurance plans for the tracking and follow-up of errors discovered during the integration process.' (p. 462)

'A group separate from the software developers should have responsibility for planning the software integration process, for developing the integration procedures, and for integrating the software according to these procedures.' (p. 463)

'Control is achieved by careful system partitioning, incremental product construction, and constant product evaluation.' (p. 465)

15.1.9 Eason

Source: Ken Eason, HUSAT, Loughborough, UK, 'Methodological Issues in the Study of Human Factors in Teleinformatic Systems', Behaviour and Information Technology, Taylor & Francis, UK, 1983, Vol. 2, No. 1, pp. 357-364

'One of the best ways of achieving action research and active collaboration between technical and social scientists is to follow an evolutionary process of design . . . In this process early versions of the system are implemented, user responses assessed, the system revised, enhanced, etc. the new version implemented. . . . If this iterative process is not present in design the result will probably be that technical staff dominate design and, subsequently, evaluations are conducted by human and social scientists. The latter will consequently have no impact on the former.' (p. 363)

15.1.10 Gilb

Source: T. Gilb, Software Metrics, October 1976, (Winthrop).

'Evolution is a designed characteristic of a system development which involves gradual stepwise change.' (p. 214)

On step results measurement and retreat possibility

'A complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a "retreat" possibility to a previous successful step upon failure.' (p. 214)

On minimizing failure risk, using feedback, correcting design errors

'The advantage is that you cannot have large failures. You have the opportunity of receiving some feedback from the real world before throwing in all resources intended for a system, and you can correct possible design errors before they become costly live systems.' (p. 214)

On total project time

'The disadvantage is that you may sometimes have to wait longer before the whole system is functioning. This is offset by the fact that some results are produced much earlier than they would be if you had to wait for total system completion. It is also important to distinguish between a date for total system operation and a date for total "successful" system operation.' (p. 215)

On the general applicability

'Many people claim that their system cannot be put into operation gradually. It is all or nothing. This may conceivably be true in a few cases . . . I think we shall find that virtually all systems can be fruitfully put-in in more than one step, even though some must inevitably take larger steps than others.' (p. 215)

A measure of degree of evolution

'A metric for evolution is degree of change to system "S" during any time interval "t".' (p. 214)

On risk and predicting requirements

'Risk estimates plus/minus worst case are key to selection of step size', and 'Saving of analysis of future real world'. (p. 217)

The first remark is recognition that step sizing is determined by the need to control risk of failure. It is not small steps in themselves which are important. A large step may be taken if the risk is under control; for example by using contract guarantees or known technology. The second remark is recognition that the evolutionary method avoids the need to predict requirements and environments in the future; it allows us to wait until the future has arrived, to see the current requirements and the current environment.

On the scientific experiment analogy

'The concept of stability (where evolution is a technique for achieving stability) at individual levels of a system has the same usefulness as the concept of keeping all-factors-except-one constant in a scientific experiment. It allows systematic and orderly change of systems where the cause and effect may be more accurately measured without interfering factors, which may cause doubt as to the reason for good or bad results.' (p. 217)

'Systems may be specifically designed to go through a revolution in several phases, where only one level of the system is changed significantly at a time.' (pp. 217-8)

Evolutionary modularity design: conflict and priority

On p. 187 I raised the issue of 'Modularity division criteria', and gave six examples of rules for dividing software modules. Rule six was 'By calendar schedule of need of module' and the explanation for this rule was: 'Early implementation; evolutionary project develop.'

'Each rule can conflict with other modularization rules and with other design criteria. Resolution of the conflict can be achieved by a clearly stated set of priorities'.

This is a forerunner to the present perception of step design and selection being based on those elements of the total system which will contribute the greatest value towards stated objectives at the least development resource cost.

Later writings on the subject

The evolutionary idea was developed in my articles in the trade press: 'Evolutionary Planning and Delivery: an Alternative', Computer Weekly, 2 August 1979; 'Evolutionary Planning can prevent Failures', Computer Data, Canada, April 1979, p. 13; 'Realistic Time/Cost Data', Computer Weekly, 16 August 1979; 'Eleven Guidelines for Evolutionary Design and Implementation', Computer Weekly, 12 March 1981; and 'The Seventh Principle of Technology Projects: Small Steps will Result in Earlier Success', Computer Weekly, 30 July 1981. In all there were about 122 Gilb's Mythodology Columns in Computer Weekly (UK), which developed many of the ideas in this book.

15.1.11 Glass

Source: Robert L. Glass, 'An Elementary Discussion of Compiler/Interpreter Writing', ACM Computing Surveys, Vol. 1, No. 1, March 1969,
pp. 55-77

'Chronological Development
In the case of the SPLINTER interpreter, two facts dominated the chronology:

1. The processor was to be developed incrementally.

2. Some of the building blocks were available from other, previously developed processors.

The first fact meant that the initial development goal was to reach a minimally usable level of implementation in a minimal amount of time. The assumption was that with a well-modularized system design, the clutter which often comes with systems development conducted in this add-on fashion could be avoided.' (p. 65)

'It is the opinion of this author that incremental development is worthwhile. Reaching system usability early in development leads to a more thorough shakedown, avoids implementer and management discouragement and/or disinterest, and allows the user to get "on the air" in minimum time. . . . However, incremental development demands careful planning of the basics, especially table and list formats and modular construction, if it is to avoid resembling a house made of a packing case with rooms tacked on helter-skelter as they become needed.' (p. 68)

Open-endedness and the original 'stub'

'The SPLINTER processor has been built incrementally via an open-ended design process. Because of this there are always loose ends in the system that have not been implemented. lMPDEL, a general purpose subroutine, magically handles all these problems. (lMPDEL merely prints IMPLEMENTATION DELAYED as a diagnostic and returns control to the normal logic stream).' (p. 73)

This paper is particularly interesting because of its early date, beating even Basili and Turner by six years. It must be one of the earliest clear published recognitions of evolutionary delivery methods in the computer business.

15.1.12 Jackson and McCracken

Source: Michael A. Jackson and Daniel D. McCracken, 'Life Cycle Concept Considered Harmful', ACM Software Eng. Notes, Vol. 7, No. 2, April
1982, pp. 29-32

At a conference in September 1980 (at Georgia State University), these two well-known authors developed a 'minority dissenting position,' which eventually became this paper.

'To contend that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous. (p. 30)

'The life cycle concept perpetuates our failure so far, as an industry, to build an effective bridge across the communications gap between end-user and systems analyst. In many ways it constrains future thinking to fit the mold created in response to failures of the past.' (p. 30)

'It ignores . . . an increasing awareness that systems requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't even know them in advance - not even in principle.' (p. 31)

'We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose.' (p. 31)

The authors eloquently point out that the life cycle is obsolete. They do so at a time when most others are starting to adopt the idea. They do not suggest a particular remedy.

15.1.13 Jahnichen and Coos

Source: Stefan Jahnichen and G. Goos. GMD Research Center. Karlsruhe, 'Towards an Alternative Model for Software Developments', ACM Software Eng. Notes, August 1986, pp. 36-38

This paper proposes a novel idea.

'We therefore propose to view the process of software construction as a network in which each node represents the product in a certain state and each edge is an action (transition) to transform one state into another. Alternative actions are mode/led by multiple edges originating from the same node. Whenever a state is inconsistent [with objectives] a backtracking takes place which leads to the previous state where alternative paths are possible, which have not been tried. As the information on alternatives is part of a node's properties, the node cannot be disconnected from any previous node and the full development history remains stable and consistent.' (p. 37)

15.1.14 Krzanik

Source: Lech Krzanik, 'Dynamic Optimization of Delivery Step Structure in Evolutionary Project Delivery Planning', Proc. Cybernetics in Organization and Management, 7th European Meeting, Vienna 24-27 April 1984, R. Trappl 'ed.), North-Holland, 1984

Dr Krzanik has since 1980 worked on the automation of these methods on personal computers. The objective of that research effort is to see how far the software engineering design process can be automated. The implementation of the tool, the 'Aspect Engine,' operates on the Macintosh and is shared with suitable research colleagues. The author's conclusion includes:

'An approach to delivery step structure optimization in evolutionary project delivery has been presented. A model and two simple and easy-to-use optimal algorithms MI and VMI for controlling the contents of the project transient set have been given. Elsewhere ('On-line tuning of the smallest useful deliverable policy in evolutionary delivery planning,' 1983) we have given alternative methods for simultaneous optimization of delivery schedule, step range and structure.'

For the management reader, this means that one day you may be offered personal computer tools for dealing with evolutionary planning. For the academic reader, it implies that there is a fairly unexplored mathematical area out there and that evolutionary delivery is capable of formal treatment.

Kai Thomas Gilb has developed an Excel tool for applying the ideas in this book. (Contact author's address or by email to KaiGilb@eworld.com.)

15.1.15 Lehman and Belady

Source: M.M. Lehman and L.A. Belady, Program Evolution: Processes of
Software Change, Academic Press, 1985; originally published ill Journal of Systems and Software, Vol. 1, No. 3, 1980 'Qc 1980 Elsevier Science Publishing Co, Inc.)

This text and the research of the authors cannot be ignored in any overview of software engineering evolution. In one sense it is outside of the scope of our text because it takes an anthropological study view of program evolution, while this book's main subject matter is in management of the development process. The exploitation of specific evolutionary delivery mechanisms in order to achieve specific management targets is our subject. However, the reader is bound to find much of the material rich in ideas and insights. The authors primarily depart from their own well-known studies of the evolution of the IBM 360 Operating System (1969, IBM Research Report RC 2722, The Programming Process, M.M. Lehman).
Since this book is fond of trying to state principles, it is fitting that we introduce this work to the reader by citing some they have derived from their studies. These were apparently first formulated in 1974.

Continuing change

'A program that is used and that, as an implementation of its specification, reflects some other reality, undergoes continuing change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the program with a recreated version.' (p. 381)

This can be compared with Gilb's Fourth 'Law':

'A system tends to grow in steps of complexity rather than of simplification; this continues until the resulting unreliability becomes intolerable.'

This Law was first published in Gilb, Reliable Data Systems, 1971, Universitetsforlaget, Oslo, in Datamation, March 1975, and in Gilb, Reliable EDP Application Design, 1973, Petrocelli,O.A.P. It was later used in Gilb and Weinberg, Humanized Input, 1984, QED Inc., Waltham, Mass.(Currently published by Little Brown).

Increasing complexity

'As an evolving program is continuously changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain it or reduce it.' (p. 381)

The fundamental law (of program evolution)

'Program evolution is subject to a dynamics which makes the programming process, and hence measures of global project and system attributes, self-regulating with statistically determinable trends and invariances.' (p. 381)

Conservation of organization stability (invariant work rated)

'The global activity rate in a project supporting an evolving program is statistically invariant.' (p. 381)

Conservation of familiarity (perceived complexity)

'The release content (changes, additions, deletions) of successive releases of an evolving program is statistically invariant.' (p. 381)

The authors provide comment and data to support their Laws.

A source of Lehman's work more in line with our interest in the development process itself will be found in ACM Software Engineering Notes, Aug. 1986, 'Approach to a Disciplined Development Process - the lSTAR Integrated Project Support Environment,' pp. 2~3. A co-operative project with British Telecom, it stresses a 'contractural model of system development.'

15.1.16 Melichar

Source: Paul R. Melichar, IBM Information Systems Management Institute, Chicago, 'Management Strategies for High-risk Projects', Class Handout,
approx. 1983

Melichar identifies three project strategies, monolithic, incremental, and evolutionary, which are 'different in their ability to cope with risks that undermine manageability, because they reflect different attitudes towards: productivity. . . responsiveness . . . adaptability and .
control.'

'Projects get into trouble precisely because managers treat them as if they were all alike, disregarding three vital factors that impact manageability: duration . . . expectations and . . . volatility.'

Using an IBM study his organization carried out, Melichar goes into depth on optimum project length before delivering meaningful results to the user.

'This testimony strongly suggests that there is a narrow six to twelve month "time window" for optimum manageability. A good rule of the thumb is nine months.'

His distinction between monolithic, incremental and evolutionary system development strategies is argued with case studies and comparative tables, in favor of the latter two options. His incremental strategy is what we have defined as an evolutionary delivery strategy. What he calls evolutionary is what most people would call 'usable prototypes, made by the users themselves', as opposed to professional developers. I would personally not make the distinction, since both options are valid strategies under the evolutionary umbrella. Indeed there is nothing to inhibit us from mixing such strategies within a project. Terminology, is a minor issue. He is bringing the nonmonolithic development options to the attention of his students in a lively and deeply analytical manner.

15.1.17 Parnas

Source: David L. Parnas, 'Designing Software for Ease of Extension and Contraction', IEEE Trans. on Software Engineering, Vol. SE-5, No. 2, March 1979. (Qc 1979 IEEE)

'Software engineers have not been trained to design for change. (p. 129)

'In my experience identification of the potentially desirable subsets is a demanding intellectual exercise in which one first searches for the minimal subset that might conceivably perform a useful service and then searches for a set of minimal increments to the system. Each increment is small - sometimes so small that it seems trivial. The emphasis on minimality stems from our desire to avoid components that perform more than one function. Identifying the minimal subset is difficult because the minimal system is not usually one that anyone would ask for. If we are going to build the software family, the minimal subset is useful; it is not usually worth building by itself. Similarly the maximum flexibility ("easily changed") is obtained by looking for the smallest possible increments in capability . . .' (p. 130)

'There is no reason to accomplish the transformation . . . (to) all of the desired features in a single leap. Instead we will use the machine at hand to implement a few new instructions. At each step we take advantage of the newly introduced features. Such a step-by-step approach turns a large problem into a set of small ones and . . . eases the problem of finding the appropriate subsets. Each element in this series . . . is a useful subset of the system. (p. 131)

'Subsetability is needed, not just to meet a variety of customers' needs, but to provide a fail-safe way of handling schedule slippage.' (p. 136)

Parnas has also said in a private communication:

'There are lots of people preaching evolutionary delivery. For a few of those whose content is more than mere exhortation, see Habermann (Modularization and Hierarchy, CACM, Vol. 5, 1976), Liskov (The Design of the Venus Operating System, CACM, July 1975), Dijkstra (The Structure of T.H.E. -multiprogramming system, CACM, May 1968, and CACM, August 1975), Per Brinch-Hansen (The Nucleus of a Multiprogramming System, CACM, April 1970),
P.A. Janson (Using Type Extension to Organize Virtual Memory, MITLTS-TR167, September 1976).'

15.1.18 Quinnan

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77

Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability.' When a satisfactory design at cost target is achieved for a single increment, the 'development of each increment can proceed concurrently with the program design of the others.'

'Design is an iterative process in which each design level is a refinement of the previous level.' (p. 474)

It is clear from this that they avoid the big bang cost estimation approach. Not only do they iterate in seeking the appropriate balance between cost and design for a single increment, but they iterate through a series of increments, thus reducing the complexity of the task, and increasing the probability of learning from experience, won as each increment develops, and as the true cost of the increment becomes a fact.

'When the development and test of an increment are complete, an estimate to complete the remaining increments is computed.' (p. 474)

This article is far richer than our few selected quotations can show, with regard to concepts of cost estimation and control.

15.1.19 Radice

Source: Ron A. Radice et al., A Programming Process Architecture, IBM Systems Journal, Vol. 24, No. 2,1985, pp. 79-90

Radice and his team developed a model of software engineering management which has been voluntarily adopted as a basis by many IBM development laboratories. this model is the basis for the 'Capability Maturity Model' (CMM) taught by Software Engineering

Institute, Carnegie Mellon University (See IEEE Software, July 1993).It is partly based on the best practices of several laboratories in the past. The central idea of the method is that IBM should not establish the particular programming languages and software tools to be used corporate-wide at all. They should rather give the laboratories a framework for making their own decisions on the particular tools to be applied to particular product developments at particular times.

The idea is that software engineering should be based on a 'process control idea.' Subsidiary support ideas are that Fagan's inspection method should be used to collect basic data about the development process. In addition, the driving force should be measurable multidimensional objectives (using Gilb's goal quantification method).

'An underlying theme of the architecture process is a focus on process control through process management activities. Each stage of the process includes explicit process management activities that emphasize product and process data capture, analysis and feedback.' (p. 83)

'Indeed, to achieve consistently improving quality, the management practices of goal setting, measurement, evaluation, and feedback are an absolutely essential part of the process.' (p.82)

The actual selection of particular software development languages and tools is thus evolutionary. IBM is using a very conscious application of evolutionary delivery to deliver improvement to their individual laboratories' development process.

Some further quotations from that article follow:

'Just as timely data are needed to manage the quality of the developing product, historical data are required to evaluate and correct weaknesses in the process over a succession of projects.' (p. 88)

'The (IBM) Process Architecture emphasizes quality over productivity, with the understanding that as quality improves, productivity will follow.' (p. 88)

'Early quality goal setting and evaluations can lead to an earlier focus on areas of initial high difficulty. As a result, better initial allocation of key personnel and other resources can follow.' (p. 88)

It is my personal opinion that the work of the IBM team is a very important set of ideas for other people trying to organize their software engineering process for the long term. Earlier efforts in our field concentrated on the 'product development itself, or upon the tools for making that product. Radice and his team gave us a framework for making those more short term decisions, based on a rich process control architecture for the entire development process. The paper is so rich in ideas that the serious reader should read the complete paper.

15.1.20 Robertson and Secor

Source: Leonard B. Robertson and Glenn A. Secor, AT&T, 'Effective Management of Software Development', AT&T Technical Journal, March/ April 1986, Vol. 65, Issue 2, pp. 9~01. 'QC1986 AT&T)

'Large projects usually have more success by spreading releases over time. Development strategy addresses the same issue internally: one delivery to the test organization or several incremental deliveries. Projects in which the interval from design through unit test is longer than four to six weeks should use incremental development.' (p. 96)

'In addition, quality goals and quality improvement goals should be stated.' (p. 96)

'Testing should start during the requirements phase and should use an independent system test group, test inspections, and frequent demonstrations.' (p. 97)

'To provide for the unexpected, the development plan should include a contingency plan, which may involve having increments only partially full, or an extra increment following a risky increment.' (p. 96)

'At the end of each project review meeting, supervision should see a demonstration of completed increments. Demonstrations, more than any other approach, make mileposts visible.' (p. 100)

15.1.21 Rzevski

Source: Leonard B. Robertson and Glenn A. Secor, AT&T, 'Effective Management of Software Development', AT&T Technical Journal, March/April 1986, Vol. 65, Issue 2, pp. 9~1 01. 'QC1 986 AT&T)

The evolutionary design methodology

'The evolutionary design methodology (EDM) is a body of knowledge aimed to help designers to:

1. identify and formulate design problems,
2. establish design goals,
3. understand the design process,
4. select and apply methods for design and management.

The word "evolutionary" in the title indicates that EDM gives prominence to design methods that allow systems to grow in an incremental fashion and thus enable both user and designers to learn as they take part in the design process. It also indicates that EDM evolves and changes with time.'

Rzevski's detailed picture of the EDM method, which he uses primarily as a teaching vehicle, not as a publicly marketed methodology, emerges as essentially similar in objectives and nature - though not exact detail - to the methods in this book (which I collectively call design by objectives [DBO]). He simply chooses to view the set of sub-methods he teaches from the evolutionary point of view, while I prefer to think of my methods primarily in terms of the design objectives to be attained, and evolutionary delivery is but one tool for reaching those objectives.

'There are two major objectives of EDM; firstly to increase productivity of the design process, and secondly to achieve the desired quality of the design product.'

In his detailed treatment of quality it is clear that Rzevski has a very broad multidimensional and quantitative view of quality -including for example 'social acceptability.'

'EDM can cope with a variety of types of design problems including those characterized by fuzziness and complexity.'

This specific willingness to deal with fuzziness is a clear sign that Rzevski is of the real world. Indeed he is also an active industrial consultant. He is closer in his thinking to my ideas than perhaps any other author cited here.

'Systems whose requirements are rather complex or fuzzy should not be designed and implemented in one step. It is wiser to allow them to evolve and thus enable both users and designers to learn as design progresses.

This gives explicit recognition of the necessary learning process.

'It is advisable to produce solutions that are easy to modify or replace.

I take this as recognition of the necessity for the open-ended design solutions discussed in this book.

Additional Source: G. Rzevski, 'Prototypes versus Pilot Systems: Strategies for Evolutionary Information System Development', in

Approaches to Prototyping, Budde et al. (eds.), Springer-Verlag, 1984

Rzevski on Popper and evolutionary knowledge growth

'According to Popper (K.R. Popper, Objective Knowledge, an Evolutionary Approach, Oxford University Press, 1972) human knowledge grows by means of never-ending evolution. The vehicle for this growth is the process of problem solving: we create theories (i. e. knowledge) in order to solve problems; however, every solution to a problem creates new problems which arise from our own creative activity . . . they emerge autonomously from the field of new relationships which we cannot help bringing into existence with every action, however little we intend to do so.

The inevitable growth of knowledge which takes place during systems development should not be suppressed by imposing linear life-cycle discipline upon the development process. On the contrary, every effort should be made to take advantage of the human propensity to learn. .

Kuhn's paradigm theory

T.S. Kuhn (The Structure of Scientific Revolutions, University of Chicago Press, 1970) has a theory of revolutionary growth of knowledge -which needs to be balanced against Popper's ideas. It can be summarized as follows:

'Knowledge grows through the work of scientists who organize themselves into different disciplines . . . solving problems within the framework of a dominant paradigm. . . . Over a period of time problems emerge which cannot be solved within the established paradigm . . . new paradigms are proposed . . . one . . . emerges as the main challenge to the established order . . . transfer to a new paradigm occurs . . . only after considerable resistance . from. . . established . . . members . . . who do not accept that there is a need for change. . . . Scientific argument and feuds are typical for those periods preceding the revolutionary change of the dominant scientific world view. . . . The evolutionary approach. . . offers. . . a new paradigm. . .

15.1.22 Sachs

Source: Susan Lammers, Programmers at Work, Microsoft Press (USA and Canada), Penguin Books elsewhere, 1986 (Qc 1986 by Microsoft Press. All rights reserved)

Jonathan Sachs wrote the best-selling Lotus 1-2-3 software. In his interview in Programmers at Work, he cites several evolutionary viewpoints:

'The spreadsheet was already done, and within a month I had converted it over to C. Then it started evolving from that point on, a little at a time. In fact, the original idea was very different from what ended up as the final version of 1-2-3.' (p. 166)

'The methodology we used to develop 1-2-3 began with a working program, and it continued to be a working program throughout its development. I had an office in Hopkinton where I lived at the time, and I came to the office about once a week and brought in a new version. I fixed any bugs immediately in the next version. Also, people at Lotus were using the program continuously. This was the exact opposite of the standard method for developing a big program, where you spend a lot of time and work up a functional spec., do a modular decomposition, give each piece to a bunch of people, and integrate the pieces when they're all done. The problem with that method is that you don't get a working program until the very end. If you know exactly what you want to do, that method is fine. But when you're doing something new, all kinds of problems crop up that you just don't anticipate. In any case our method meant that once we had reached a certain point in development, we could ship if we wanted to. The program may not have had all the features, but we knew it would work.' (p. 167).

Sachs then goes on to remark that this method 'doesn't work very well' with more than one to three people! A conclusion that must be based on the wrong experiences or none at all, as the documented large-scale cases in this book evidence.
Sachs continues:

'Success comes from doing the same thing over and over again; each time you learn a little bit and you do it a little better the next time.' (p. 170)

Sachs even touches on open-endedness when asked to describe his basic approach to programming.

'First, I start out with a basic program framework, which I keep adding to. Also I try not to use many fancy features in a language or a program. . . . As a rule I like to keep programs simple.'

15.1.23 Shneiderman

Source: Ben Shneiderman, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley, 1987

'Designs must be validated through pilot and acceptance tests that can also provide a finer understanding of user skills and capabilities.' (p. 390)

Iterative design during development

Design is inherently creative and unpredictable. Interactive system designers must blend a thorough knowledge of technical feasibility with a mystical esthetic sense of what will be attractive to users. Carroll and Rosson ('Usability specifications as a tool in iterative development', in H. Rex (ed.), Advances in Human-Computer Interaction 1, Ablex Publishing, Norwood NJ, 1985) characterize design this way:

- 'Design is a process: it is not a state and cannot adequately be represented statically.
- The design process is non-hierarchical; it is neither strictly bottom-up nor strictly top-down.
- The process is radically transformational; it involves the development of partial and interim solutions that may ultimately play no role in the final design.
- Design intrinsically involves the discovery of new goals.

These characterizations of design convey the dynamic nature of the process.' (p. 391)

,15.2 Management sources

15.2.1 Garfield

Source: Charles Garfield, Peak Performers, William Morrow & Co., Inc., NY, 1986

'Many of the major changes in history have come about through successive small innovations, most of them anonymous. Our dramatic sense (or superficiality) leads us to seek out "the man who started it all" and to heap upon his shoulders the whole credit for a prolonged, diffuse and infinitely complex process. It is essential that we outgrow this immature conception. Some of our most difficult problems today . . . defy correction by any single dramatic solution. They will yield, if at all, only to a whole series of innovations.'
(Quoting John Gardner, founder of 'Common Cause' p. 128

'Again and again, we see results emerging from the many jobs that take meaning from - and give form to - a few strategies. Lawrence Gilson, a former vice-president of Amtrak, is one of a group that worked to build a high-speed "bullet train" railroad in the United States. The odds, as it turned out, proved too great even for peak performers. But it was a near thing: the Japanese government cooperated; Wall Street gave it a serious look; builders invested $1 million of their own money. Investors were not putting their money into a fuzzy R&D project. Gilson knew that "you have to know what the three or four steps out in front of you are. You have to set milestones that are achievable. You can't expect someone to come in on the basis of being sold the big picture. You have to sell each incremental step. What you bring to them at each phase is not just conceptual, it is work completed."

Visionaries who were less than peak performers in handling incremental steps might have failed to get the project out of the dream stage, or might have deluded themselves that they could continue when the fact was they could not. Gilson and his partners raised $10 million toward the $3.1 billion project. They knew they would need another $50 million in risk capital to keep operating until the planned beginning of construction in 1985. They had done their detail work. When they saw that the $50 million was not going to come in by the time they had to have it, they knew it was time to quit, and sold their engineering plans to Amtrak. The peak performer"'. perspective not only lets you know when to continue. It also lets you know when to stop.' (p. 129)

'Through repeated educated risks, the peak performers learn as they go along,. and over time their confidence in their own judgment gains strength. It is not fear of failure that drives them along, but a strong desire for achievement.

Remember Warren Bennis's finding that the ninety leaders he interviewed would use almost any word - "glitch", "false start", "bug" - rather than "failure". The reason goes beyond semantics. It has to do with learning. When high achievers get less than the results they plan for and work toward, they allow the normal human feelings of disappointment, or anger, or fatigue, to pass; then they start analyzing. They search for information in the situation: Where are we now? Where are we headed? How do we get there? They operate as both innovator and consolidator, and resume moving towards completion of their mission and goals.

Even when circumstances are totally beyond their control, peak performers learn what they can from an experience so as not to knock their heads against the wall again. They keep their eyes open so that they do not, as Joseph Campell once put it, "get to the top of the ladder and find it's the wrong wall."' (p. 138)

This activity is clearly identical to the evolutionary delivery pattern of working towards well defined objectives.

15.2.2 Grove

Source: Andrew S. Grove, Intel Chairman and Founder, High Output Management, Souvenir Press (UK), Random House (USA), 1983

'How far ahead should the planners look? At Intel, we put ourselves through an annual long-range planning effort in which we examine our future five years off. But what is really being influenced here? It is the next year - and only the next year. We will have another chance to replan the second of the five years in the next year's long-range planning meeting, when that year will become the first year of the five.

So, keep in mind that you implement only a portion of a plan that lies within the time window between now and the next time you go through the exercise. Everything else you can look at again.

We should also be careful not to plan too frequently, allowing ourselves time to judge the impact of the decisions we made and to determine whether our decisions were on the right track or not. In other words, we need the feedback that will be indispensable to our planning the next time around.'

This statement is similar to the evolutionary delivery philosophy of keeping the steps beyond the next one as fluid planning elements, to be finally decided on in the light of real experience.

15.2.3 Moss Kanter and Quinn

Source: Rosabeth Moss Kanter, The Changemasters, copyright Qc 1983. Reprinted by permission of Simon & Schuster, Inc.

'The most saleable projects are likely to be trial-able (can be demonstrated in a pilot basis); reversible (allowing the organization to go back to pre-project status if it doesn't work); divisible (can be done in steps or phases); consistent with sunk costs (builds on prior resource commitments); concrete (tangible, discrete); familiar (consistent with a successful past experience); congruent (fits the organization's direction); and with publicity value (visibility potential if it works).' (p. 221)

This is a fairly complete description of the main parameters of the evolutionary delivery process.

'"Too much talk, too little action" is a common complaint about participative vehicles that do not have concrete tasks to carry out. For this reason, a Hewlett-Packard facility uses its MBO (management by objectives) process to prioritize a team's activities; they are encouraged to work on a succession of easy problems before tackling tough ones.' (p. 254)

This philosophy is consistent with the evolutionary delivery rule of prioritizing the high value and low development cost steps first.

'"Breakthrough" changes that help a company attain a higher level of performance are likely to reflect the interplay of a number of smaller changes that together provide the building blocks for the new construction. Even when attributed to a single dramatic event or a single dramatic decision, major changes in large organizations are more likely to represent the accumulation of accomplishments and tendencies built up slowly over time and implemented cautiously. "Logical incrementalism," to use Quinn's term, may be a better term for describing the way major corporations change their strategy:

The most effective strategies of major enterprises tend to emerge step-by-step from an iterative process in which the organization probes the future, experiments, and learns from a series of partial (incremental) commitments rather than through global formulations of total strategies. Good managers are aware of this process, and they consciously intervene in it. They use it to improve the information available for decisions and to build the psychological identification essential to successful strategies. . Such logical incrementalism is not "muddling" as most people understand that word . ... [It] honors and utilizes the global analyses inherent in formal strategy formulation models [and] embraces the central tenets of the political power-behavioural approaches to such decision-making.' (pp. 289-90 quoted from James Brian Quinn, Strategies for Change: Logical Incrementalism, Homewood, Illinois: Richard D. Irwin, 1980)

15.2.4 Peters and Austin
Source: Tom Peters and Nancy Austin, A Passion for Excellence, Collins
(UK), Random House (USA), 1985 (QC 1985 Thomas J. Peters and Nancy K. Austin)

'It is precisely when the buyer has become less dependent on the technical help or brand support of the originating buyer, that greater attention may be beneficially focussed on a systematic program of finding customer-benefiting and therefore customer-keeping augmentation.' (pp. 69-70)

This point simply reminds us of the evolutionary nature of all product development which needs to compete for customers.

'And yet we go wrong time and again because we do rely on numbers and transparencies alone, and lose our "feel". The only way to enhance feel is to be there.' (p. 94)

This point is central to evolutionary delivery which is among many things a way to regain realistic touch with a complex software development, and to avoid relying too much on paper specifications for understanding and control.

'The course of innovation - from the generation of the idea through prototype development and contact with the initial user to breakthrough and then to final market - is highly uncertain. Moreover it is always messy, unpredictable and very much affected by the determined ("irrational"?) champions, and that is the important point. It's important, because we must learn to design organizations - those that are public as well as private, banks as well as software developers - that take into account, explicitly, the irreducible sloppiness of the process and take advantage of it, rather than systems and organizations that attempt to fight it. Unfortunately, most innovation management seems to be predicated on the implicit assumption that we can beat the sloppiness out of the process if only we can make the plans tidier and the teams better organized. . . in that single phrase "Let's get organized for the next round" lie the seeds of subsequent disaster.' (pp.115-6)

Evolutionary delivery is a specific example of a process for coping with the inherent messiness of user requirements and our poor understanding of new untried technology.

'Myth: Complete technical specs. and a thoroughly researched market plan are invariant first steps to success.
Counterpoint: You must move as rapidly as possible to real tests of real products (albeit incomplete) with real customers. That is, you must experiment and learn your way toward perfection/ completion.

Myth: Time for reflection and thought built into the development process are essential to creative results.

Counterpoint: "Winners"- e.g.successful champions/skunks - are above all, pragmatic non-blue sky dreamers who live by one dictum: "Try it, now!"

Related Myth: Big projects are inherently different from small projects - or, an airplane is not a calculator.
Counterpoint: Some projects are indeed much bigger than others. Yet the most successful big-project management comes from small within big mindset, in which purposeful "suboptimization" is encouraged.'

The above comments are directly aimed at the heart of the debate between waterfall model planning and evolutionary delivery.

'Develop a prototype, or a big hunk of it in 60 to 90 days. Whether your product or service is a digital switch, a new aircraft or a computer - or a new health service or financial instrument or a store format - our evidence suggests that something can always be whacked together in that time.

Then evaluate the prototype: that takes another 60 days . You're already playing with something tangible, or, say, a large hunk of primitive software code. Now you take the next little step. Maybe it costs a little more, for a more fully developed prototype . . . But again you build it fast . . . And this time you can probably get it, or part of it, onto the premises of a user (customer) - not an average user (that is a bit away, but a "lead user" who's willing to experiment with you, or at least an in-house lead user (a forward thinking department). And the process goes: slightly larger investments, timeframes that never run more than 60 to 90 days. It's the "learning organization" or the "experimenting organization."

At each step you learn a little more, but you have harsh reality tests - with hard product/service and live users/customers - very early. If it doesn't work you weed it out quickly, before you have career lock-in and irreversible psychological addiction to the "one best design." (This approach) can cut the time it takes to complete the development cycle by 50% or more.' (pp. 129-130)

This quotation is an excellent explanation of the reasoning behind evolutionary software delivery methods. Needless to say the entire
book is rich with practical examples and detail to support this theory.

'Multiple passes usually take much less time, and result ultimately in the development of simpler (more reliable), more practical (if less "beautiful") systems than the single "Get it exactly right the first time" blitz.' (p. 150)

This comment applies directly to the big-bang theory of software development compared to the 'multiple pass' evolutionary development model. Maybe the interfacing isn't beautiful, but it is more practical.

'A "learning system" is vital. . . . And make sure the learning "system" or process encompasses (and generates) many small wins. Get people to make daily assessments; then act on those assessments. (Incidentally the small-win quick-feedback process actually generates practicality.' (p. 298)

Evolutionary delivery is a learning process with many small wins on the way which generates practical action.

'For heaven's sake, go after the easy stuff first! What's the thrill of beating your head against a brick wall?' (p. 301)

This is one of our evolutionary delivery methods central principles: the highest user-value to development-cost steps ('easy stuff') shall be identified and done first. I have never been able to understand why some software people plan as though they enjoy waiting years to see any results handed to their users and customers. My theory is that the problem is caused by the fact that they get paid monthly regardless.

'With respect to individuals, psychology (theory) focuses on the overriding importance of commitment, if motivation is to be sustained, and of the quick feedback associated with human-scale, tangible achievements. The literature on resistance to change (in both individuals and groups) suggests that the best way to overcome it is taking tiny steps, and, moreover working on the positive ("we can do something right"), rather than trying to confront negative feelings directly. . . . The small win is exactly about the creation of plausible, positive role models.' (p. 304)

15.2.5 Peters and Waterman

Source: Peters and Waterman, In Search of Excellence, Harper and Row, New York, 1982

'The essence of excellence is the thousand concrete minute-to-minute actions performed by everyone in an organization to keep a company on its course.

'P&G (Procter and Gamble) is apparently not afraid of testing and therefore telegraphing its move. Why? Because, we suspect, the value added from learning before the nationwide launch so far exceeds the costs of lost surprise.' (p. 136)

TI's (Texas Instruments) ability to learn quickly, to get something (almost anything) out in the field. They surprised themselves: as a very small company, $20 million, with very limited resources, they found they could outmaneuver large laboratories like Bell Labs; RCA and GE in the semiconductor area, because they'd just go out and try to do something, rather than keep it in the lab.' (Charles Phipps, of TI) (p. 136)

'At Activision the watchword for video-game design is "build a game as quickly as you can." Get something to play with. Get your peers fooling with it right away. Good ideas don't count around here. We've got to do something.' (p. 136)

'At HP (Hewlett-Packard), it's a tradition that product-design engineers leave whatever they are working on out on top of their desk so that anyone can play with it. . . . You are told probably on the first day that the fellow walking around playing with your gadget is likely to be a corporate executive, maybe even Hewlett or Packard.' (p. 137)

¯15.3 Engineering sources

15.3.1 Deming

Source: W. Edwards Deming, Out of the Crisis, MIT CAES, 1986 and Cambridge University Press

Deming cites the 'Shewhart Cycle', known in Japan as the Deming Cycle. It is an example of an evolutionary product development method under competitive conditions.

'At every stage there will be. . . continual improvement of methods and procedures aimed at better satisfaction of the customer (user) at the next stage. Each stage works with the next stage and with the preceding stage toward optimum accommodation, all stages working together toward quality that the ultimate customer will boast about.' (p. 87)

In addition to this direct mention of the cycle, it is worth noting that the statistical quality control charts, which are the primary tool of Dr. Deming, are one way of viewing the evolutionary progress results. They can also be viewed by readers of this book as another kind of measurement process for critical system attributes. Indeed, Deming is cited by Michael E. Fagan, as one of his sources on quality control ideas which led him to develop software inspections.

15.3.2 Koen

Source: Billy V. Koen, 'Toward a Definition of the Engineering Method', in Spring 1985 THE BENT of Beta Pi, 0C1EEE reprinted there with permission from Proceedings; Frontiers in Education, 14th Annual Conference, Philadelphia, PA, 3-5 October 1984. It also appeared in Engineering Education, December 1984.

Koen is at University of Texas, Austin, E.T.C. Building, Room 5.134B, Austin TX, USA 78712. He solicits engineering heuristics. Koen is a professor of mechanical engineering. In his article he defines as one of several heuristics, the Engineering Method:

'The Engineering Method is the use of heuristics to cause the best change in a poorly understood situation within the available resources.

He defines heuristics as hints or rules of thumb in seeking a solution to a problem. The principles in this book are 'heuristics' in this sense. Professor Koen manages to comment on several points central to this book, including evolutionary delivery.

'Engineering is a risk-taking activity. To control these risks, engineers have many heuristics.

1. They make only small changes in what has worked in the past, but they also
2. try to arrange matters so that if they are wrong they can retreat, and
3. they feed back past results in order to improve future performance.

Any description of engineering that does not acknowledge the importance of these three heuristics and others like them in stabilizing engineering design and, in effect, making engineering possible, is hopelessly inadequate as a definition of engineering method.'

Later in discussing the structure of engineering methods he says:

engineers cannot simply work their way down a list of steps, but. . . they must circulate freely within the proposed plan - iterating, backtracking, and skipping stages almost at random. Soon structure degenerates into a set of heuristics badly in need of other heuristics to tell what to do when.

15.3.3 Shewhart

In W.E. Deming, 'Tribute to Walter A. Shewhart', Industrial Quality Control, Vol. 24, No. 2, August 1967, cited in AT&T Technical Journal March/April 1986, pp. 11-12, Deming emphasizes the grand old man of industrial quality control had a very wide view of the process:

'Quality control meant to him use of statistical methods all the way from raw material to consumer and back again, through redesign of product, reworking of specifications, in a continuous cycle, as results come in from consumer research and from other tests.'

In the version of evolutionary delivery recommended in this book, this is exactly the view. The evolutionary cycle must encompass all elements of system design and construction. It must deliver results to consumers. And, it must learn both from data collected from real consumer> and other tests.

15.4 Architectural sources

15.4.1 Alexander

Source: Christopher Alexander, Murray Silverstein, Sara Ishikawa et al., The Oregon Experiment, copyright 0c 1975 by The Center for Environmental Structure. Reprinted by permission of Oxford University Press, Inc.

Alexander and his group developed and practiced a number of relevant and interesting ideas within architecture. They practiced them in connection with the long-term architectural planning at the University of Oregon, thus the name of the book.

The major idea is that long-term developments should not be constrained by a static master plan. They should be allowed to grow incrementally, by user-participation, within certain overall guiding principles called 'patterns.'

There are six fundamental principles of implementation;

'1. The principle of organic order. Planning and construction will be guided by a process which allows the whole to emerge gradually from local acts.
2. The principle of participation. All decisions about what to build, and how to build it, will be in the hands of the users.
3. The principle of piecemeal growth. The construction undertaken in each budgetary period will be weighed overwhelmingly towards small projects.
4. The principle of patterns. All design and construction will be guided by a collection of communally adopted planning principles called patterns.
5. The principle of diagnosis. The well being of the whole will be protected by an annual diagnosis which explains, in detail, which spaces are alive and which ones dead, at any given moment in the history of the community.
6. The principle of coordination. Finally, the slow emergence of organic order in the whole will be assured by a funding process which regulates the stream of individual projects put forward by the users.' (pp. 5-6)

Each principle is further exploded into more detailed principles and explained and illustrated in the book. For example:

'The principle of participation:
All decisions about what to build, and how to build it, will be in the hands of the users.
To this end:

- there shall be a users' design team for every proposed building project;
- any group of users may initiate a project, and only those projects initiated by users shall be considered for funding;
- the planning staff shall give the members of the design team whatever patterns, diagnosis and additional help they need for their design;
- the time that users need to do a project, shall be treated as a legitimate and essential part of their activities;
- the design team shall complete their schematic designs before any architect or builder begins to play a major role.' (p. 58)

The 'patterns' can be compared with our notion in this book of open-ended solutions.' They are design rules which do not merely limit themselves to ensuring ease of growth and change, but they ensure all manner of other objectives such as human convenience and economics.

The ideas here are quite exciting and revolutionary - we must wonder who will be the people to document that they have applied such rules in software development?

15.4.2 Frank Lloyd Wright

Source: Frank Lloyd Wright, An Autobiography, Horizon Press, NY, 1977

Admitting to being an evolutionist

'The revolutionary evolutionist is never exactly penitent.'
(p. 447) 'Mastery is no mystery. Simple principles of nature apply with
particular emphasis and force to all a true master does:
.

Planned progressions, thematic evolution, the never-ending variety in differentiation of pattern, integral ornament always belonging naturally enough to the simplest statement of the prime idea upon which structure is based: Beethoven's rhythms are like that - integral like those of nature!

And likewise the work of the inspired Architect.' (p. 454)

Another Source: Patrick /. Meehan (ed.), The Master Architect - Conversations with Frank Lloyd Wright, John Wiley & Sons, Inc., 1984

On organic architecture versus military architecture

'Well, call organic architecture a natural architecture. It means building for and with the individual as distinguished from the pseudo-classical order of the American schools today, mainly derived from the survivals of ancient military and monarchic orders.' (p. 122)

Evolutionary delivery could well also be called something like organic systems development. It is distinguished by 'building for and with the user' as opposed to building for the technologist.

but "organic architecture" which is the architecture of nature, the architecture based on principle and not upon precedent. Precedent is all very well so long as precedent is very well but who knows when it is very bad? Now that's something to guard against in architecture - know when to leave your precedent and establish one.' (p. 80)

The major philosophy of this book is that design is based on principle, not precedent. We must not spend so much energy looking for the right languages, structures, and tools, as we should spend in finding the principles by which we can select our technologies for the task at hand and the environment of the present day.

'Organic architecture comes of nature.' (p. 112)

I find all too often that software people are too ready to throw away existing systems entirely, and replace them with a totally new design. In doing this they ignore the fact that the existing system is thrown away without adequate replacement. This includes not only code but also traditions of work, methods that have been learned, patterns that are now easily recognized (screens, forms and codes), are also - often inadvertently - the very glue and oils that make the system work well in the real world. We throw the baby out with the bath water.

Software people need to have much greater respect for existing natural' ideas and systems. They are too cock-sure that their traditions and methods are the right ones. This is explored in depth in our book Humanized Input (Gilb and Weinberg, 1984, QED Publishers). For example, computer people are so sure that a ten digit number is the only right way to refer uniquely to a customer or product, when the age old tradition of names and varying alphabetic names is usually clearly superior from a human point of view according to Bell Labs research (cited in Humanized Input), because it is easier to access, to remember and contains useful redundancy from which a computer can spot errors and automatically correct them.

'Organic architecture is the architecture from the inside out.' (p. 201)

Evolutionary systems (a synonym for organic architecture) evolve from an inner essential core of the system and they add layers of function and qualities - from the inside and outwards.

'Lao-Tse, of course, was the man which proclaimed modern organic architecture and that was 500 years before Jesus.' (p. 218)

The main point here being that we are not speaking of a new idea, but one which the ancients recognized. Indeed, how could they not observe nature?

On flexibility design for robustness

(The Imperial Hotel in Tokyo survived the great earthquake there.) 'And it was built to do it. It was thought-built to stand against an earthquake. That was the thought from beginning to end, and when the earthquake came it got up against that thought, and sneaked off.

Falkenburg (interviewer) "Was it the only building in Tokyo to stay standing?"

Wright: "The only building, practically, that's ever been built on the principle of flexibility. . . . It is welded together on the principle of flexibility, and that was new in the building world.' (p. 280)

This is one famous example of Wright's design engineering. He carefully noted the earthquake-prone environment. He designed a building to be robust and withstand the disaster. He used a design principle of unified flexibility of parts of the system. He spoke here about 'the new architecture . . . is organic . . . making it all as one.

On the definition of architecture

'I think architecture is the science of structure and the structure of whatever is, whether it is music, whether it is painting or building or city planning or statesmanship.' (p. 131)

Wright admits to a very broad interpretation of architecture. This perhaps admits 'softecture' (software architecture) and 'infotecture' (information systems architecture) as sub-specialities. Certainly one characteristic of information system> is that they must bend to serve their environment and real people, just like buildings. They must be composed of many disciplines of technology - just like building.

On the value to cost relationship

'Student: " . . . aren't most of your buildings relatively expensive as far as the common man is concerned?"

Wright: "I wouldn't say so, although the profession has slapped me with that. I don't think that my buildings, wherever they stand, for the space they enclose and the accommodation you'll find ill them cost as much as those standing around them and I'm prepared to demonstrate.

What I'm anxious to do is the best that can be done no matter what. Now you don't sell houses, you don't sell buildings, you sell your services to help the man get the best thing that can be had according to his idea of the thing -- you're working for him. . Ask them, lots of them will say: 'Well it cost more, Mr. Wright, than we wanted it to cost but we're glad to get it.' None of them are sorry. Now isn't that the real thing.........I don't believe that one building that I've built. . . per square foot. . . costs any more than those standing around it and often times very much less." (pp. 22~1)

Wright is concentrating on providing the best possible value to the customer. But, he takes pride in the fact that he does it in a competitively economical way. Evolutionary delivery has the basic planning principle built into it that we should build in evolutionary steps which create the highest value for the user in relation to the cost as we can, and we should do it as soon as we can in the evolutionary process.

15.4.3 Victor Papanek

Source: Victor Papanek, Design for the Real World, Granada, 1974

Victor Papanek studied with Frank Lloyd Wright. He champions meaningful and socially responsible design. He describes his design method in terms which have elements of the evolutionary delivery method. He describes a series of steps:

- Assembling a design team representing all relevant disciplines, as well as members of the 'client group'.
- Research and fact-finding phase
- Design and development of ideas.
- Checking of these designs against goals established . . , and correcting the designs in the light of these design experiences
- Building of models, prototypes, test models, and working models.
- Testing of these by the relevant user-groups.
- The results of these tests are now fed back into master plans.
- Redesign, retesting and completion of the design job together with whatever documentation is necessary.
- The master plan is to be preserved and used as a follow-up guide in checking actual in-use performance characteristics of the designed objects. It can then also be used as a template for future design jobs that are similar in nature.

'It should be obvious that in reality the design process can never follow a path quite as linear and sequential as suggested by this example. (For one thing new research data emerge continuously.)' (pp. 25~7)

While we are on to Papanek, there are some other quotations which are relevant to this book:

'The wrong kind of problem statement . . . can effectively stop problem solving.' (p. 133)

'The most important ability that a designer can bring to his work is the ability to recognize, isolate, define and solve problems.

(p. 132)
design as a problem-solving activity can never, by definition, yield the one right answer: it will always produce an infinite number of answers, some "righter" and some "wronger". The rightness of any design solution will depend on the meaning with which we invest the arrangement.

Design must be meaningful.'

He then proceeds to examine system function and related attributes of use, method, aesthetics, need, telesis (the deliberate, purposeful utilization of the process of nature and society to obtain particular goals), and association (the psychological conditioning.
which pre-disposes us, or provides us with antipathy against a given value). The main point in this context being that he goes far beyond mere functional (what is the design supposed to do?) thinking into the other attributes of the product.

15.5 Other sources

15.5.1 Davies

Source: A. Morley Davies, Evolution and its modern critics, Thomas Murby & Co., London, 1937. Reproduced by kind permission of Unwin Hyman Ltd

Cuvier's principle of correlation

He translates Cuvier's principle of correlation:

'Every organized being forms a whole, a unique and closed system, of which all parts mutually correspond and cooperate by reciprocal reaction for the same definite end. None of these parts can change without the others changing also; consequently each of them, taken separately, indicates and gives all the others.' (p. 133)

This sounds very much like a software (and hardware and human) system. As we evolve we are forced to consider the effect on all parts of the system. This reminds us also that we need configuration management, so that we can account for all related parts during the evolutionary change process, and so that we can define the exact status of a particular evolutionary step.

The fundamental necessity principle

'The one fundamental necessity of a developing animal is that at every stage of its growth it should be able to live in its particular surroundings.' (p. 138)

Evolutionarily developing systems must also live in a real world of some form of usage, not mere testing of individual modules - in order that the 'test' reflects realistic conditions. We want the data provided to be as near to the truth of the future as possible.

Dohrn's principle of change of function

'Anton Dohrn was the founder of the Naples Zoological station, (who) enunciated the "principle of change of function" (Princip des Funtionwechels) in 1875. The principle is that an organ may have, in addition to its primary function, one or more subsidiary functions, and when changed conditions render the original function unnecessary one of the minor functions may assume primary importance and lead to new developments in the organ. The value of this principle lay in the clearing away of those formidable obstacles to the acceptance of evolution presented by organs or systems of organs which would apparently be quite useless until fully developed.' (p. 149)

This principle reminds us that in an evolving system we must design some technologies which have only a short-term function at early steps; or even no initial use at all. Yet, we are wise if we can find and apply technologies (solutions) which have additional attributes to those initially necessary, and perhaps not useful in the long term. We should want solutions which display versatility in helping us fulfill our objectives even when the exact sequence of content of the evolutionary steps is unknown. I have called these technologies 'open-ended architectures' in this book.

For example, a simple facility for enabling a text comment, which can be inserted in the midst of a programming language can initially serve as a means of documentation of the language usage. But, as Leon Stucki has shown us it can also later be adapted to extend the language system by means of allowing for comments in a formal language which can be interpreted by a computer. For example: 'Comment: all global variables are positive or zero now.' See Figure 13.2c.

Dollo 's law, or the principle of irreversibility

'The past is indestructible.' (Louis Dollo, Belgian paleontologist, 1857-1931.)

'It was never intended as a denial of the possibility of reversing its direction, but of the possibility of such reversal being exact.' (p.164)

We can decide that an evolutionary step is a failure and we can revert to the status immediately previous to that step. But, we cannot eliminate the user and developer experience of the failed step. Indeed, we want to learn from the mistake and change the future for the better.

The user experience however must never be so bitter as to reduce their willingness to use the system we are developing. This means that the developer must exercise caution when designing a step so that it cannot at worst be destructive (for example lead to long down-time or destroyed databases). It means that new steps should be introduced cautiously and spread further only after being proven in a limited environment. It means not merely that a step be small - this is not even in itself important. It does mean that the maximum risk of negative experience at one single delivery step must be kept to a planned maximum, to a level fully acceptable to the users involved.

The principle of vestigal organs

'The existence in many animals of structures to which no use can be assigned, but which are obviously identical with structures that are useful in other animals, has always been a fact easier to reconcile with evolution than with creation.' (p. 166)

Software system evolution has analogies to this. We are often forced to keep data codes, report formats, programming language artifacts and other structures which have long since lost their present and future meaning, but which were necessary at some time in the past of the evolution of the system. They remain when the damage they do is less than the cost and potential damage done if we get rid of them, or if they are invisible and not the cause of serious problems.

A book such as this which takes the entire concept of evolution up to lengthy debate is rich in concepts which could be of interest to systems engineers. But, I hope that the above samples show some of the thinking around the conventional biological evolutionary world which might give us some insights about the software evolutionary world.

15.5.2 Franklin via Tuchman

Source: in Barbara Tuchman, The March of Folly, Abacus, 1984, ii. 248

'Benjamin Franklin, a wise man and one of the few who derived principles from political experience and were able to state them, wrote during the Stamp Act crisis that it should not be supposed that honor and dignity are better served "by persisting `in a wrong measure once entered into than by rectifying an error as soon as it is discovered."'`

================================== end of Chapter
This Chapter is posted at gilb.com/downloads papers 20 Nov 2012, as is Chapter 14 on Productivity Posem Ch 14
http://www.gilb.com/dl560

======================== Here are some Agile references to Mywork ======

Tom

Tom@Gilb.com, www.gilb.com, Cell +47 9206 6705 Twitter @imtomgilb. , Hon. FBCS

Agile References:

"Tom Gilb invented Evo, arguably the first Agile process. He and his son Kai have been working with me in Norway to align what they are doing with Scrum.

Kai has some excellent case studies where he has acted as Product Owner. He has done some of the most innovative things I have seen in the Scrum community."

Jeff Sutherland, co-inventor of Scrum, 5Feb 2010 in Scrum Alliance Email.

"Tom Gilb's Planguage referenced and praised at #scrumgathering by Jeff Sutherland. I highly agree" Mike Cohn, Tweet, Oct 19 2009

"I've always considered Tom to have been the original agilist. In 1989, he wrote about short iterations (each should be no more than 2% of the total project schedule). This was long before the rest of us had it figured out." Mike Cohn http://blog.mountaingoatsoftware.com/?p=77

Comment of Kent Beck on Tom Gilb's book , "Principles of Software Engineering Management": " A strong case for evolutionary delivery – small releases, constant refactoring, intense dialog with the customer". (Beck, page 173).

In a mail to Tom, Kent wrote: “I'm glad you and I have some alignment of ideas. I stole enough of yours that I'd be disappointed if we didn't :-), Kent” (2003)

"But if you really want to take a step up, you should read Tom Gilb. The ideas expressed in **Principles of Software Engineering Management** aren’t quite fully baked into the ADD-sized nuggets that today’s developers might be used to, but make no mistake, Gilb’s thinking on requirements definition, reliability, design generation, code inspection, and project metrics are beyond most current practice."   Corey Ladas
http://leansoftwareengineering.com/2007/12/20/tom-gilbs-evolutionary-delivery-a-great-improvement-over-its-successors/

Jim Highsmith (an Agile Manifesto signatory) commented: “Two individuals in particular pioneered the evolution of iterative development approached in the 1980’s – Barry Boehm with his Spiral Model and Tom Gilb with his Evo model. I drew on Boehm’s and Gilb’s ideas for early inspiration in developing Adaptive Software Development. …. Gilb has long advocated this more explicit (quantitative) valuation in order to capture the early value and increase ROI” (Cutter It Journal: The Journal of Information Technology Management, July 2004page 4, July 2004).

Ward Cunningham wrote April 2005: “Tom -- Thanks for sharing your work. I hope you find value in ours. I'm also glad that the agile community is paying attention to your work. We know (now) that you were out there ahead of most of us. Best regards. – Ward”, http://c2.com

Robert C. Martin (Agile Manifesto initial signatory, aka Uncle Bob): "Tom and I talked of many things, and I found myself learning a great deal from him. The item that sticks most prominently in my mind is the definition of progress.", "Tom has invented a planning formalism that he calls Planguage that captures this idea of customer need. I think I'm going to spend some serious time investigating this. " from
http://www.butunclebob.com/ArticleS.UncleBob.TomGilbVisit

'1985: perhaps the first explicitly named, incremental alternative to the “waterfall” approach is Tom Gilb’s Evolutionary Delivery Model, nicknamed “Evo” '

http://guide.agilealliance.org/timeline.html

Gilb T. (1985). "Evolutionary Delivery versus the "waterfall model" " ACM SIGSOFT, http://dl.acm.org/citation.cfm?id=1012490